A Project Report

on

AUTOMATED SURVEILLANCE AND ALERT GENERATION SYSTEM

Submitted for partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

BY

KAREDLA ANANTHA SASHI SEKHAR (160114733091)

DASARADA RAM REDDY MUDIAM (160114733092)

Under the guidance of

Mrs. E. Padmalatha

Assistant Professor

Department of Computer Science

Department of Computer Science and Engineering

Chaitanya Bharathi Institute of Technology(A)

Hyderabad, India - 500 075

April - 2018

Chaitanya Bharathi Institute of Technology (Autonomous)
(Regd. No. 855/2009)
(Affiliated to O.U; All U.G and 5 P.G. Programmes (Civil, CSE, ECE, Mech. & EEE) Accredited by NBA; Accredited by NAAC (UGC); ISO Certified 9001 : 2008)
Chaitanya Bharathi P.O., CBIT Campus, Gandipet, Kokapet (V),
Gandipet Mandal, Ranga Reddy District, Hyderabad - 500 075, Telangana
e-mail : principal@cbit.ac.in; Website: www.cbit.ac.in ✆ : 040 - 24193276, 277, 280, Fax: 040 - 24193278

CERTIFICATE

This is to certify that the project work entitled Automated Surveillance and Alert Generation System is a bonafide work carried out by Karedla Anantha Sashi Sekhar (160114733091) and Dasarada Ram Reddy Mudiam (160114733092) in partial fulfilment of the requirements for the award of Degree of BACHELOR OF ENGINEERING in COMPUTER SCIENCE AND ENGINEERING, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any Degree or Diploma.

Project Guide
Mrs. E. Padmalatha
Assistant Professor
Department of CSE
CBIT, Hyderabad.

Head of Department
Dr. M Swamy Das
Professor and
Head Department of CSE
CBIT, Hyderabad.

Declaration

This is to declare that the work reported in the present project entitled Automated Surveillance and Alert Generation System is a record of work done by us in the Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology. The reports are based on the project work done entirely by us and not copied from any other source.

Sekhar Karedla
160114733091

Dasarada Mudiam
160114733092

Acknowledgments

We would like to express our deep-felt appreciation and gratitude to **Mrs. E Padmalatha**, our project guide, for her skilful guidance, constant supervision, timely suggestion, keen interest and encouragement in completing the individual seminar within the stipulated time.

We wish to express our gratitude to **Dr. T Sridevi** and **Mr. J Shiva Sai**, Project Coordinators, who have shown keen interest and even rendered their valuable time and guidance in terms of suggestions and encouragement.

We are honoured to express our profound sense of gratitude to **Dr. M Swamy Das**, Head of Department, CSE, who has served as a host of valuable corrections and for providing us time and amenities to complete this project.

We gratefully express our thanks to **Dr. P Ravinder Reddy**, Principal of our college and the management of CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY for providing excellent academic and learning environment in the college.

We wish to express our heartfelt gratitude to the Members of Staff and all others who helped us in bringing up our project. We would also like to thank the Lab assistants and Programmers for helping us through our project.

Karedla Anantha Sashi Sekhar

160114733091

Dasarada Ram Reddy Mudiam

160114733092

Abstract

A large number of people gathered together and arranged in an unruly manner is known as a crowd. Monitoring the events taking place in the presence of crowd is of utmost importance. Today it is done manually by a human surveyor with the help of CCTV cameras. Our idea is to reduce the burden on the human surveyor by automating the process of detecting disturbance in the crowd. Critical amount of time may be saved by detecting disturbance in crowd instantaneously. This critical time we save may be the difference between life and death. This disturbance may be caused by any event such as bomb blast, fire, riots etc. This project focuses on implementation of an algorithm that can detect disturbance in the crowd. It considers flow-vector magnitudes change over time for a short set of frames to statistically determine whether those short set of frames are violent or non-violent. The challenge in this project is to keep the processing quick and real time, an alert should be generated within few seconds of change in crowd behaviour.

List of Figures

2.1 Occlusion Problem 6

2.2 Crowd Running in Marathon 7

2.3 Crowd Running in a Mall 7

3.1 System Design 11

3.2 Dataset 13

3.3 Data Flow Diagram Level 0 of the proposed system 14

3.4 Data Flow Diagram Level 1 of the proposed system 15

3.5 Component Diagram of the proposed system 17

4.1 Frames before and after preprocessing respectively 27

4.2 Example Optical Flow 28

4.3 Neural Net Layer Information 30

5.1 Frames before and after preprocessing 31

5.2 Optical Flow Car Example 32

5.3 Accuracy of Neural Net by 70:30 method 39

5.4 Neural Net Folds Accuracy 41

5.5 Surveillance on Video 42

5.6 No Violence 42

5.7 Violence about to start 43

5.8 Slight Violence 43

5.9 Furious Violence 43

5.10 Surveillance output of video in Terminal 44

5.11 Surveillance through Camera 45

List of Tables

5.1 Confusion Matrix for 70:30 dataset 39

Contents

Certificate i

Declaration ii

Acknowledgements iii

Abstract iv

List of Figures v

List of Tables vi

1 Introduction 1

1.1 Scope of Project 1

1.2 Motivation 1

1.3 Existing System 2

1.3.1 Problems in Existing System 2

1.3.2 Proposed System 3

1.4 Organisation of Project Report 3

2 Literature Survey 4

2.1 What is a video 4

2.2 What is Video Processing 4

2.3 What is crowd 5

2.3.1 Analysis of Crowd 5

2.3.2 Challenges faced in crowd behaviour analysis 6

2.4 Optical Flow 8

2.4.1 Estimation 8

2.5 Violent Flow(ViF) Descriptors 9

2.6 Oriented Violent Flows(OViF) 9

3 Methodology and Design 11
3.1 System Design 11
3.2 Implementation of Proposed Solution 12
3.2.1 Input Footage 12
3.2.2 Preprocessing 12
3.2.3 Flow Vector Magnitude 12
3.2.4 Feature Extraction 12
3.2.5 Classification 13
3.2.6 Detection 13
3.3 Dataset 13
3.4 Data Flow Diagram 14
3.5 UML Diagrams 15
3.5.1 Building blocks of UML 15
3.5.2 Things in UML 16
3.5.3 Relationships in the UML 16
3.5.4 Diagrams in UML 16
3.5.5 Component Diagram 17
3.6 Requirements, Analysis and Specifications 19
3.6.1 Unix 19
3.6.2 Python 20
3.6.3 OpenCV 22
3.6.4 Hardware Requirements 23
3.6.5 Keras and TensorFlow 24
3.6.6 Others 25

4 Implementation 27
4.1 Video Preprocessing 27
4.2 Optical Flow 28
4.3 Violent Features 28
4.4 Training Neural Net 29
4.5 Violence Detection 30

5 Results and Discussions 31
5.1 Video Preprocessing 31
5.2 Optical Flow 32
5.3 Violent Features 32
5.4 Neural Net Accuracy Testing 39
5.5 Neural Net N-folds Cross Verification 40
5.6 Real Time Surveillance on Video Input 41
5.7 Real Time Surveillance on Camera Input 44

6 Conclusion and Future Work 46
6.1 Conclusion 46
6.2 Future Work 46

References 47

Appendix A Appendix 49
A.1 Video Preprocessing 49
A.2 Optical Flow 51
A.3 Generate ViFs for Violent Videos 51
A.4 Generate ViFs for Non Violent Videos 52
A.5 Training SVM and Testing Accuracy 52
A.6 SVM N-folds Cross Verification 54
A.7 Training Neural Net and Testing Accuracy 56
A.8 Training Neural Net and Storing on disk 58
A.9 Load Neural Net from Disk and Visualize 61
A.10 Load Neural Net and perform N-Folds Cross Verification 61
A.11 Surveillance System Working Class 64
A.12 Surveillance System Main File 67

1. Introduction

Automation in Real-Time analysis of crowd can make surveillance more efficient. In this modern era, the number of cameras for surveillance is continuously increasing which leads to increase in burden on the human. Real Time alert generation is a system that can analyse abnormalities accurately in real time and create an alert. These automatic alerts may help to proact rather than to react. This time we save between time of occurance and time of reaction may be the difference between life and death.

This automation is of much importance and very less attention has been given to it the past. The system being proposed will have the ability to detect violence in real time with high accuracy. There is always a limit to the human capacity, it is basic human tendency to gradually perform less on a repetitive boring job. Even today, most of the repetitive jobs such as assemble floor working are being assigned to computers and robots. There is no reason we should not assign the job of surveillance to computers too.

1.1 Scope of Project

Public safety is an important concern for any organization. So as to achieve that an organization takes the help of CCTV cameras. With the decrease in price of cameras, amount of information generated in terms of cctv footage is huge. Real time processing of this footage is required so that the burden on the human surveyors may be decreased.

Aim of this project is to implement an algorithm that can process the incoming footage in real time and detect disturbance within few seconds of an abnormal activity. Crucial amount of time will be saved if an alert is generated. This time may be the difference between the life and death of a person.

1.2 Motivation

Crowd can be defined as a large number of people in close proximity to each other. Whenever an abnormal event happens in crowd, all the people present in the crowd react to that at once. This gives us opportunity to detect violence in crowd, if we detect that exact moment where the abnormal activity happens.

If an alert is generated during the exact moment when the outburst takes place, it may be used to alert the local bodies such as riot control team to take control of the situation. It would be highly beneficial for us to detect violence at the moment it occurs rather than reacting to the incident later on.

1.3 Existing System

Now-a-days every public area will have CCTV coverage so as to protect the public. In the existing manual surveillance system, a human surveyor continuously pays attention to screens. As the number of cameras increase burden on the human will also increase. This system is laggy and it may or may not detect every outbreak.

Violence detection is a part of Action Recognition. There has been intensive research on action recognition in the past. Much research has been done in person to person fight detection, sports violence detection, violence detection in movies and slow motion fight detection. Violence detection in crowd is one of the most trending topics in the area of action recognition.

Existing Person-to-Person fight detection[5] takes heavy computational power and cannot be refined to be used in real-time detection. Blob method used is quick and requires less computational power but it can be used for only Person-to-Person fight detection. We cannot refine it to be used for crowd violence detection.

1.3.1 Problems in Existing System

- As the number of cameras increase burden on the human will also increase. This system is laggy and it may or may not detect every outbreak.
- Existing Person-to-Person fight detection[5] takes heavy computational power and cannot be refined to be used in real-time detection.
- Blob method used is quick and requires less computational power but it can be used for only Person-to-Person fight detection, we cannot refine it to be used for crowd violence detection.

1.3.2 Proposed System

- We propose an automated system to detect and generate alert in real time incase outbreak of violence in crowd using Video Processing, Optical Flow and Violent Flow Descriptors(ViF).
- In the proposed system, surveillance videos are taken as input and output is detection of violence(if present) in the video.
- Our system works in real time i.e it detects disturbance or violence in crowd present in the video within milliseconds of outset of violence.
- The Real time detection of violence helps to proact rather than to react.

1.4 Organisation of Project Report

This project report is mainly divided into 9 modules as follows:

- The first chapter deals with the Introduction of the Project and explanation about some basic concepts.
- The second chapter discusses about the literature survey of this project which includes an insight into the core concepts of our project.
- Third chapter deals with the Hardware and Software requirements of the project.
- Fourth chapter gives the methodology of the project, the way proposed algorithm is generated.
- Fifth chapter gives details of the "in the wild" dataset.
- The sixth chapter deals with the design of our proposed system.
- The seventh chapter deals with the implementation of our system which discusses about the algorithms used in building our system.
- The eighth chapter displays our results and discussions through a series of screenshots.
- From ninth chapter onwards details about the conclusions, future scope of our project and the limitations.

2. Literature Survey

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy and company strength. Once these things are satisfied, then next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool, the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system above considerations are taken into account for developing.

2.1 What is a video

Video is a sequence of frames that has fixed frame width and height throughout its duration. Video is an electronic medium for the recording, copying, playback, broadcasting, and display of moving visual media.

Video initially was supposed to be played in Mechanical TVs, with the advancement in technology , Flat Screened LEDs and LCDs are also available. Video have various properties such as display resolution, aspect ratio, refresh rate, color capabilities and other qualities. Analog and digital formats exist and can be carried on a variety of media such as radio broadcast, magnetic tape, optical discs, computer files, and network streaming.

2.2 What is Video Processing

In pure electrical engineering, video processing is something that is done through input analog signal manipulation through filters and other inputs.

Video processing techniques are used in devices such as:

- television sets,
- VCRs,
- DVDs,
- video codecs,

- video players,
- video scalers and other devices.

2.3 What is crowd

A crowd is a large group of people that are gathered or considered together. The term "the crowd" may sometimes refer to the lower orders of people in general (the mob). A crowd may be definable through a common purpose or set of emotions, such as at a political rally, a sports event, or during looting (this is known as a psychological crowd), or may simply be made up of many people going about their business in a busy area.

The term crowd is sometimes defined in contrast to other group nouns for collections of humans or animals, such as aggregation, audience, group, mass, mob, populous, public, rabble and throng. Opinion researcher Vincent Price compares masses and crowds, saying that "Crowds are defined by their shared emotional experiences, but masses are defined by their interpersonal isolation."

2.3.1 Analysis of Crowd

Crowd Analysis[6] is an important part aspect real time surveillance since rarely we see CCTV cameras in private areas. Areas under surveillance(public area) usually have a lot of crowd. Crowd analysis involves the analysis of data gained by studying the natural movement of groups. Masses of bodies, particularly humans, are under observation of these crowd tracking analyses that include how a particular a group of people move and when a movement pattern changes. The data is implemented in order to predict future crowd movement, crowd density, and potential events such as an evacuation route. Applications of crowd analysis can range from video game crowd simulation to security and surveillance.

Due to exponential population growth, crowd analysis has become a major interest in social and technical areas of study. Crowd analysis is being used to develop stratergies that will help us to perform a safe evacuation and a quicker response in times of distress.

2.3.2 Challenges faced in crowd behaviour analysis

Crowd analysis is a critical problem in understanding crowd behavior for surveillance applications. The current method is manually screening video feeds from several sources. Video analytics allows the automatic detection of events of interest, but it faces many challenges because of highly unpredecitable crowd motions and occlusions.

Video analysis and scene understanding usually involve object detection, tracking and behavior recognition. For crowded scenes, due to extreme distortions, severe occlusions and ambiguities, the conventional methods without special considerations are not appropriate. This makes it very challenging to figure out an appropriate level of granularity to model the dynamics of a crowd. Another challenge in crowded scene analysis is that the specific crowd behaviors needed to be detected and classified may be both rare and subtle , and in most surveillance scenarios, these behaviors have few examples to learn. The problem of occlusion is one of the main reasons why computer vision is hard in general. Specifically, this is much more problematic in Object Tracking. Occlusion means that there is something you want to see, but can't due to some property of your sensor setup, or some event. For tasks which tracks objects (people, cars, ...) then occlusion occurs if an object that is being tracked is hidden (occluded) by another object. Like two persons walking past each other as shown in the Fig. 2.1, or a car that drives under a bridge. The problem in this case is what you do when an object disappears and reappears again.

Figure 2.1: Occlusion Problem

Crowd scenes contain some uncertainities such as change of density, shape, boundaries of the crowd. They do not define how to behave or share clear expectations on what will

happen. They often feel something must be done right away to address their common concern. Attitudes and ideas about the common concern spread very quickly among crowd members. They often do and say things that they would normally not do, and they go along with the actions of others in the crowd.

Certain crowd behaviours are normal in one scenario but may become hazards in another. For example as shown in Fig. 2.2, running in a marathon can be considered as a normal crowd behaviour but whereas running in a shopping mall as in Fig. 2.3 or any such rare places can be considered as an abnormal behaviour.

Figure 2.2: Crowd Running in Marathon

Figure 2.3: Crowd Running in a Mall

2.4 Optical Flow

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. The concept of optical flow was introduced by the American psychologist James J. Gibson in the 1940s to describe the visual stimulus provided to animals moving through the world. Gibson stressed the importance of optic flow for affordance perception, the ability to discern possibilities for action within the environment.

Followers of Gibson and his ecological approach to psychology have further demonstrated the role of the optical flow stimulus for the perception of movement by the observer in the world; perception of the shape, distance and movement of objects in the world; and the control of locomotion. The term optical flow is also used by roboticists, encompassing related techniques from image processing and control of navigation including motion detection, object segmentation, time-to-contact information, focus of expansion calculations, luminance, motion compensated encoding, and stereo disparity measurement.

2.4.1 Estimation

Sequences of ordered images allow the estimation of motion as either instantaneous image velocities or discrete image displacements. Fleet and Weiss provide a tutorial introduction to gradient based optical flow. John L. Barron, David J. Fleet, and Steven Beauchemin provide a performance analysis of a number of optical flow techniques. It emphasizes the accuracy and density of measurements.

The optical flow methods try to calculate the motion between two image frames which are taken at consecutive time frames at every voxel position. These methods are called differential since they are based on local Taylor series approximations of the image signal; that is, they use partial derivatives with respect to the spatial and temporal coordinates.

For a 2D+t dimensional case (3D or n-D cases are similar) a voxel at location (x,y,t) with intensity I(x,y,t) will have moved between the two image frames, and the following brightness constancy constraint can be given:

$$I(x, y, t) = T(x + \Delta x, y + \Delta y, t + \Delta t) \tag{2.1}$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{2.2}$$

2.5 Violent Flow(ViF) Descriptors

For a sequence of frames, Violent Flows(ViF) Descriptors[2] are produced by first estimating Optical Flow. Optical Flow[3] gives a flow vector of a pixel in current frame matching it with a pixel in next frame. Magnitudes of these flow vectors are calculated which are arbitrary components which are dependent on frame resolution, different motions in different spatio-temporal locations, etc. By comparing magnitudes, meaningful measures of the significance of observed motion magnitudes in each frame compared to its predecessor are obtained. Binary indicators for each pixel is obtained by comparing change of magnitudes of each pixel with mean change of magnitudes of a frame. Average of these binary indicators are calculated for each pixel over all frames. Later these average values of binary indicators are divided into M * N non-overlapping cells in which each cell contain magnitude change frequencies. Those frequencies in the cells are represented by fixed-sized histogram and all the histograms are concantenated to form a single descriptor vector. In a simple way, ViF descriptor is a vector of frequencies quantized values of binaty indicators.

2.6 Oriented Violent Flows(OViF)

ViF[2] doesn't consider difference in direction of the same pixel in sequential frames and treat there is no difference if pixels differ only in directions. OViF[1] considers direction of the pixel and it is calculated by using

$$\phi_{i,j,t} = arctan(V^y_{i,j,t}/V^x_{i,j,t}) \tag{2.3}$$

Here, t means the t-th frame in a video sequence, and (i, j) indicates the pixel location. Then, for each flow map which corresponds to a frame, we partition it into M N non-overlapping blocks. After this, since 360^0 can be equally divided into B sectors and each sector corresponds to a bin of a histogram, the flow-vector magnitude $|V_{i,j,t}|$ is added into the bin where the flow-vector angle $\phi_{i,j,t}$ locates.For each block, we get a histogram and these histograms are then concatenated into a single vector H, which is called Histogram

of Oriented Optical Flow (HOOF)[1] with X-dimensions, $X = M \times N \times B$.

Subsequently, the HOOF vectors are used to obtain binary indicators:

$$b_{x,t} = \begin{cases} 1, & if \quad |H_{x,t} - H_{x,t-1}| \geq \theta \\ 0 & otherwise \end{cases} \tag{2.4}$$

Here, x is the x-th dimension of the feature vector H, and θ is the average value of $|H_{x,t} - H_{x,t-1}|$, x$\subseteq$[1,X]. The above equation explicitly reflects the magnitude changes in different sectors. The mean magnitude-change vector is donated as:

$$\bar{b}_x = \frac{1}{T} \sum_t b_{x,t} \tag{2.5}$$

Finally, b is the final OViF vector for a sequence of frames, which counts the motion magnitude change frequencies in both direction sectors and spatial regions.

3. Methodology and Design

Methodology is the systematic, theoretical analysis of the methods applied to develop a project. It comprises the theoretical analysis of the body of methods and principles associated with a branch of knowledge. Typically, it encompasses concepts such as paradigm, theoretical model, phases and quantitative or qualitative techniques.

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

3.1 System Design

The project focuses on the implementation of an algorithm that can detect disturbance in crowd. The design of the project is shown in Fig. 3.1. It considers how flow-vector magnitudes change over time, which are collected for short frame sequences, and then classified as either normal or abnormal situation using a classifier.

The aim of the project is to keep the processing very quick, a detection should be made within a few seconds of the outbreak of violence. It has to detect the change of normal behaviour to abnormal behaviour with the shortest delay from the time that the change has occurred.

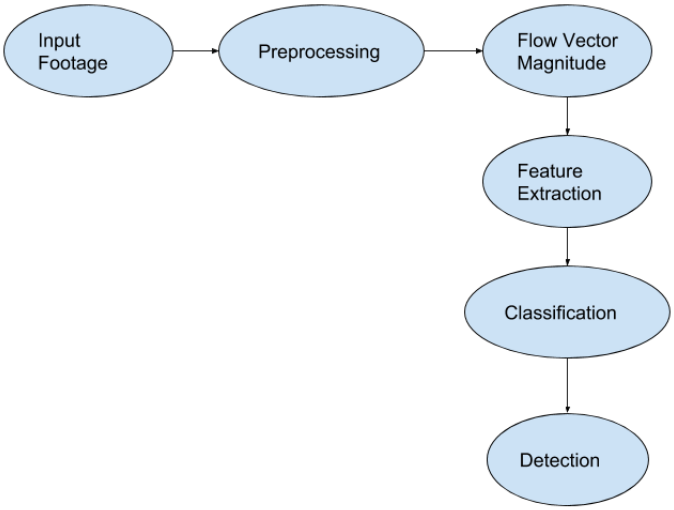

Figure 3.1: System Design

3.2 Implementation of Proposed Solution

3.2.1 Input Footage

For training of the classifier, input of the footage will be from a fixed dataset. For practical real time implementation, footage from externally attached cameras can be used. The resolution of the input footage need not be specific. Generally the resolution of a CCTV footage is of 704 X 480 pixels. This proposed algorithm reduces the height of the video to 100 pixels and width accordingly. This increases the scalability of the algorithm. Even high definition videos can be processed with the proposed algorithm.

3.2.2 Preprocessing

Preprocessing the input data is very important part of the algorithm. Optical flow algorithm will take at least one minute to calculate optical flow of a high definition image. So as to constraint the processing to real time, preprocessing must done. As mentioned earlier any image is being resized to 100 pixels height and respective width accordingly. Preprocessing is done in a way such that the aspect ratio of the video is not disturbed.

3.2.3 Flow Vector Magnitude

Each pixel in the video has some velocity corresponding to it, along x direction and y direction. The Flow Vector Magnitude is nothing but the magnitudes of these velocities. Computing Flow Vector Magnitude is computationally heavy and this is the most time taking part of the proposed algorithm. Since we are reducing the size of frames considerably, this computation is quick enough to cope us with real time violence detection.

3.2.4 Feature Extraction

Feature extraction is based on the computation of ViF (Violence Flow Descriptors)[2]. These descriptors are quantised values of the above generated Flow Vector Magnitudes. After ViF is generated a histogram is built which will assign different ViF values to corresponding bins in the range of 0 - 1.0 with the intervals of 0.05, i.e 20 bins.

3.2.5 Classification

After the features have been extracted, classification can done with the help of a simple Linear SVM. So as to further increase the accuracy of the algorithm , SVM with AdaBoost can be used. For our project a trained neural net has been used, since it is providing a better accuracy.

3.2.6 Detection

Detection in a video footage is the final output of the proposed algorithm. It is taking place with the help classifier model that is being trained in the above step. According to the base paper, a sequence of 10 frames i.e on an average two-fifth of a second is enough to detect violence in a footage. Detection may be further improved by continuous training of neural net in real-time also.

3.3 Dataset

In-the-wild violence dataset is chosen as dataset for the project. The dataset consists of 246 videos in which half are violent and other half are non-violent. The video duration range from 1.04 sec to 6.52 sec with an average duration of 3.60 sec as shown in Fig. 3.2. All the videos are downloaded from YouTube which are produced under in-the-wild and uncontrolled conditions presenting a wide range of real-world viewing conditions, video qualities and surveillance scenarios. Videos are compressed using the DivX codec (mpeg4) and resized to 240 X 320 pixels.

General statistics:	
# of videos	246
# unique urls	214
# unique YouTube titles	218
Video statistics:	
Shortest video duration	1.04 sec.
Longest video duration	6.52 sec.
Average video duration	3.60 sec.

Figure 3.2: Dataset

3.4 Data Flow Diagram

A data flow diagram is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFD's can be used for the visualization of data processing. Level-0 DFD of the proposed system is shown in the Fig. 3.3.

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to system design. This context-level DFD is next "exploded", to produce a level-1 DFD that shows some of the detail of the system being modeled. The Level-1 DFD shows how the system is divided into subsystems, each of which deals with one or more of the data flows to from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system. Level-1 DFD of the proposed system is shown in the Fig. 3.4.

Figure 3.3: Data Flow Diagram Level 0 of the proposed system

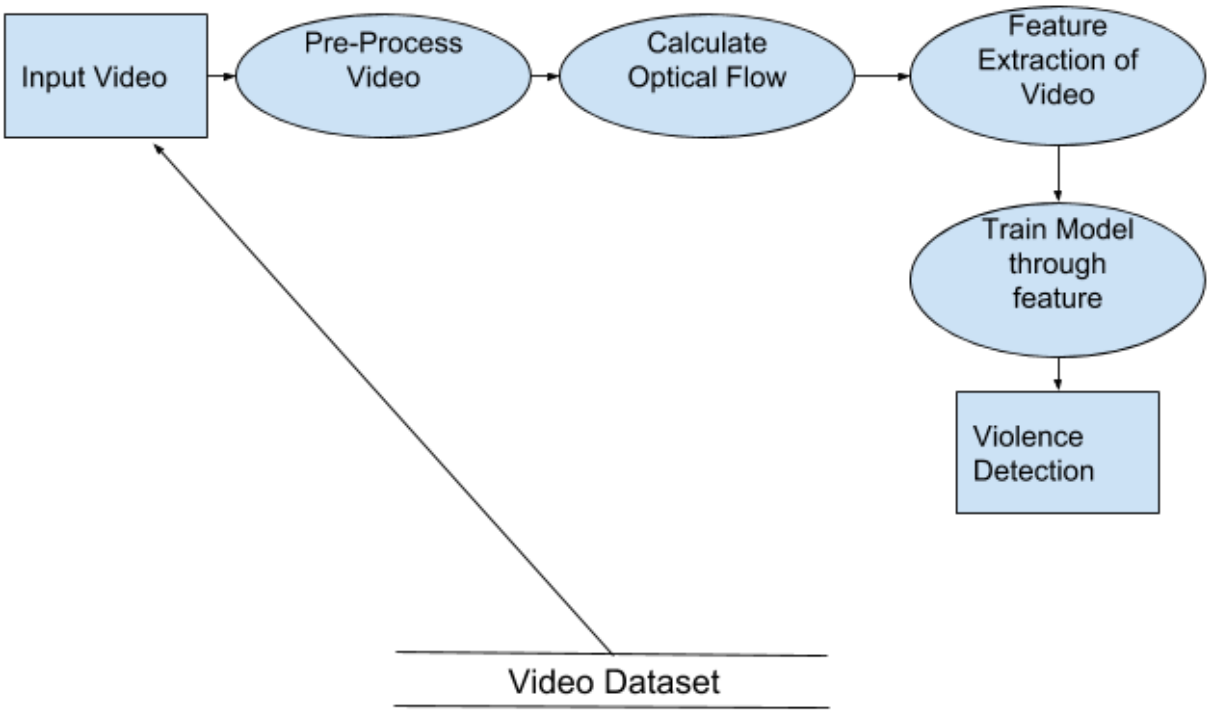

Figure 3.4: Data Flow Diagram Level 1 of the proposed system

3.5 UML Diagrams

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

3.5.1 Building blocks of UML

The vocabulary of the UML encompasses three kinds of building blocks.

1. Things.
2. Relationships.
3. Diagrams.

3.5.2 Things in UML

Things are the abstractions that are first-class citizen in a model.
There are four kinds of things in the UML.

1. Structural things.
2. Behavioural things.
3. Grouping things.
4. Annotational things.

These things are the basic object-oriented building blocks of the UML. We use them to write well-formed models.

3.5.3 Relationships in the UML

Things can be connected to logically be physically with the help of relationship in object oriented modelling.These are four kinds of relationships in the UML.

1. Dependency.
2. Association.
3. Generalization.
4. Realization.

3.5.4 Diagrams in UML

A diagram is a graphical representation of a set of elements. These are nine kinds of diagrams in the UML.

1. Class diagram.
2. Object diagram
3. Use Case diagram.
4. Sequence diagram.

5. Collaboration diagram.
6. Activity diagram.
7. Component diagram.
8. State chart diagram.
9. Deployment diagram.

3.5.5 Component Diagram

Component diagram of the proposed system is shown in Fig. 3.5. It has the following components:

1. OpenCV
2. Bob
3. Video Preprocess
4. Optical Flow
5. Violent Feature Extract

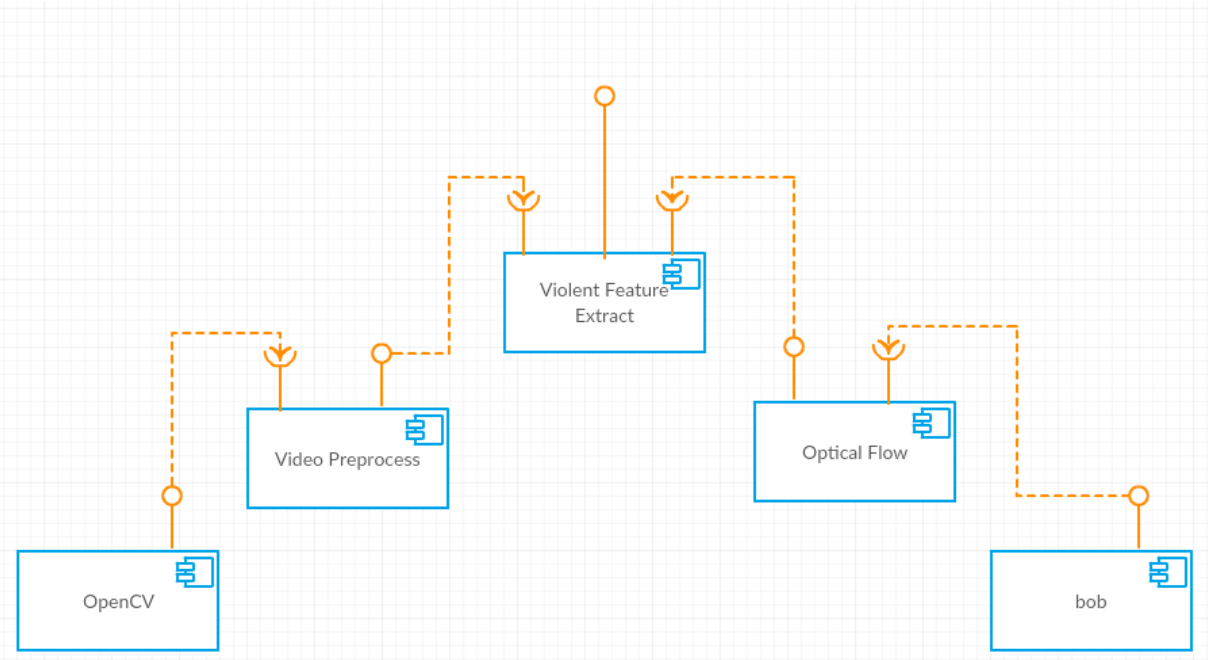

Figure 3.5: Component Diagram of the proposed system

OpenCV

OpenCV (short for Open Computer Vision)[8] is a package originally written in C language but later it was ported to Python. OpenCV possesses a rich set of interfaces and functions that help us to read and manipulate video files. OpenCV can read video from input cameras and also from attached cameras to the system. Given a source of CCTV footage , through OpenCV we are able to manipulate frame size, the colour and the frame intervals. Video Preprocessing has to be done so that the further components can work at real-time which is on and average 1/25 th of a second for a frame.

Bob

Bob[4] is a signal processing and machine learning platform available for python. Bob is provided as a precompiled source for Linux or Mac OS through Anaconda package manager. Bob helps us to port the signal processing procedures which are initially written in C language. Signal processing methods are usually written in C language as it can make native function calls and kernel function calls. So as to port these procedures into Python bob platform is used. C Liu's Optical Flow algorithm[5] is being ported here through bob.

Video Preprocess

This is a self written Python library. It Preprocesses the video according to our needs. This library makes the necessary calls from the OpenCV so as to do the pixel level manipulations. Frame by Frame access is done through this package. Frame interval is set as 3, default frame rate is taken as 25, each frame is resized to 100 pixel width and corresponding height. Frame is further converted into grayscale.

Optical Flow

This is also a self written package. It contains the procedures to calculate Optical flow[3] which further call procedures from bob platform. Optical flow will return 3 things in a tuple, velocity along x axis , y axis and the wrap. This calculation is done by considering some pre calculated parameters.

Video Feature Extract

At a considered moment, three frames are under consideration, previous frame , current frame and the next frame. First thing we do is we preprocess these frames and then calculated optical flows between successive frames. Now we have two optical flow vectors, now we calculate the absolute change between these two vectors. Average of this change vector is calculated and it is stored as threshold. Now the change vector is quantised with binary 1 and 0 by comparing it with threshold vector. This binary vector is summed for the whole video and normalized by dividing it with the number of iterations done till now. Binary vector generated till now is divided into (4 * 4) parts. For each of these parts , a histogram is built which has the bin size of 0.05 ranging from 0 to 1. Frequency of each bin is calculated and normalized by dividing with sum of all frequencies. Now all of the normalized histograms are appended one after another and the resulting vector is known as *Violent Feature Vector*.

3.6 Requirements, Analysis and Specifications

3.6.1 Unix

Unix is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, development starting in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. Unix was originally meant to be a basic platform for programmers developing software to be run on it and on other systems, rather than others. The system grew larger as the operating system started spreading in academic circles, as users added their own tools to the system and shared them with colleagues. The best feature of linux is that it is open source. Being open source makes anyone in the world to contribute to it. This makes unix highly robust and stable.

- Kernel - /usr/sys which contains the sub-components.
- Development Env - Linux code itself can be built in the linux. Provdes great environment for the programmers.
- Commands in Linux -
  - for system operation and maintenance,

  - commands of general utility
  - general-purpose applications
  - text formatting and
  - typesetting package.
  - Document formatting - Unix systems were used for document preparation and typesetting systems, and included many related programs such as nroff, troff, tbl, eqn, refer, and pic. Some modern Unix systems also include packages such as TeX and Ghostscript.
  - Graphics - the plot subsystem provided facilities for producing simple vector plots in a device-independent format, with device-specific interpreters to display such files. Modern Unix systems also generally include X11 as a standard windowing system and GUI, and many support OpenGL.
  - Communications - early Unix systems contained no inter-system communication, but did include the inter-user communication programs mail and write. V7 introduced the early inter-system communication system UUCP, and systems beginning with BSD release 4.1c included TCP/IP utilities.

- Multiple Development Environments
- Multithreaded Programming
- Manipulation of Device Drivers
- Support for latest languages and packages

3.6.2 Python

Only open source languages and tools are used in developing the project. Unix environment is being used for the development of the process. Language used is Python and the main tool used is OpenCV for video processing. Python is interpreted language which is used for general-purpose programming. It is a user-friendly language which emphasizes on code readability and its syntax allows users to write programs with relatively fewer lines of code when compared to C/C++, Java etc. Python 2.7 version is used in the project.

Python's **features** include:

- East Learning: Python has less keywords, simple structure, and a clearly defined syntax. This allows to pick up the language quickly.
- Scripting Language: Python code is more clearly defined and visible to the eyes. Since it is scripting language it can be easily read.
- Maintenance: Python code is fairly easy-to-maintain.
- A broad collection of standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh. It uses the pip package for maintenance and installation of those standard packages.
- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code through tools such as iPython notebooks.
- Portability: Python can run on a wide variety of hardware and has the same interpretation on all platforms.
- Extendable Interpretation: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Database Support: Python provides interfaces to all commercial databases such as Oracle, MySQL.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalability: Python programs are highly scalable. They provide better structure and support than Shell Scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

3.6.3 OpenCV

OpenCV (Open Source Computer Vision Library)[8] is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCVs deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

OpenCV is written in C++ and its primary interface is in C++, but it still retains

a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C-sharp, Perl, Ch, Haskell and Ruby have been developed to encourage adoption by a wider audience. All the new developments and algorithms in OpenCV are now developed in the C++ interface.
There are many **applications** of OpenCV. A few of them are cited as below:

- 2D and 3D feature toolkits
- Facial recognition system
- Gesture recognition
- Human Computer Interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure From Motion (SFM)
- Motion tracking

3.6.4 Hardware Requirements

- Processor - Intel Core i5 5th gen
- RAM - 8 GB
- Hard Disk - 500 GB
- OS - Unix based such as Debian , MacOS
- WebCam
- USB 2.0 Ports

3.6.5 Keras and TensorFlow

Keras

Keras[7] is a high-level neural networks API, written in Python and capable of running on top of **TensorFlow, CNTK, or Theano**. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.
Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Keras is compatible with: **Python 2.7-3.6.**
Guiding Principles:

- **User friendliness**. Keras[7] is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity**.A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility**. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

TensorFlow

TensorFlow[9] is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks.

TensorFlow is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization for the purposes of conducting machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

3.6.6 Others

Libraries

Along with OpenCV we have used bob. Bob[4] is a machine learning platform used in python, it helped us to port Matlab code of optical flow to python. Scikit learn is Python package which is similar to WEKA tool for java. It will be used for training a classifier model in the proposed algorithm of the project.

Editor

An editor is required to create python source files and to view violent features generated for the dataset. Open source editors like Atom and PyCharm have been used which also provide terminal access.

Video Player

To play all the videos through openCV and to view some random videos. Video players like VLC is required which is open source. ffmpeg library is also required which provides a way to read video files through openCV. ffmpeg can also be used to format videos.

4. Implementation

Implementation involves actual development process of the proposed system.

4.1 Video Preprocessing

Surveillance footage is usually generated of size 240 x 320 i.e of aspect ratio of 3:4. Considered video format is of avi. If the video is not present in the given format, we use the ffmpeg command to convert the video to the required format.

Video conversion command:-

ffmpeg -i inputVideo -vf scale=320:240 outputVideo.avi

Further the frame is resized to 75 x 100 size as shown in the Fig. 4.1 maintaining the same aspect ratio using openCV functions. This resized frame is further converted to grayscale using openCV[8] cvtColor() method.

Size reduction command:

frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)

Grayscale conversion commands:

frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

Figure 4.1: Frames before and after preprocessing respectively

4.2 Optical Flow

Optical Flow is estimated between the pair of consecutive frames which gives a flow vector for each pixel in the current frame, matching it to pixel in next frame as shown in the Fig. 4.2. C. Liu's Optical Flow algorithm[3] is used. It was initially developed in C++ which MATLAB can easily port. Conda package was used to port the algorithm into Python.

In our implementation we are considering two frames in every four frames i.e the third frame from current frame and calculating optical flow between them. This process continues for entire video.

bob.ip.optflow.liu.sor.flow(frame1,frame2) is used to calculate optical flow. It returns value of each pixel into a numpy array of dimension equal to resolution of the frame.

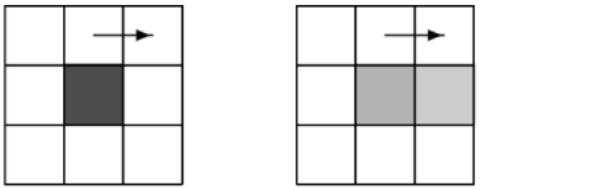

Figure 4.2: Example Optical Flow

4.3 Violent Features

Once the optical flow has been generated, flow vector magnitude is calculated through the following formula:-

$m_{x,y,t} = \sqrt{V_{x,t}^2 + V_{y,t}^2}$

After calculating flow vector , for each pixel in each frame we obtain the binary indicators using the following formula:-

$$b_{x,y,t} = \begin{cases} 1, & if \quad |m_{x,y,t} - m_{x,y,t-1}| \geq \theta \\ 0 & otherwise \end{cases} \tag{4.1}$$

Next we calculate the mean magnitude change by simply averaging these binary indicators for each frame:-

$$\bar{b}_{x,y} = \frac{1}{T} \sum_{t} b_{x,y,t} \tag{4.2}$$

This average binary vector obtained is known as Violent Flow Descriptor (ViF). This ViF values is used for further training and classifications purposes. After ViFs are obtained for a video, a histogram in created of bin size 0.05 and from range 0.0 to 1.0. Which means 21 bins are created. Generated ViF is divided into 16 parts, each part is mapped into a separate histogram, the counts are further normalized by total counts obtained. This process is known as Histogram normalization. Now all these histogram bins values for all 16 parts are appended one after the other leading to generation 336 values for a particular video. These 336 values are used for training the neural net and for predicting using the neural net.

4.4 Training Neural Net

Keras[7] module along with TensorFlow[9] backend is used to build the Neural Net and Train it. The Neural Net built contains an Input Layer, two Dense Layers and an Output Layer as shown in the Fig. 4.3. Each layer is of 336 nodes. Input to the neural net will the Violent Flow Features(ViF) which is a numpy array of dimensions 129*336. Activation function for Input and Middle Layers is ReLU(Rectifier Linear Unit) and for output layer, Sigmoid activation function is used. Training is done for 150 epochs with batch size of 10. Outputs will in the range of 0.0 to 1.0 which will rounded off accordingly.

ReLU activation Function

$$f(x) = max(0, x) \tag{4.3}$$

Sigmoid activation function

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{1 + e^{x}} \tag{4.4}$$

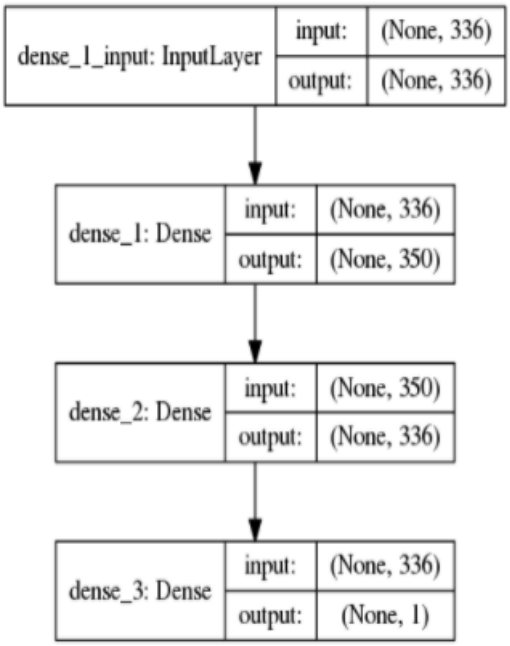

Figure 4.3: Neural Net Layer Information

4.5 Violence Detection

Keras module has been used to train the neural network. The average FPS rate of a video is to be considered 30fps. Average surveillance videos have an FPS rate of 30fps. We consider every 3rd frame for calculating ViFs.

Every 30 frames , i.e every 1 second 336 length array is generated and that array is sent to the previously generated model for prediction. If violence is detected, violence probability is shown on the screen. Multiple lengthy videos have been given as input to the program and it is able to detect the instance where crowd behaviour tends to violent from non_violent.

5. Results and Discussions

The following partial results will be present in the further sections:

- Video Preprocessing
- Optical Flow
- Violent Features
- Neural Net N-Folds accuracy
- Real Time Surveillance on Video
- Real Time Surveillance on Camera Input

5.1 Video Preprocessing

Input frames are resized to 240 X 320. They are further converted to grayscale as shown in Fig. 5.1.

Figure 5.1: Frames before and after preprocessing

5.2 Optical Flow

Optical flow refers to the visible motion of an object in an image, and the apparent 'flow' of pixels in an image. It is the result of 3d motion being projected on a 2-d image plane. Fig. 5.2 shows two consecutive frames of a video in which car is in motion and the output of optical flow. The black regions in output shows the relative motion of car between the frames.

Figure 5.2: Optical Flow Car Example

5.3 Violent Features

Violent Features for a video will contains 336 features. There are 21 bins in a histogram and each frame is divided into 16 blocks. Hence it results into a total of 21 * 16 = 336 features ranging from 0.0 to 1.0 .

Sample Vif

```
3.977272727272727210e-02
1.193181818181818232e-01
0.000000000000000000e+00
1.505681818181818232e-01
0.000000000000000000e+00
2.414772727272727348e-01
0.000000000000000000e+00
1.477272727272727348e-01
0.000000000000000000e+00
1.534090909090909116e-01
8.806818181818182323e-02
0.000000000000000000e+00
3.977272727272727210e-02
0.000000000000000000e+00
1.704545454545454419e-02
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
```

```
0.00000000000000000e+00
0.00000000000000000e+00
1.42045454545454407e-02
7.67045454545545581e-02
0.00000000000000000e+00
1.84659090909090116e-01
0.00000000000000000e+00
2.32954545454454419e-01
0.00000000000000000e+00
1.70454545454454419e-01
0.00000000000000000e+00
1.64772727272727652e-01
9.94318181818181677e-02
0.00000000000000000e+00
3.40909090909090839e-02
0.00000000000000000e+00
2.27272727272727790e-02
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
8.52272727272727096e-03
5.68181818181818629e-02
0.00000000000000000e+00
1.07954545454454558e-01
0.00000000000000000e+00
1.67613636363636535e-01
0.00000000000000000e+00
2.15909090909090116e-01
0.00000000000000000e+00
1.70454545454454419e-01
1.47727272727727348e-01
0.00000000000000000e+00
7.67045454545545581e-02
0.00000000000000000e+00
3.12500000000000000e-02
0.00000000000000000e+00
1.13636363636363395e-02
0.00000000000000000e+00
5.68181818181818629e-03
0.00000000000000000e+00
0.00000000000000000e+00
1.33522727272727652e-01
1.10795454545454442e-01
0.00000000000000000e+00
1.53409090909090116e-01
0.00000000000000000e+00
1.53409090909090116e-01
0.00000000000000000e+00
```

```
1.676136363636363535e-01
0.000000000000000000e+00
1.221590909090909116e-01
9.375000000000000000e-02
0.000000000000000000e+00
4.261363636363636048e-02
0.000000000000000000e+00
1.704545454545454419e-02
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
0.000000000000000000e+00
2.840909090909090988e-03
3.693181818181818371e-02
0.000000000000000000e+00
7.386363636363636742e-02
0.000000000000000000e+00
1.647727272727272652e-01
0.000000000000000000e+00
1.676136363636363535e-01
0.000000000000000000e+00
1.960227272727272652e-01
1.761363636363636465e-01
0.000000000000000000e+00
1.193181818181818232e-01
0.000000000000000000e+00
3.977272727272727210e-02
0.000000000000000000e+00
1.988636363636363605e-02
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
1.704545454545454419e-02
0.000000000000000000e+00
5.113636363636363952e-02
0.000000000000000000e+00
1.534090909090909116e-01
0.000000000000000000e+00
2.215909090909090884e-01
0.000000000000000000e+00
2.301136363636363535e-01
1.505681818181818232e-01
0.000000000000000000e+00
1.079545454545454558e-01
0.000000000000000000e+00
5.681818181818181629e-02
0.000000000000000000e+00
```

```
8.522727272727272096e-03
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
8.522727272727272096e-03
2.272727272727272790e-02
0.000000000000000000e+00
7.954545454545454419e-02
0.000000000000000000e+00
1.250000000000000000e-01
0.000000000000000000e+00
1.903409090909090884e-01
0.000000000000000000e+00
2.357954545454545581e-01
1.732954545454545581e-01
0.000000000000000000e+00
9.943181818181817677e-02
0.000000000000000000e+00
4.829545454545454419e-02
0.000000000000000000e+00
1.420454545454545407e-02
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
5.681818181818181976e-03
3.693181818181818371e-02
0.000000000000000000e+00
5.397727272727272790e-02
0.000000000000000000e+00
1.676136363636363535e-01
0.000000000000000000e+00
2.045454545454545581e-01
0.000000000000000000e+00
2.187500000000000000e-01
1.590909090909090884e-01
0.000000000000000000e+00
9.375000000000000000e-02
0.000000000000000000e+00
5.113636363636363952e-02
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
1.704545454545454419e-02
5.965909090909091161e-02
0.000000000000000000e+00
1.250000000000000000e-01
```

```
0.00000000000000000e+00
1.90340909090909884e-01
0.00000000000000000e+00
1.81818181818181232e-01
0.00000000000000000e+00
1.98863636363636535e-01
1.39204545454545419e-01
0.00000000000000000e+00
6.25000000000000000e-02
0.00000000000000000e+00
1.70454545454545419e-02
0.00000000000000000e+00
5.68181818181818976e-03
0.00000000000000000e+00
2.84090909090909088e-03
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
1.42045454545454507e-02
0.00000000000000000e+00
5.39772727272727790e-02
0.00000000000000000e+00
7.38636363636363742e-02
0.00000000000000000e+00
1.61931818181818768e-01
0.00000000000000000e+00
2.13068181818181232e-01
2.47159090909090916e-01
0.00000000000000000e+00
1.47727272727272348e-01
0.00000000000000000e+00
6.25000000000000000e-02
0.00000000000000000e+00
1.98863636363636605e-02
0.00000000000000000e+00
5.68181818181818976e-03
0.00000000000000000e+00
0.00000000000000000e+00
2.84090909090909088e-03
2.55681818181818976e-02
0.00000000000000000e+00
6.81818181818177677e-02
0.00000000000000000e+00
1.19318181818181232e-01
0.00000000000000000e+00
1.36363636363636535e-01
0.00000000000000000e+00
2.44318181818181232e-01
1.90340909090909884e-01
0.00000000000000000e+00
1.27840909090909884e-01
```

```
0.00000000000000000e+00
3.97727272727272710e-02
0.00000000000000000e+00
3.97727272727272710e-02
0.00000000000000000e+00
5.68181818181818181976e-03
0.00000000000000000e+00
0.00000000000000000e+00
5.68181818181818181976e-03
2.84090909090909090814e-02
0.00000000000000000e+00
6.25000000000000000e-02
0.00000000000000000e+00
1.42045454545454545581e-01
0.00000000000000000e+00
1.84659090909090909116e-01
0.00000000000000000e+00
2.35795454545454545581e-01
1.64772727272727272652e-01
0.00000000000000000e+00
1.19318181818181818232e-01
0.00000000000000000e+00
4.54545454545454545581e-02
0.00000000000000000e+00
1.13636363636363636395e-02
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
8.52272727272727272096e-03
7.10227272727272727904e-02
0.00000000000000000e+00
1.39204545454545454419e-01
0.00000000000000000e+00
1.93181818181818181768e-01
0.00000000000000000e+00
2.27272727272727272652e-01
0.00000000000000000e+00
1.90340909090909090884e-01
1.27840909090909090884e-01
0.00000000000000000e+00
3.97727272727272710e-02
0.00000000000000000e+00
2.84090909090909090988e-03
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
0.00000000000000000e+00
2.84090909090909090988e-03
```

```
5.11363636363636363952e-02
0.00000000000000000000e+00
1.10795454545454545442e-01
0.00000000000000000000e+00
1.50568181818181818232e-01
0.00000000000000000000e+00
2.07386363636363636465e-01
0.00000000000000000000e+00
2.18750000000000000000e-01
1.56250000000000000000e-01
0.00000000000000000000e+00
7.38636363636363636742e-02
0.00000000000000000000e+00
1.98863636363636363605e-02
0.00000000000000000000e+00
8.52272727272727272096e-03
0.00000000000000000000e+00
0.00000000000000000000e+00
0.00000000000000000000e+00
0.00000000000000000000e+00
8.52272727272727272096e-03
1.98863636363636363605e-02
0.00000000000000000000e+00
1.53409090909090909116e-01
0.00000000000000000000e+00
1.93181818181818181768e-01
0.00000000000000000000e+00
2.18750000000000000000e-01
0.00000000000000000000e+00
1.98863636363636363535e-01
1.10795454545454545442e-01
0.00000000000000000000e+00
6.25000000000000000000e-02
0.00000000000000000000e+00
2.27272727272727272790e-02
0.00000000000000000000e+00
1.13636363636363636395e-02
0.00000000000000000000e+00
0.00000000000000000000e+00
0.00000000000000000000e+00
0.00000000000000000000e+00
1.13636363636363636395e-02
7.95454545454545454419e-02
0.00000000000000000000e+00
8.52272727272727272096e-02
0.00000000000000000000e+00
2.10227272727272727348e-01
0.00000000000000000000e+00
2.04545454545454545581e-01
0.00000000000000000000e+00
1.96022727272727272652e-01
```

```
326 9.09090909090909161e-02
327 0.00000000000000000e+00
328 7.10227272727272904e-02
329 0.00000000000000000e+00
330 3.69318181818181371e-02
331 0.00000000000000000e+00
332 1.42045454545454507e-02
333 0.00000000000000000e+00
334 0.00000000000000000e+00
335 0.00000000000000000e+00
336 0.00000000000000000e+00
```

5.4 Neural Net Accuracy Testing

```
(bob_py2) dasarada@dasarada:~/Desktop/ASAGS/keras (copy)$ python keras_neural_ne
tworks_training_70_30_random.py
Using TensorFlow backend.
2018-04-13 13:23:35.658603: I tensorflow/core/platform/cpu_feature_guard.cc:137]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
accuracy is : 0.902777777778
--- 58.6380441189 seconds ---
[[33  2]
 [ 5 32]]
0.902777777778
[76, 8, 111, 75, 50, 74, 25, 56, 55, 83, 102, 129, 9, 17, 85, 101, 35, 117, 107,
 64, 104, 97, 95, 38, 14, 96, 81, 93, 67, 48, 112, 62, 87, 115, 61, 109, 11, 119
, 36, 4, 22, 30, 89, 66, 71, 33, 114, 60, 21, 37, 24, 100, 34, 121, 16, 10, 51,
5, 18, 6, 128, 77, 110, 125, 53, 43, 68, 103, 108, 65, 58, 19, 39, 2, 3, 28, 122
, 127, 84, 59, 13, 90, 105, 54, 98, 29, 94, 124, 26, 1, 46, 42, 76, 8, 111, 75,
50, 74, 25, 56, 55, 83, 102, 129, 9, 17, 85, 101, 35, 117, 107, 64, 104, 97, 95,
 38, 14, 96, 81, 93, 67, 48, 112, 62, 123, 87, 115, 61, 109, 11, 119, 36, 4, 22,
 30, 89, 66, 71, 33, 114, 60, 21, 37, 24, 100, 34, 121, 16, 10, 51, 5, 18, 6, 12
8, 77, 110, 125, 53, 43, 68, 103, 108, 65, 58, 19, 39, 2, 3, 28, 122, 127, 84, 5
9, 13, 90, 105, 54, 98, 29, 94, 124, 26, 1, 46]
[70, 113, 12, 82, 78, 92, 23, 45, 80, 73, 120, 40, 27, 118, 69, 52, 88, 86, 32,
7, 126, 47, 41, 15, 57, 31, 63, 91, 49, 116, 72, 106, 20, 44, 99, 42, 70, 113, 1
2, 82, 78, 92, 23, 45, 80, 73, 120, 79, 40, 27, 118, 69, 52, 88, 86, 32, 7, 126,
 47, 41, 15, 57, 31, 63, 91, 49, 116, 72, 106, 20, 44, 99]
```

Figure 5.3: Accuracy of Neural Net by 70:30 method

	p	n
P	33	2
N	5	32

Table 5.1: Confusion Matrix for 70:30 dataset

So as to test the accuracy of the generated neural net. The dataset which contains 246 videos is divided in the ratio of 70:30. 70% of the data is used to train the neural net, 30% of the data is used to test the accuracy of generated model. Output in Fig. 5.3 shows

that the accuracy obtained is 90.27% . As we can see in the confusion matrix Table 5.1, the number of False Negatives are just 2, that means there are only 2 cases in the test set which are actually violent but our system was not able to detect it. Whereas there were 5 cases in which videos were not violent but our system detected some violence.
Following are some results obtained:

- Accuracy = TP/total = 0.9027
- True Positive Rate = TP/P = 0.9428
- Precision = TP/(predicted yes) = 0.8684
- Specificity = TN/(actual no) = 0.8648

5.5 Neural Net N-folds Cross Verification

Seven fold cross validation is the validation manner which is adopted in experiments. All the videos are divided into seven heaps with the same ratio between violent and non-violent ones. At each time one distinct heap is selected for testing and the other six heaps for training. This procedure is then repeated for seven times. The accuracy for seven-folds and overall accuracy is shown in Fig. 5.4.

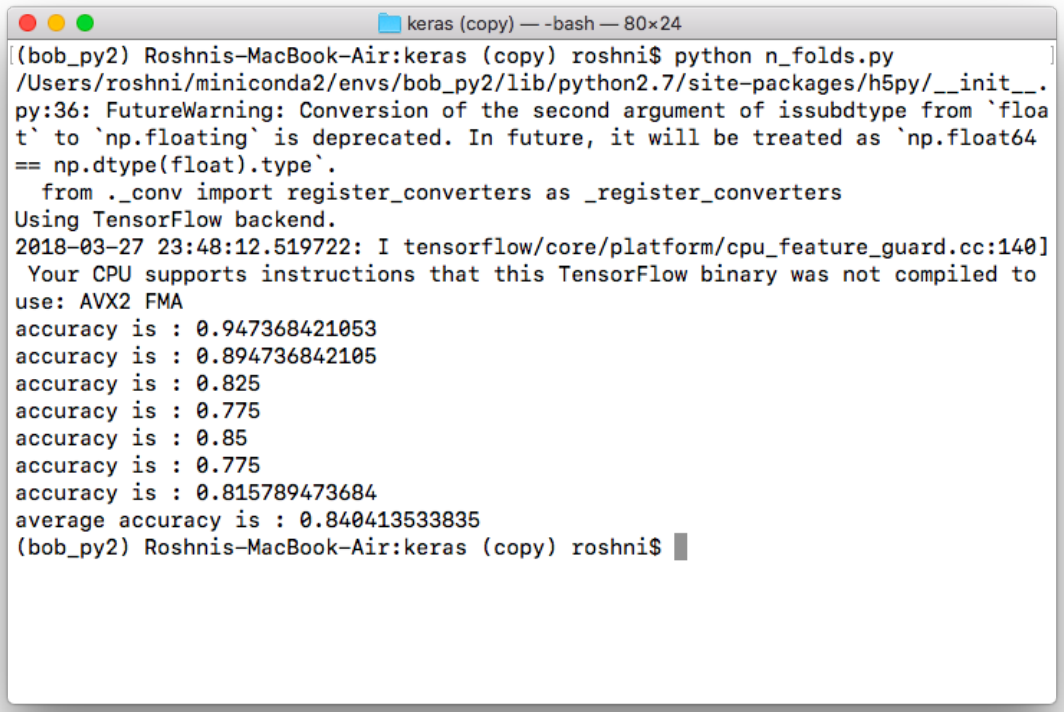

Figure 5.4: Neural Net Folds Accuracy

5.6 Real Time Surveillance on Video Input

The usual FPS rate of a standard surveillance is 25. Which means our algorithm has to process each frame in less than 1/25 th of a second. Real Time Surveillance of a Video outputs on terminal and it provides the exact second where the frames go from violent to non-violent as shown in Fig. 5.5. So with the help of this we are able to decide the exact second where the violence occurs.

```
(bob_py2) dasarada@dasarada:~/Desktop/ASAGS/continous_system$ python test_packag
e_surveillance_2.py
Using TensorFlow backend.
2018-04-13 17:11:25.198689: I tensorflow/core/platform/cpu_feature_guard.cc:137]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
loaded model from disk
FPS is : 30.0
violent  ---   5 seconds , processing time : 4.36195492744
violent  ---   9 seconds , processing time : 7.27132701874
violent  ---   11 seconds , processing time : 8.73718595505
violent  ---   29 seconds , processing time : 21.8036780357
violent  ---   33 seconds , processing time : 24.698925972
violent  ---   39 seconds , processing time : 28.9961290359
violent  ---   41 seconds , processing time : 30.4373898506
violent  ---   45 seconds , processing time : 33.2991299629
violent  ---   76 seconds , processing time : 55.042840004
violent  ---   80 seconds , processing time : 57.9317998886
violent  ---   104 seconds , processing time : 75.2060849667
violent  ---   110 seconds , processing time : 79.4895160198
violent  ---   124 seconds , processing time : 89.627259016
violent  ---   130 seconds , processing time : 93.901059866
violent  ---   132 seconds , processing time : 95.3465769291
violent  ---   134 seconds , processing time : 96.7611138821
violent  ---   136 seconds , processing time : 98.1910500526
violent  ---   138 seconds , processing time : 99.6176309586
violent  ---   140 seconds , processing time : 101.034286976
violent  ---   162 seconds , processing time : 116.934856892
violent  ---   172 seconds , processing time : 124.091480017
violent  ---   174 seconds , processing time : 125.52096796
violent  ---   182 seconds , processing time : 131.316578865
violent  ---   184 seconds , processing time : 132.749922991
violent  ---   186 seconds , processing time : 134.187556028
violent  ---   188 seconds , processing time : 135.628131866
Done
```

Figure 5.5: Surveillance on Video

Consider the above figure, it shows where the violence occurs along with the processing time required to detect the violence at that moment. As we can see the processing time required for the complete video is less than the play time of the video. Hence it shows that surveillance is taking place in real time.

Consider the following example:

Figure 5.6: No Violence

Figure 5.7: Violence about to start

Figure 5.8: Slight Violence

Figure 5.9: Furious Violence

Above four frames are taken from a surveillance video. In Fig. 5.6 violence has not yet started, in Fig. 5.7 violence has not started but it is about to start. In Fig. 5.8 violence has started and in Fig. 5.9 there is furious violence.

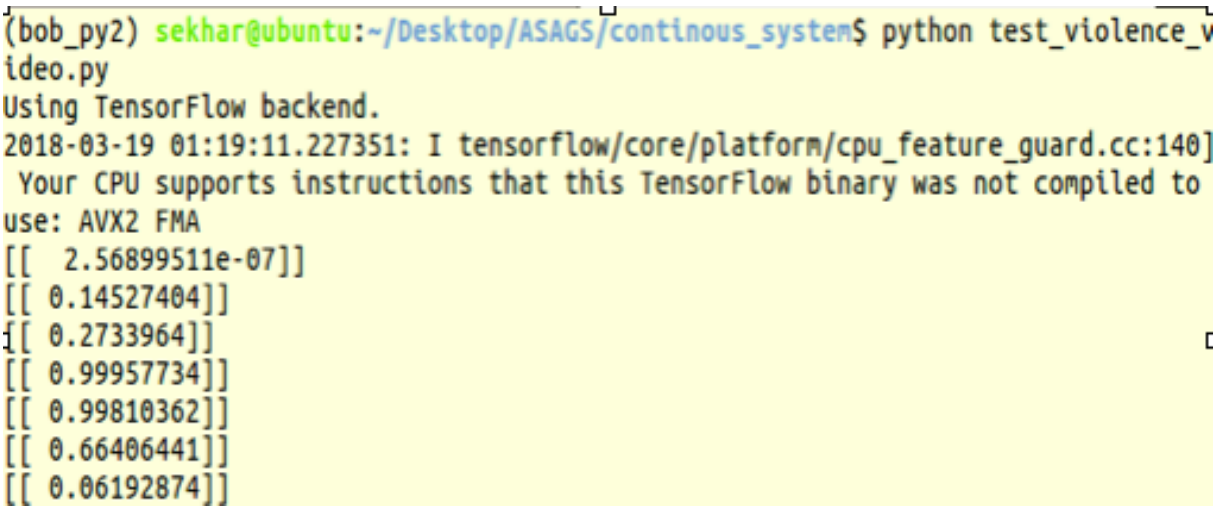

Figure 5.10: Surveillance output of video in Terminal

As we can see in Fig. 5.10, the surveillance output is initially very less since there is no violence, as the chance of violence is increased due to increase in motion the corresponding value appering on the terminal also increases. When there is furious violence the value goes upto 0.999. Further the violence decrease and we can see the output values also decrease.

5.7 Real Time Surveillance on Camera Input

OpenCV[8] helps us to take video input directly from the camera connected. It can be externally connected through a usb port or it can be the internal webcam. Surveillance of webcam video is shown in Fig. 5.11.

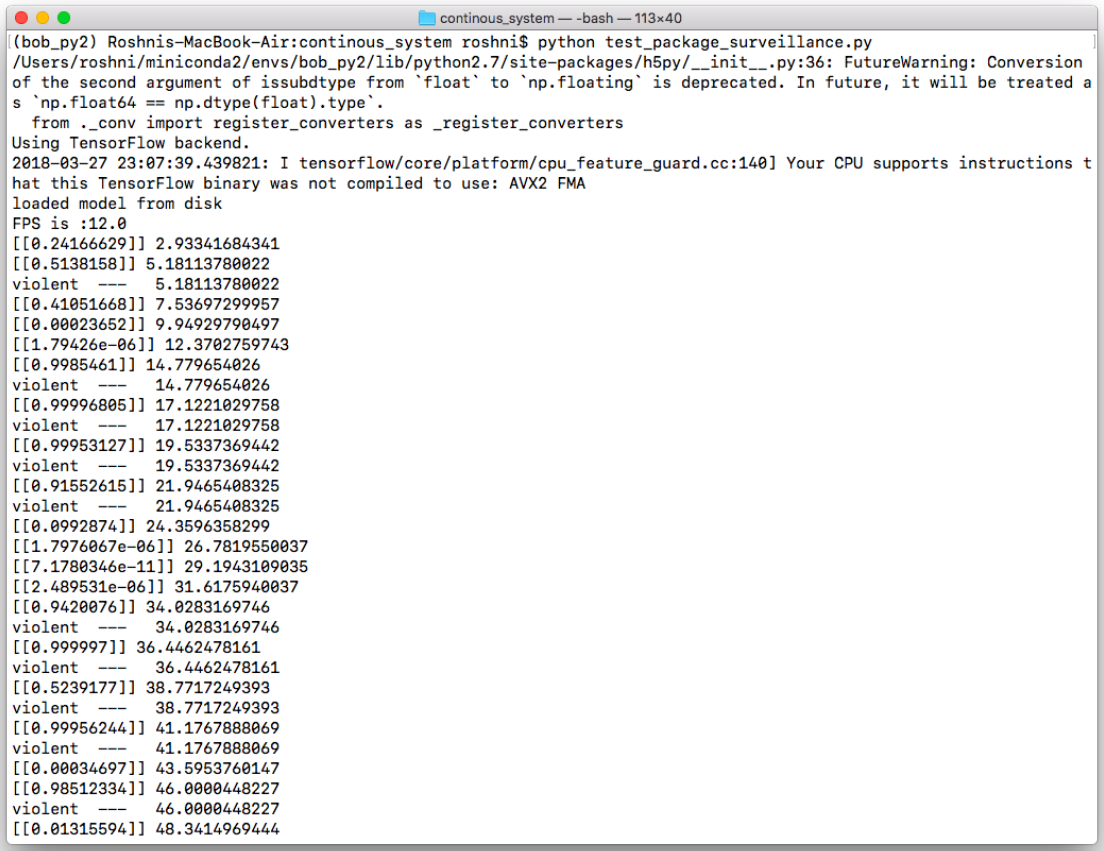

Figure 5.11: Surveillance through Camera

6. Conclusion and Future Work

6.1 Conclusion

A system which can identify disturbance in the crowd is performing successfully in real-time with high accuracy. Disturbance is the unruly behaviour of crowd caused due to panic or violence. The system is clearly able to differentiate between non-violent crowded scenes such as 'fans cheering in stadium' and violent crowded scenes such as 'prison gang fights'. The surveillance system generated can be used in real-time with CCTV cameras input and with video inputs also. The surveillance system has enough reaction time accuracy and classification accuracy to work independently. The generated system can perform surveillance on upto four input video streams at a time.

6.2 Future Work

- Scalability can be further improved using threads and multiprocessing.
- Code can be extended to HD videos also.
- Using AdaBoost feature selection algorithm top features among the 336 features have been extracted.
- We can use above selected features and increase the weights of the nodes that accept inputs from those particular features.
- By above step accuracy will be greatly increased.
- System can be connected with multiple cameras and the application can be tested.
- We can include the angle of motion change during calculation of ViF to improve accuracy.

References

[1] Gao, Yuan and Liu, Hong and Sun, Xiaohu and Wang, Can and Liu, Yi. *Violence Detection Using Oriented Violent Flows*,Key Laboratory of Machine Perception, Shenzhen Graduate School, Peking University, Beijing 100871, China. Image Vision Comput, April 2016

[2] T. Hassner, Y. Itcher, O. Kliper Gross. *Violent Flows: Real-Time Detection of Violent Crowd Behavior*,3rd IEEE International Workshop on Socially Intelligent Surveillance and Monitoring (SISM) at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR),June 2012

[3] Ce. Liu. *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis*,Massachusetts Institute of Technology,Ph.D. Thesis,2009

[4] Anjos, André AND El Shafey, Laurent AND Wallace, Roy AND Günther, Manuel AND McCool, Christopher AND Marcel, Sébastien. *Bob: a free signal processing and machine learning toolbox for researchers*,20th ACM Conference on Multimedia Systems (ACMMM), Nara Japan, 2012s

[5] Eugene Yujun Fu, Hong Va Leong, Grace Ngai, Stephen Chan *Automatic Fight Detection in Surveillance Videos* 2016

[6] Andrea Pennisi, Domenico D. Bloisi, Luca Iocchi *Online real-time crowd behaviour detection in video sequences* 2015

[7] Chollet, François and others. *Keras*, 2015. Source available at`https://keras.io`

[8] Itseez. *Open Source Computer Vision Library*, 2015. Source available at `https://github.com/itseez/opencv`

[9] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan

Man, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Source available at `https://www.tensorflow.org`

A. Appendix

A.1 Video Preprocessing

process.py

```
import numpy
import cv2
import sys
import time

class PreProcess:
    def __init__(self):
        #constants----------------
        self.FRAME_RATE = 25 #25 frames per second
        self.MOVEMENT_INTERVAL = 3 #difference between considered frames
        self.N = 4 #number of vertical blocks per frame
        self.M = 4 #number of horizontal blocks per frame
        self.FRAME_GAP = 2 * self.MOVEMENT_INTERVAL
        #-------------------------
        self.cap = ''
        self.total_frames = 0
        self.fps = 0
        self.time = 0
        #-------------------------
        self.dim = 100
        #-------------------------
        self.frame_number = 0

    def read_video(self,video_name):
        self.cap = cv2.VideoCapture(video_name)
        self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        self.fps = self.cap.get(cv2.CAP_PROP_FPS)
        self.time = self.total_frames / self.fps
        # self.last_frame = self.read_frame()

    def getFrameFromIndex(self,frame_no):
        #Number 2 defines flag CV_CAP_PROP_POS_FRAMES which is a 0-based
            ↪ index of the frame to be decoded/captured next.
        #The second argument defines the frame number in range 0.0-1.0
        self.cap.set(1,frame_no)
        ret , img = self.cap.read()
        if img is None:
            sys.exit('Done')
        img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        return img

    def resize_frame(self,frame):
        rescale = float(self.dim)/(frame.shape[1])
```

```
43          if rescale<0.8:
44              dim = (self.dim, int(frame.shape[0] * rescale))
45              frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
46          return frame
47
48      def setVideoDimension(self,dim):
49          self.dim = dim
50
51      def useCamera(self):
52          self.cap = cv2.VideoCapture(0)
53          self.last_frame = self.read_frame()
54
55      def showInputFromCamera(self):
56          while True:
57              ret , frame = self.cap.read()
58              cv2.imshow('camera_frame',frame)
59              cv2.imshow('resized_camera_frame',self.resize_frame(cv2.cvtColor(
                    ↪ frame,cv2.COLOR_BGR2GRAY)))
60
61              if cv2.waitKey(1) & 0xFF == ord('q'):
62                  break
63
64      def read_frame(self):
65          ret , frame = self.cap.read()
66          return frame
67
68      def getFPS(self):
69          return self.cap.get(cv2.CAP_PROP_FPS)
70
71      def getFramesFromVideoSource(self):
72          PREV_F = self.getFrameFromIndex(self.frame_number)
73          CURRENT_F = self.getFrameFromIndex(self.frame_number + self.
                ↪ MOVEMENT_INTERVAL)
74          NEXT_F = self.getFrameFromIndex(self.frame_number + (2 * self.
                ↪ MOVEMENT_INTERVAL))
75
76          frames = (PREV_F,CURRENT_F,NEXT_F,self.frame_number)
77          self.frame_number += 6
78
79          return frames
80
81      def getFramesFromCameraSource(self):
82          frame1 = self.last_frame
83          for i in range(0,self.MOVEMENT_INTERVAL-1):
84              self.read_frame()
85          frame2 = self.read_frame()
86          for i in range(0,self.MOVEMENT_INTERVAL-1):
87              self.read_frame()
88          frame3 = self.read_frame()
89          self.last_frame = frame3
90
```

```
91          # cv2.imshow('camera_frame',frame2)
92
93          frame1 = self.resize_frame(frame1)
94          frame2 = self.resize_frame(frame2)
95          frame3 = self.resize_frame(frame3)
96
97          frame1 = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
98          frame2 = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
99          frame3 = cv2.cvtColor(frame3,cv2.COLOR_BGR2GRAY)
100
101         # cv2.imshow('resized_camera_frame',frame2)
102
103         frames = (frame1,frame2,frame3,time.time())
104
105         return frames
```

A.2 Optical Flow

flow.py

```
1  import bob.ip.optflow.liu.sor
2  import numpy as np
3  class OptFlow:
4      def __init__(self):
5          self.alpha = 0.0026
6          self.ratio = 0.6
7          self.minWidth = 20
8          self.nOuterFPIterations = 7
9          self.nInnerFPIterations = 1
10         self.nSORIterations = 30
11         self.flows = ()
12
13     def sorFlow(self,frame1,frame2):
14         self.flows = bob.ip.optflow.liu.sor.flow(frame1,frame2,self.alpha,
               ↪ self.ratio,self.minWidth,self.nOuterFPIterations,self.
               ↪ nInnerFPIterations,self.nSORIterations)
15         #self.flows = bob.ip.optflow.liu.sor.flow(frame1,frame2)
16         return self.flows
17
18
19     def getFlowMagnitude(self,vx,vy):
20         flow_magnitude = np.sqrt(np.square(vx) + np.square(vy))
21         return flow_magnitude
```

A.3 Generate ViFs for Violent Videos

violent_features_VIOLENT.py

```
1  from ViolentFlow import VioFlow
```

```
import time
file_vio = open('violent_list.txt')
path = '/Users/roshni/Desktop/VideoData/Violence/'
start_time = time.time()
for each_file in file_vio.readlines():
    each_file = each_file[:-1]
    feature = VioFlow(path + each_file)
    out_file = each_file[:-3] + 'txt'
    print each_file + '------------------------------------------------------'
    try:
        feature.writeFeatureToFile('violent_features_VIOLENT/' + out_file)
        print each_file + ' done'
    except:
        print 'error in ' + each_file
print("--- %s seconds ---" % (time.time() - start_time))
```

A.4 Generate ViFs for Non Violent Videos

violent_features_NON_VIOLENT.py

```
from ViolentFlow import VioFlow
import time
file_vio = open('non_violent_list.txt')
start_time = time.time()
path = '/Users/roshni/Desktop/VideoData/Non-Violence/'
for each_file in file_vio.readlines():
    each_file = each_file[:-1]
    feature = VioFlow(path + each_file)
    out_file = each_file[:-3] + 'txt'
    print each_file + '------------------------------------------------------'
    try:
        feature.writeFeatureToFile('violent_features_NON_VIOLENT/' + out_file
            ↪ )
        print each_file + ' done'
    except:
        print 'error in ' + each_file
print("--- %s seconds ---" % (time.time() - start_time))
```

A.5 Training SVM and Testing Accuracy

train_predict_svm_70_30_random.py

```
from sklearn import svm
import numpy as np
import time
import random
acc_all = 0.0
for i in range(1,21):
    start_time = time.time()
```

```
    X_train = []
    Y_train = []
    X_test = []
    Y_test = []
    count = 0
    data = range(1,130)
    random.shuffle(data)
    #reading non violent video features
    for i in data:
        try:
            file_name = 'violent_features_NON_VIOLENT/nonvio_'+str(i)+'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:
                continue
            if count < 92:
                X_train.append(vif)
                Y_train.append(0)
            else:
                X_test.append(vif)
                Y_test.append(0)
            file_obj.close()
            count+=1
        except:
            continue
            print 'error in reading nonvio_%d.txt'%i
    #reading violent video features
    count = 0
    for i in data:
        try:
            file_name = 'violent_features_VIOLENT/vio_'+str(i)+'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:
                continue
            if count < 92:
                X_train.append(vif)
                Y_train.append(1)
            else:
                X_test.append(vif)
                Y_test.append(1)
            file_obj.close()
            count+=1
        except:
            continue
            print 'error in reading vio_%d.txt'%i

    #print len(X_train)
    #print len(X_test)
    #training
    clf = svm.SVC(kernel = 'linear')
```

```
    clf.fit(X_train,Y_train)
    print clf
    print("--- %s seconds ---" % (time.time() - start_time))


    #predicting
    pred = []

    for i in X_test:
        pred.append(clf.predict(i.reshape(1,-1)))

    count = 0

    for i in range(0,len(Y_test)):
        if pred[i][0] == Y_test[i]:
            count = count + 1

    accuracy = float(count)/len(Y_test)
    print 'accuracy is : ' + str(accuracy)
    acc_all = acc_all + accuracy
print 'overall : ' + str(acc_all/20.0)
```

A.6 SVM N-folds Cross Verification

n_folds_cross_verification.py

```
from sklearn import svm
import numpy as np
import random

total_accuracy = 0.0
iters = 0

data_violent = range(1,130)
data_nonviolent = range(1,130)
random.shuffle(data_violent)
random.shuffle(data_nonviolent)

for i in range(10,131,10):
    X_train = []
    Y_train = []
    X_test = []
    Y_test = []
    iters += 1
    test_set = range(i-10,i)
    for j in test_set:
        try:
            file_name = 'violent_features_NON_VIOLENT/nonvio_'+str(
                ↪ data_nonviolent[j])+'.txt'
            file_obj = open(file_name,'r')
```

```
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:# avoiding hd videos
                continue
            X_test.append(vif)
            Y_test.append(0)
            file_obj.close()
        except:
            continue
            print 'error in reading nonvio_%d.txt'%data_nonviolent[i]
        try:
            file_name = 'violent_features_VIOLENT/vio_'+str(data_violent[j])
                ↪ +'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:# avoiding hd videos
                continue
            X_test.append(vif)
            Y_test.append(1)
            file_obj.close()
        except:
            continue
            print 'error in reading nonvio_%d.txt'%data_violent[i]
    for j in range(1,130):
        try:
            if j in [data_nonviolent[l] for l in test_set]:
                continue
            file_name = 'violent_features_NON_VIOLENT/nonvio_'+str(j)+'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:# avoiding hd videos
                continue
            X_train.append(vif)
            Y_train.append(0)
            file_obj.close()
        except:
            continue
            print 'error in reading nonvio_%d.txt'%j
    for j in range(1,130):
        try:
            if j in [data_violent[l] for l in test_set]:
                continue
            file_name = 'violent_features_VIOLENT/vio_'+str(j)+'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 336:# avoiding hd videos
                continue
            X_train.append(vif)
            Y_train.append(1)
            file_obj.close()
        except:
            continue
```

```
            print 'error in reading vio_%d.txt'%j

    clf = svm.SVC(kernel = 'linear')
    if len(X_train) == 0:
        iters -= 1
        continue
    clf.fit(X_train,Y_train)
    print clf

    pred = []

    for i in X_test:
        pred.append(clf.predict(i.reshape(1,-1)))

    count = 0

    for i in range(0,len(Y_test)):
        if pred[i][0] == Y_test[i]:
            count = count + 1

    total_accuracy += float(count)/len(Y_test)
    print 'accuracy is : '+str(float(count)/len(Y_test))

print 'average accuracy is : ' + str(total_accuracy/iters)
```

A.7 Training Neural Net and Testing Accuracy

keras_neural_networks_training_70_30_random_20avg.py

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
import numpy as np
import random
import time

ovr_acc = 0.0
start_time = time.time()
for j in range(1,21):
    X_train = np.empty((0,336))
    Y_train = np.array([])
    X_test = np.empty((0,336))
    Y_test = np.array([])
    count = 0
    data = range(1,130)
    random.shuffle(data)

    for i in data:
        try:
```

```
21            file_name = 'violent_features_NON_VIOLENT/nonvio_' + str(i) + '.
                  ↪ txt'
22            file_obj = open(file_name, 'r')
23            vif = np.loadtxt(file_obj)
24            if vif.shape[0] == 630: # avoiding hd videos
25                continue
26            vif = np.reshape(vif, (-1, vif.shape[0]))
27            if count < 92:
28                X_train = np.vstack((X_train,vif))
29                Y_train = np.append(Y_train,0)
30            else:
31                X_test = np.vstack((X_test,vif))
32                Y_test = np.append(Y_test,0)
33            file_obj.close()
34            count += 1
35        except:
36            continue
37            print 'error in reading nonvio_%d.txt' % i
38
39 # reading violent video features
40    count = 0
41    for i in data:
42        try:
43            file_name = 'violent_features_VIOLENT/vio_' + str(i) + '.txt'
44            file_obj = open(file_name, 'r')
45            vif = np.loadtxt(file_obj)
46            if vif.shape[0] == 630: # avoiding hd videos
47                continue
48            vif = np.reshape(vif, (-1, vif.shape[0]))
49            if count < 92:
50                X_train = np.vstack((X_train, vif))
51                Y_train = np.append(Y_train, 1)
52            else:
53                X_test = np.vstack((X_test, vif))
54                Y_test = np.append(Y_test, 1)
55            file_obj.close()
56            count += 1
57        except:
58            continue
59            print 'error in reading vio_%d.txt' % i
60
61
62
63
64    seed = 7
65    np.random.seed(seed)
66    model = Sequential()
67    model.add(Dense(350, activation="relu", kernel_initializer="uniform",
          ↪ input_dim=336))
68
69    for l in range(1,7):
```

```
70          model.add(Dense(336, activation='relu', kernel_initializer="uniform")
                ↪ )
71
72      model.add(Dense(1, activation="sigmoid", kernel_initializer="uniform"))
73
74      model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
            ↪ accuracy'])
75
76      model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)
77
78      predictions = model.predict(X_test)
79
80      pred = [round(x[0]) for x in predictions]
81
82
83      acc_count = 0
84      for k in range(0,len(pred)):
85          if pred[k] == Y_test[k]:
86              acc_count += 1
87
88      cm = confusion_matrix(Y_test, pred)
89      print cm
90
91      accuracy = float(acc_count)/len(pred)
92      print 'accuracy is : ' + str(accuracy)
93      ovr_acc += accuracy
94
95  print 'overall : ' + str(ovr_acc/20.0)
96  print("--- %s seconds ---" % (time.time() - start_time))
```

A.8 Training Neural Net and Storing on disk

keras_train_100_once_test_30_multiple_store_model.py

```
1  from keras.models import Sequential
2  from keras.layers import Dense
3  from sklearn.metrics import confusion_matrix
4  import numpy as np
5  import random
6  import time
7
8  data = range(1,130)
9  random.shuffle(data)
10 X_train = np.empty((0,336))
11 Y_train = np.array([])
12 for i in data:
13     try:
14         file_name = 'violent_features_NON_VIOLENT/nonvio_' + str(i) + '.txt'
15         file_obj = open(file_name, 'r')
16         vif = np.loadtxt(file_obj)
```

```
        if vif.shape[0] == 630: # avoiding hd videos
            continue
        vif = np.reshape(vif, (-1, vif.shape[0]))

        X_train = np.vstack((X_train,vif))
        Y_train = np.append(Y_train,0)

        file_obj.close()
    except:
        continue
        print 'error in reading nonvio_%d.txt' % i

# reading violent video features
for i in data:
    try:
        file_name = 'violent_features_VIOLENT/vio_' + str(i) + '.txt'
        file_obj = open(file_name, 'r')
        vif = np.loadtxt(file_obj)
        if vif.shape[0] == 630: # avoiding hd videos
            continue
        vif = np.reshape(vif, (-1, vif.shape[0]))

        X_train = np.vstack((X_train, vif))
        Y_train = np.append(Y_train, 1)

        file_obj.close()
    except:
        continue
        print 'error in reading vio_%d.txt' % i

seed = 7
np.random.seed(seed)
model = Sequential()
model.add(Dense(350, activation="relu", kernel_initializer="uniform",
    ↪ input_dim=336))

for l in range(1,2):
    model.add(Dense(336, activation='relu', kernel_initializer="uniform"))
model.add(Dense(1, activation="sigmoid", kernel_initializer="uniform"))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
    ↪ accuracy'])

model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)

print 'model trained'

ovr_acc = 0.0
start_time = time.time()
for j in range(1,21):
    random.shuffle(data)
```

```
    X_test = np.empty((0,336))
    Y_test = np.array([])
    count = 0
    for i in data:
        try:
            file_name = 'violent_features_NON_VIOLENT/nonvio_' + str(i) + '.
                ↪ txt'
            file_obj = open(file_name, 'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 630: # avoiding hd videos
                continue
            vif = np.reshape(vif, (-1, vif.shape[0]))
            if count%2==0 and len(Y_test)<=39:
                X_test = np.vstack((X_test,vif))
                Y_test = np.append(Y_test,0)

            file_obj.close()
            count += 1
        except:
            continue
            print 'error in reading nonvio_%d.txt' % i

# reading violent video features
    count = 0
    for i in data:
        try:
            file_name = 'violent_features_VIOLENT/vio_' + str(i) + '.txt'
            file_obj = open(file_name, 'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 630: # avoiding hd videos
                continue
            vif = np.reshape(vif, (-1, vif.shape[0]))
            if count%2==0 and len(Y_test)<=78:
                X_test = np.vstack((X_test, vif))
                Y_test = np.append(Y_test, 1)

            file_obj.close()
            count += 1
        except:
            continue
            print 'error in reading vio_%d.txt' % i




    predictions = model.predict(X_test)

    pred = [round(x[0]) for x in predictions]


    acc_count = 0
```

```
    for k in range(0,len(pred)):
        if pred[k] == Y_test[k]:
            acc_count += 1

    cm = confusion_matrix(Y_test, pred)
    print cm

    accuracy = float(acc_count)/len(pred)
    print 'accuracy is : ' + str(accuracy)
    ovr_acc += accuracy

print 'overall : ' + str(ovr_acc/20.0)
print("--- %s seconds ---" % (time.time() - start_time))

model_json = model.to_json()

with open("model_100.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("model_100.h5")
print("Saved model to disk")
```

A.9 Load Neural Net from Disk and Visualize

keras_visualize.py

```
from keras.models import model_from_json
from keras.utils.vis_utils import plot_model, model_to_dot
from IPython.display import SVG,display_svg

# load model
json_file = open('model_100.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load model weights
loaded_model.load_weights("model_100.h5")

plot_model(loaded_model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)

display_svg(SVG(model_to_dot(loaded_model).create(prog='dot', format='svg')))
```

A.10 Load Neural Net and perform N-Folds Cross Verification

n_folds_neural_net.py

```
from keras.models import Sequential
```

```
from keras.layers import Dense
import numpy as np
import random
import time

ovr_acc = 0.0
iters = 0
start_time = time.time()

data_violent = range(1,130)
data_nonviolent = range(1,130)
random.shuffle(data_violent)
random.shuffle(data_nonviolent)

for i in range(20,131,20):
    X_train = np.empty((0,336))
    Y_train = np.array([])
    X_test = np.empty((0,336))
    Y_test = np.array([])
    iters += 1
    test_set = range(i-20,i)
    for j in test_set:
        try:
            file_name = 'violent_features_NON_VIOLENT/nonvio_'+str(
                ↪ data_nonviolent[j])+'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 630:# avoiding hd videos
                continue
            vif = np.reshape(vif,(-1,vif.shape[0]))
            X_test = np.vstack((X_test,vif))
            Y_test = np.append(Y_test,0)
            file_obj.close()
        except:
            continue
            print 'error in reading nonvio_%d.txt'%data_nonviolent[i]
        try:
            file_name = 'violent_features_VIOLENT/vio_'+str(data_violent[j])
                ↪ +'.txt'
            file_obj = open(file_name,'r')
            vif = np.loadtxt(file_obj)
            if vif.shape[0] == 630:# avoiding hd videos
                continue
            vif = np.reshape(vif,(-1,vif.shape[0]))
            X_test = np.vstack((X_test,vif))
            Y_test = np.append(Y_test,1)
            file_obj.close()
        except:
            continue
            print 'error in reading nonvio_%d.txt'%data_violent[i]
    for j in range(1,130):
```

```
51          try:
52              if j in test_set:
53                  continue
54              file_name = 'violent_features_NON_VIOLENT/nonvio_'+str(
                    ↪ data_nonviolent[j])+'.txt'
55              file_obj = open(file_name,'r')
56              vif = np.loadtxt(file_obj)
57              if vif.shape[0] == 630:# avoiding hd videos
58                  continue
59              vif = np.reshape(vif,(-1,vif.shape[0]))
60              X_train = np.vstack((X_train,vif))
61              Y_train = np.append(Y_train,0)
62              file_obj.close()
63          except:
64              continue
65              print 'error in reading nonvio_%d.txt'%j
66      for j in range(1,130):
67          try:
68              if j in test_set:
69                  continue
70              file_name = 'violent_features_VIOLENT/vio_'+str(data_violent[j])
                    ↪ +'.txt'
71              file_obj = open(file_name,'r')
72              vif = np.loadtxt(file_obj)
73              if vif.shape[0] == 630:# avoiding hd videos
74                  continue
75              vif = np.reshape(vif,(-1,vif.shape[0]))
76              X_train = np.vstack((X_train,vif))
77              Y_train = np.append(Y_train,1)
78              file_obj.close()
79          except:
80              continue
81              print 'error in reading vio_%d.txt'%j
82
83      if len(X_train) == 0:
84          iters -= 1
85          continue
86
87      seed = 7
88      np.random.seed(seed)
89      model = Sequential()
90      model.add(Dense(350, activation="relu", kernel_initializer="uniform",
            ↪ input_dim=336))
91
92      for l in range(1,5):
93          model.add(Dense(336, activation='relu', kernel_initializer="uniform")
                ↪ )
94
95      model.add(Dense(1, activation="sigmoid", kernel_initializer="uniform"))
96
```

```
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
        ↪ accuracy'])

    model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)

    predictions = model.predict(X_test)

    pred = [round(x[0]) for x in predictions]


    acc_count = 0
    for k in range(0,len(pred)):
        if pred[k] == Y_test[k]:
            acc_count += 1

    accuracy = float(acc_count)/len(pred)
    print 'accuracy is : ' + str(accuracy)
    ovr_acc += accuracy
print 'average accuracy is : ' + str(ovr_acc/iters)
```

A.11 Surveillance System Working Class

surveillance.py

```
import cv2
import numpy as np
from VideoProcess import PreProcess
from OpticalFlow import OptFlow
import math
from keras.models import model_from_json
import time

class ContinousSurv:
    def __init__(self):
        json_file = open('model_100.json', 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        self.model = model_from_json(loaded_model_json)
        self.model.load_weights("model_100.h5")
        print 'loaded model from disk'

        self.vid = PreProcess()
        self.vid.setVideoDimension(100)
        self.flow = OptFlow()
        self.height = 0
        self.width = 0
        self.B_height = 0
        self.B_width = 0
        self.index = 0
        self.temp_flows = []
```

```
27          self.bins = np.arange(0.0,1.05,0.05,dtype=np.float64)
28
29      def setVideoName(self,video_name):
30          self.vid.read_video(video_name)
31
32      def histc(self,X, bins):
33          map_to_bins = np.digitize(X,bins)
34          r = np.zeros(bins.shape,dtype=np.float64)
35          for i in map_to_bins:
36              r[i-1] += 1
37          return r
38
39      def getBlockHist(self,flow_video):
40          flow_vec = np.reshape(flow_video,(flow_video.shape[0]*flow_video.
                ↪ shape[1],1))
41          count_of_bins = self.histc(flow_vec,self.bins)
42          return count_of_bins/np.sum(count_of_bins)
43
44      def getFrameHist(self,flow_video_size):
45          flow_video = np.zeros(flow_video_size,dtype=np.float64)
46          for each_flow in self.temp_flows:
47              flow_video = flow_video + each_flow
48          flow_video = flow_video / self.index
49          self.index = 0
50          self.temp_flows = []
51          self.height = flow_video.shape[0]
52          self.width = flow_video.shape[1]
53          self.B_height = int(math.floor((self.height - 11)/4))
54          self.B_width = int(math.floor((self.width - 11)/4))
55          frame_hist = []
56          for y in range(6,self.height-self.B_height-4,self.B_height):
57              for x in range(6,self.width-self.B_width-4,self.B_width):
58                  block_hist = self.getBlockHist(flow_video[y:y+self.B_height,x:
                        ↪ x+self.B_width])
59                  frame_hist = np.append(frame_hist,block_hist,axis = 0)
60          return frame_hist
61
62      def doSurveillanceFromVideo(self):
63          FPS = round(self.vid.getFPS())
64          print 'FPS is : '+str(FPS)
65          while True:
66              frames = self.vid.getFramesFromVideoSource()
67              PREV_F = frames[0]
68              CURRENT_F = frames[1]
69              NEXT_F = frames[2]
70
71              frame_number = frames[3]
72
73              PREV_F = self.vid.resize_frame(PREV_F)
74              CURRENT_F = self.vid.resize_frame(CURRENT_F)
75              NEXT_F = self.vid.resize_frame(NEXT_F)
```

```
76
77              (vx1,vy1,w1) = self.flow.sorFlow(PREV_F,CURRENT_F)
78              (vx2,vy2,w2) = self.flow.sorFlow(CURRENT_F,NEXT_F)
79
80              m1 = self.flow.getFlowMagnitude(vx1,vy1)
81              self.index = self.index + 1
82              m2 = self.flow.getFlowMagnitude(vx2,vy2)
83
84              change_mag = abs(m2-m1)
85              binary_mag = np.ones(change_mag.shape,dtype=np.float64)
86              threshold = np.mean(change_mag , dtype=np.float64)
87              self.temp_flows.append(np.where(change_mag < threshold,0,
                    ↪ binary_mag))
88
89              if self.index>=int(FPS/3):
90                  vif = self.getFrameHist(CURRENT_F.shape)
91                  X_frame = np.empty((0,336))
92                  vif = np.reshape(vif, (-1, vif.shape[0]))
93                  X_frame = np.vstack((X_frame, vif))
94                  pred = self.model.predict(X_frame)
95                  pred = round(pred[0][0])
96                  if pred == 1:
97                      time_violence = float(frame_number) / self.vid.fps
98                      print 'violent --- '+str(int(time_violence))+' seconds'
99
100     def doSurveillanceFromCamera(self):
101         start_time = time.time()
102         self.vid.useCamera()
103         FPS = round(self.vid.getFPS())
104         print 'FPS is :'+str(FPS)
105         while True:
106             frames = self.vid.getFramesFromCameraSource()
107             PREV_F = frames[0]
108             CURRENT_F = frames[1]
109             NEXT_F = frames[2]
110
111             time_now = frames[3]
112
113             PREV_F = self.vid.resize_frame(PREV_F)
114             CURRENT_F = self.vid.resize_frame(CURRENT_F)
115             NEXT_F = self.vid.resize_frame(NEXT_F)
116
117             (vx1,vy1,w1) = self.flow.sorFlow(PREV_F,CURRENT_F)
118             (vx2,vy2,w2) = self.flow.sorFlow(CURRENT_F,NEXT_F)
119
120             m1 = self.flow.getFlowMagnitude(vx1,vy1)
121             self.index = self.index + 1
122             m2 = self.flow.getFlowMagnitude(vx2,vy2)
123
124             change_mag = abs(m2-m1)
125             binary_mag = np.ones(change_mag.shape,dtype=np.float64)
```

```
126             threshold = np.mean(change_mag , dtype=np.float64)
127             self.temp_flows.append(np.where(change_mag < threshold,0,
                    ↪ binary_mag))
128
129             if self.index>=int(FPS/3):
130                 vif = self.getFrameHist(CURRENT_F.shape)
131                 X_frame = np.empty((0,336))
132                 vif = np.reshape(vif, (-1, vif.shape[0]))
133                 X_frame = np.vstack((X_frame, vif))
134                 pred = self.model.predict(X_frame)
135                 print pred,time_now-start_time
136                 pred = round(pred[0][0])
137                 if pred == 1:
138                     print 'violent --- '+str(time_now - start_time)
```

A.12 Surveillance System Main File

test_package_surveillance.py

```
1 from ContSurv import ContinousSurv
2 if __name__ == '__main__':
3     obj = ContinousSurv()
4     # doing surveillance on a video file
5     # obj.setVideoName('testV2.avi')
6     # obj.doSurveillanceFromVideo()
7     # doing surveillance with the camera
8     obj.doSurveillanceFromCamera()
```