Certificate

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.

Contents

List of Figures

List of Tables

Chapter 1

Problem Definition

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 react quickly and cause less damage to civilians and the property.

Chapter 2

Introduction

2.1 Problem Statement

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.

2.2 Objective

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.

2.3 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.

2.4 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 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.

2.4.1 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 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.

2.5 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.

2.6 Organisation of Project Report

This project report is mainly divided into 6 modules as follows:

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

Chapter 3

Literature Survey

3.1 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 was first developed for mechanical television systems, which were quickly replaced by cathode ray tube (CRT) systems which were later replaced by flat panel displays of several types. Video systems vary in display resolution, aspect ratio, refresh rate, color capabilities and other qualities. Analog and digital variants exist and can be carried on a variety of media, including radio broadcast, magnetic tape, optical discs, computer files, and network streaming.

3.2 What is Video Processing

In electronics engineering, video processing is a particular case of signal processing, which often employs video filters and where the input and output signals are video files or video streams.

Video processing techniques are used in television sets, VCRs, DVDs, video codecs, video players, video scalers and other devices. For example, commonly only design and video processing is different in TV sets of different manufactures.

3.3 Analysis of Crowd

Crowd analysis involves the interpretation of data gained by studying the natural movement of groups or objects. Masses of bodies, particularly humans, are the subjects of these crowd tracking analyses that include how a particular crowd moves 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 population growth, crowd analysis has become a major interest in social and technical disciplines. Crowd analysis is being used to develop crowd management strategies in public events as well as public space design, visual surveillance and virtual environments to make areas more convenient in order to prevent crowd induced disasters.

3.3.1 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."

3.3.2 Challenges faced in crowd behaviour analysis

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

Video analysis and scene understanding usually involve object detection, tracking and behavior recognition. For crowded scenes, due to extreme clutters, severe occlusions and ambiguities, the conventional methods without special considerations are not appropriate. As Ali pointed out , the mechanics of human crowds are complex as a crowd exhibits both dynamics and psychological characteristics, which are often goal directed. 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, or a car that drives under a bridge. The problem in this case is what you do when an object disappears and reappears again.

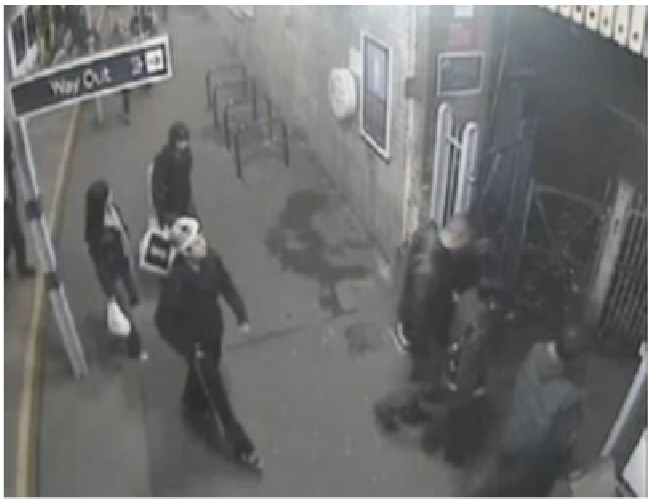


Figure 3.1: Occlusion Problem

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 if we consider the running activity, running in a marathon can be considered as a normal crowd behaviour but whereas running in a shopping mall or any such rare places can be considered as an abnormal behaviour.

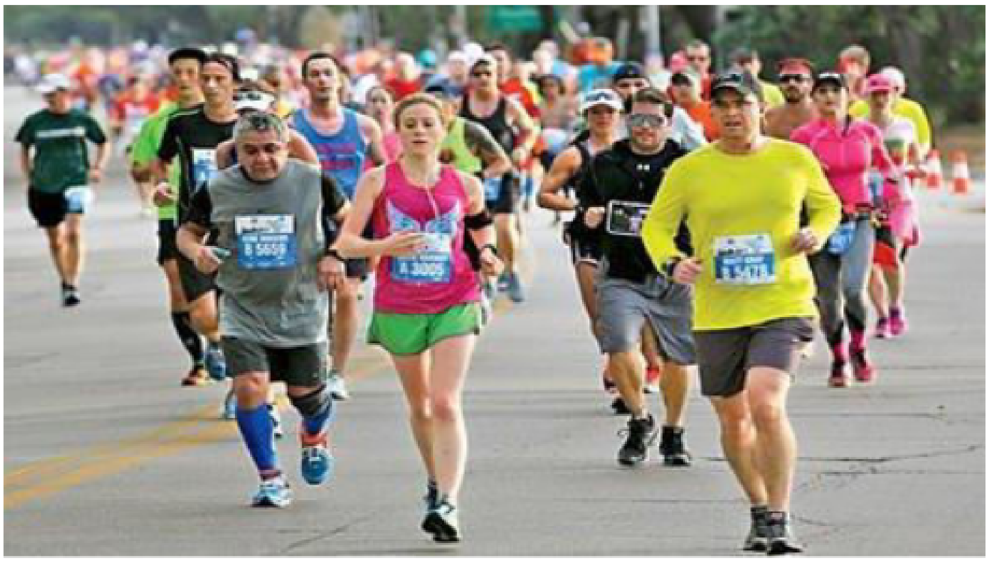


Figure 3.2: Crowd Running in Marathon

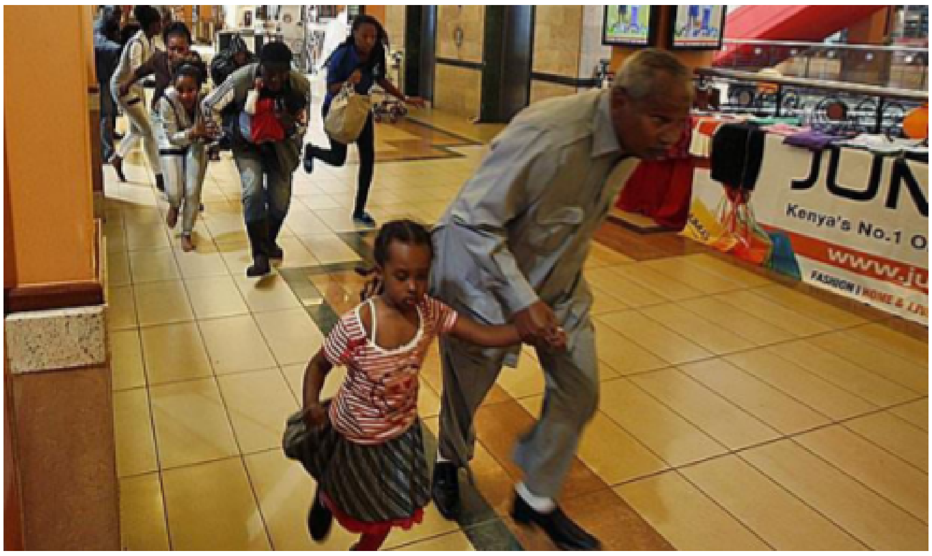


Figure 3.3: Crowd Running in a Mall

3.4 Optical Flow

3.4.1 What is 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.

3.4.2 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:

$$I(x, y, t) = T(x + \Delta x, y + \Delta y, t + \Delta t) \tag{3.1}$$

$$\frac{\partial I}{\partial x} V_x + \frac{\partial I}{\partial y} V_y + \frac{\partial I}{\partial t} = 0 \tag{3.2}$$

Chapter 4

Requirements, Analysis and Specifications

4.1 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 convenient platform for programmers developing software to be run on it and on other systems, rather than for non-programmers. 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.

- Kernel - source code in /usr/sys, composed of several sub-components.
- Development environment - early versions of Unix contained a development environment sufficient to recreate the entire system from source code.

- Commands - Unix makes little distinction between commands (user-level programs) for system operation and maintenance (e.g. cron), commands of general utility (e.g. grep), and more general-purpose applications such as the text formatting and typesetting package. Nonetheless, some major categories are listed below.

- Document formatting - Unix systems were used from the outset 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.

4.2 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 program-

ming. 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:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: 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.
- Databases: Python provides interfaces to all major commercial databases.
- 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.

- Scalable: Python provides a better structure and support for large programs 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.

4.3 OpenCV

OpenCV (Open Source Computer Vision Library) 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 startups 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

4.4 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

4.5 Keras and TensorFlow

4.5.1 Keras

Keras 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 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.

4.5.2 TensorFlow

TensorFlow 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.

4.6 Others

4.6.1 Libraries

Along with OpenCV we have used bob. Bob 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.

4.6.2 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.

4.6.3 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.

Chapter 5

Methodology

5.1 System Design

The project focuses on the implementation of an algorithm that can detect disturbance in crowd. 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.

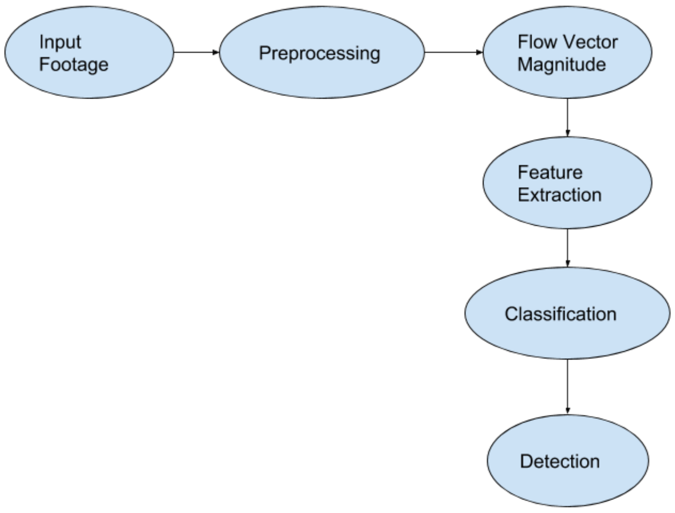


Figure 5.1: System Design

5.2 Implementation of Proposed Solution

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 Feature Extraction

Feature extraction is based on the computation of ViF (Violence Flow Descriptors). 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.

5.2.5 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.

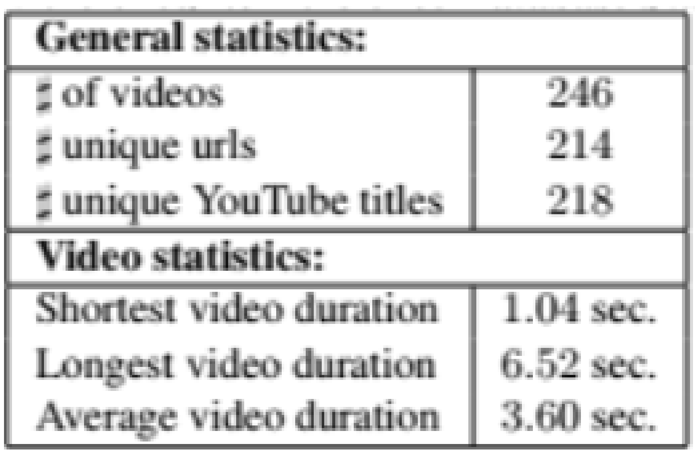
5.2.6 Detection

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.

Chapter 6

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. 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 6.1: Dataset

Chapter 7

Design

7.1 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.

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.

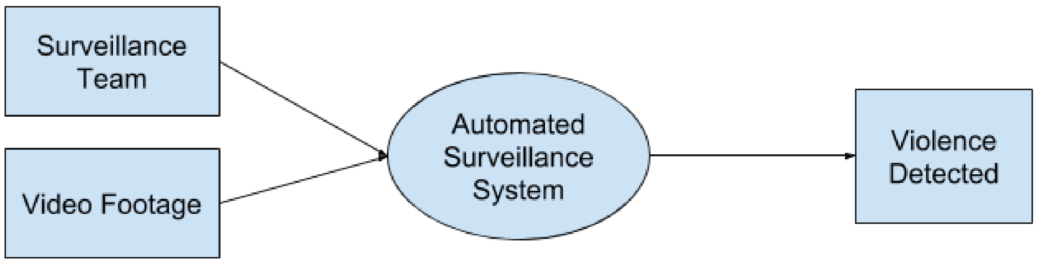


Figure 7.1: Data Flow Diagram Level 0

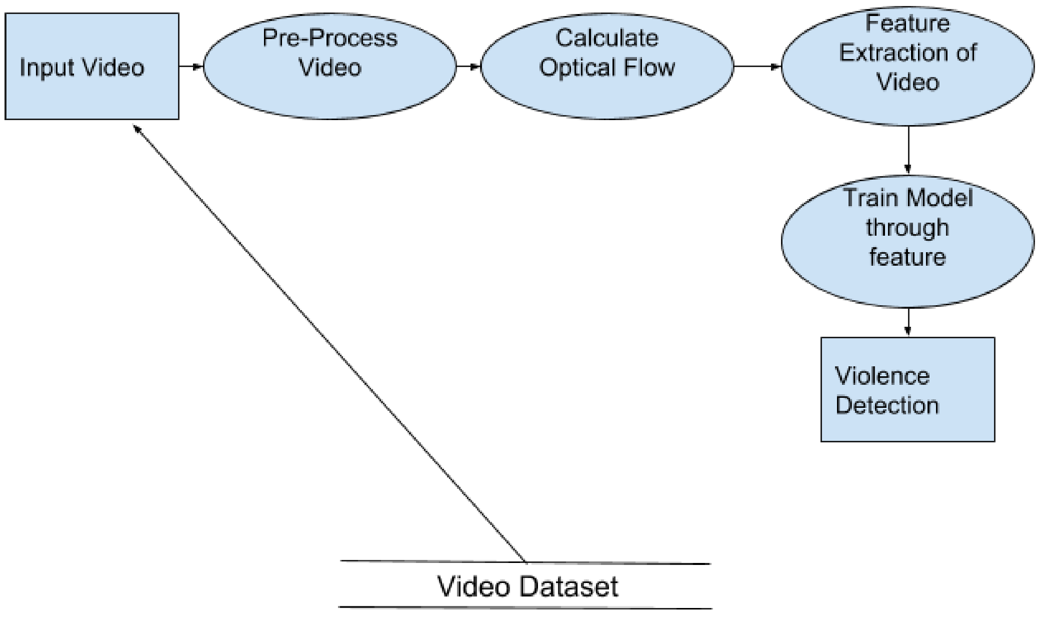


Figure 7.2: Data Flow Diagram Level 1

7.2 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.

7.2.1 Building blocks of UML

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

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

7.2.2 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.

7.2.3 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.

7.2.4 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.

7.2.5 Component Diagram

Automated Surveillance and Alert Generation System has the following components:

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

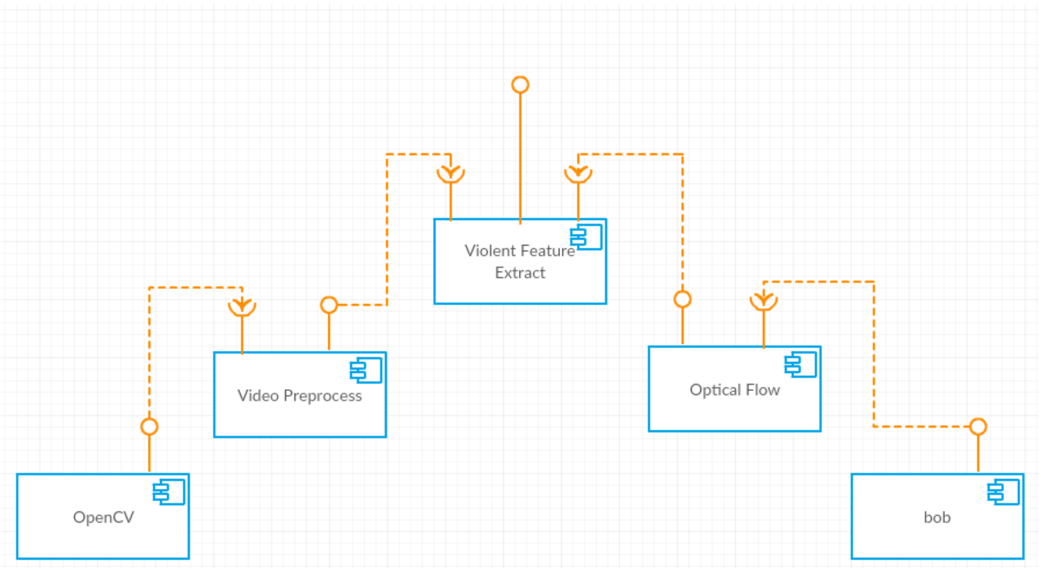


Figure 7.3: Component Diagram

OpenCV

OpenCV (short for Open Computer Vision) 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 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 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*.

Chapter 8

Implementation

8.1 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 maintaining the same aspect ratio using openCV functions. This resized frame is further converted to grayscale using openCV cvtColor() method.
Size reduction command:
frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
Grayscale conversion commands:
frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

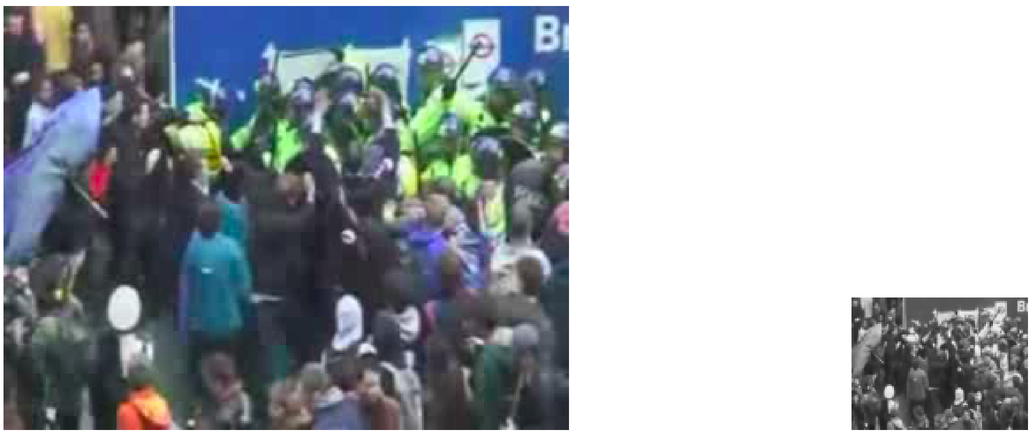


Figure 8.1: Frames before and after preprocessing respectively

8.2 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. C. Liu‘s Optical Flow algorithm 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.

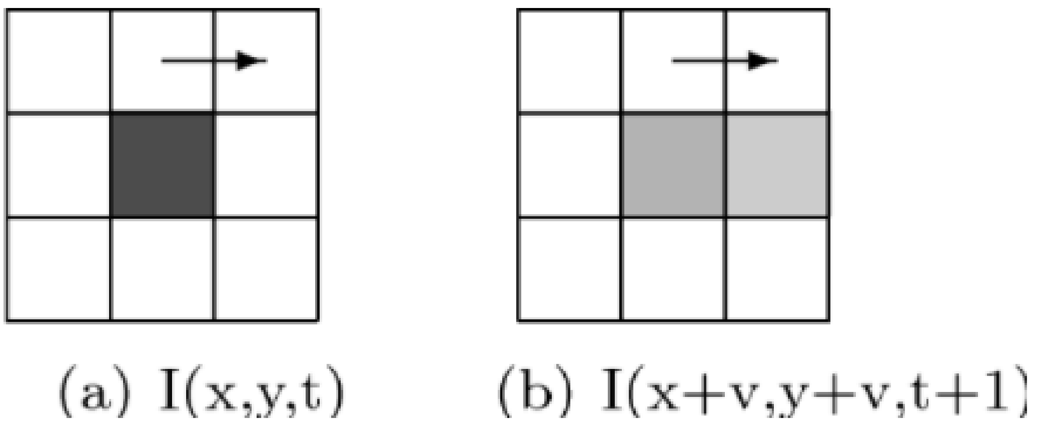


Figure 8.2: Example Optical Flow

8.3 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:-

$$b_{x,y,t} = \begin{cases} 1, & if \quad |m_{x,y,t+1} - m_{x,y,t-1}| \geq \theta \\ 0 & otherwise \end{cases} \tag{8.1}$$

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

$$\bar{b}_{x,y} = \frac{1}{T} \sum_t b_{x,y,t} \tag{8.2}$$

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.

8.4 Training Neural Net

Keras module along with TensorFlow 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. 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)
Sigmoid activation function
S(x) = $\frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}}$

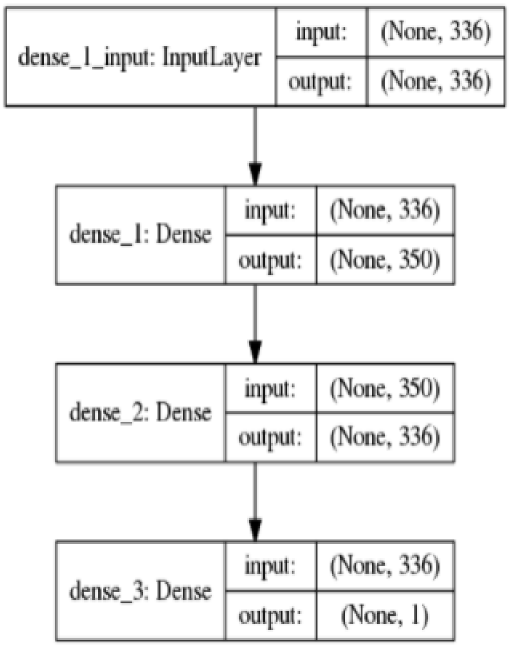


Figure 8.3: Neural Net Layer Information

8.5 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.

Chapter 9

Results

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

9.1 Video Preprocessing

Input frames are resized to 240 X 320. They are further converted to grayscale.

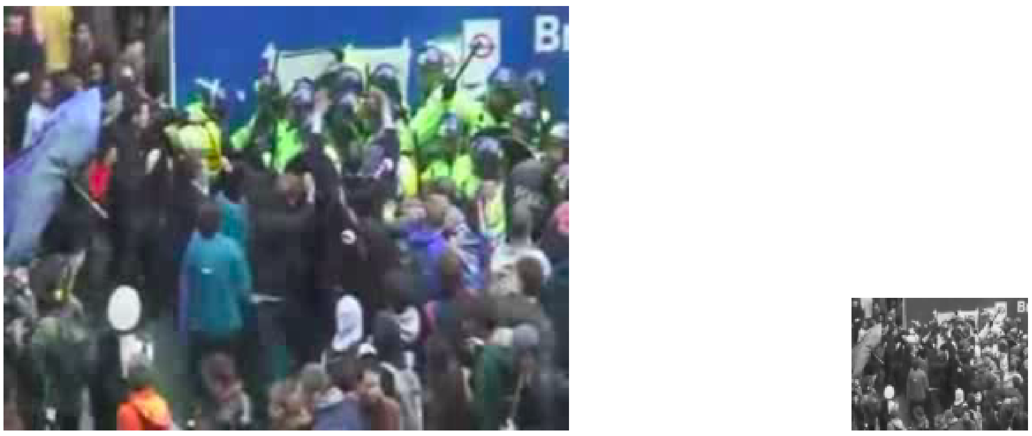


Figure 9.1: Frames before and after preprocessing

9.2 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.

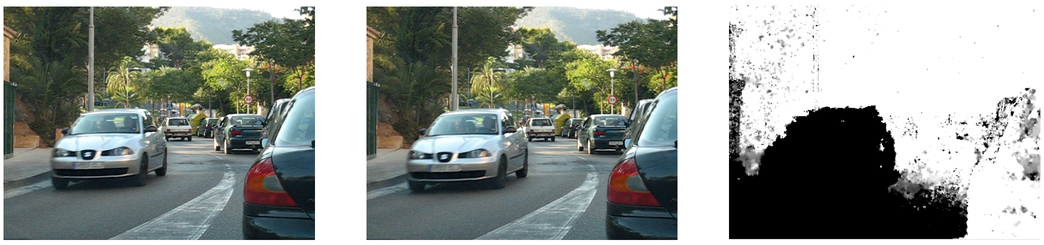


Figure 9.2: Optical Flow Car Example

9.3 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
0.000000000000000000e+00
0.000000000000000000e+00
1.420454545454545407e-02
7.670454545454545581e-02
0.000000000000000000e+00
1.846590909090909116e-01
0.000000000000000000e+00
2.329545454545454419e-01
0.000000000000000000e+00
1.704545454545454419e-01
0.000000000000000000e+00
1.647727272727272652e-01
9.943181818181817677e-02
0.000000000000000000e+00
3.409090909090908839e-02
0.000000000000000000e+00
2.272727272727272790e-02
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
```

```
0.000000000000000000e+00
0.000000000000000000e+00
8.522727272727272096e-03
5.681818181818181629e-02
0.000000000000000000e+00
1.079545454545454558e-01
0.000000000000000000e+00
1.676136363636363535e-01
0.000000000000000000e+00
2.159090909090909116e-01
0.000000000000000000e+00
1.704545454545454419e-01
1.477272727272727348e-01
0.000000000000000000e+00
7.670454545454545581e-02
0.000000000000000000e+00
3.125000000000000000e-02
0.000000000000000000e+00
1.136363636363636395e-02
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
0.000000000000000000e+00
1.335227272727272652e-01
1.107954545454545442e-01
0.000000000000000000e+00
1.534090909090909116e-01
0.000000000000000000e+00
1.534090909090909116e-01
0.000000000000000000e+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
0.000000000000000000e+00
1.903409090909090884e-01
0.000000000000000000e+00
1.818181818181818232e-01
0.000000000000000000e+00
1.988636363636363535e-01
1.392045454545454419e-01
0.000000000000000000e+00
6.250000000000000000e-02
0.000000000000000000e+00
1.704545454545454419e-02
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
1.420454545454545407e-02
0.000000000000000000e+00
5.397727272727272790e-02
0.000000000000000000e+00
7.386363636363636742e-02
0.000000000000000000e+00
1.619318181818181768e-01
0.000000000000000000e+00
2.130681818181818232e-01
2.471590909090909116e-01
0.000000000000000000e+00
1.477272727272727348e-01
0.000000000000000000e+00
6.250000000000000000e-02
0.000000000000000000e+00
1.988636363636363605e-02
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
0.000000000000000000e+00
2.840909090909090988e-03
2.556818181818181976e-02
```

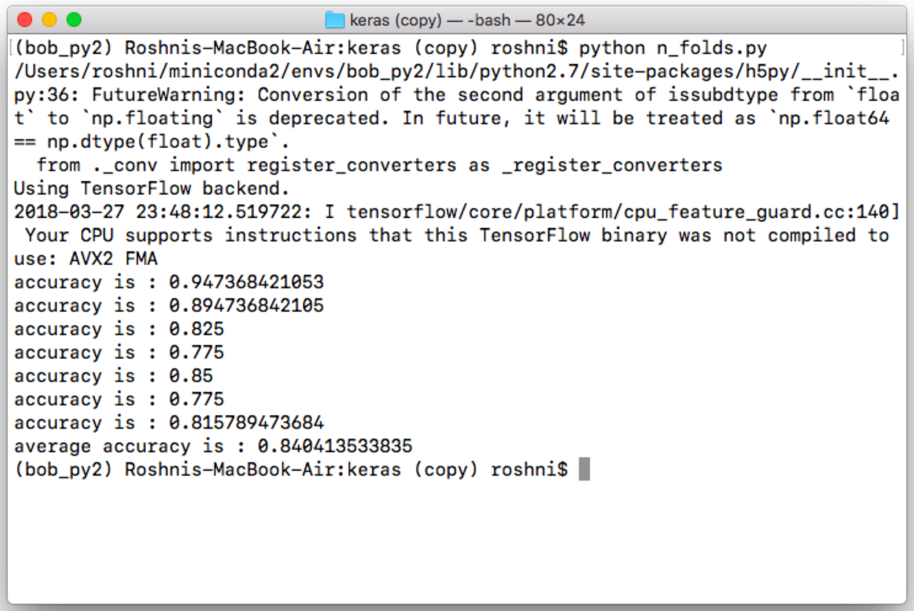
```
0.000000000000000000e+00
6.818181818181817677e-02
0.000000000000000000e+00
1.193181818181818232e-01
0.000000000000000000e+00
1.363636363636363535e-01
0.000000000000000000e+00
2.443181818181818232e-01
1.903409090909090884e-01
0.000000000000000000e+00
1.278409090909090884e-01
0.000000000000000000e+00
3.977272727272727210e-02
0.000000000000000000e+00
3.977272727272727210e-02
0.000000000000000000e+00
5.681818181818181976e-03
0.000000000000000000e+00
0.000000000000000000e+00
5.681818181818181976e-03
2.840909090909090814e-02
0.000000000000000000e+00
6.250000000000000000e-02
0.000000000000000000e+00
1.420454545454545581e-01
0.000000000000000000e+00
1.846590909090909116e-01
0.000000000000000000e+00
2.357954545454545581e-01
1.647727272727272652e-01
0.000000000000000000e+00
1.193181818181818232e-01
0.000000000000000000e+00
4.545454545454545581e-02
0.000000000000000000e+00
1.136363636363636395e-02
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
8.522727272727272096e-03
7.102272727272727904e-02
0.000000000000000000e+00
```

```
1.392045454545454419e-01
0.000000000000000000e+00
1.931818181818181768e-01
0.000000000000000000e+00
2.272727272727272652e-01
0.000000000000000000e+00
1.903409090909090884e-01
1.278409090909090884e-01
0.000000000000000000e+00
3.977272727272727210e-02
0.000000000000000000e+00
2.840909090909090988e-03
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
2.840909090909090988e-03
5.113636363636363952e-02
0.000000000000000000e+00
1.107954545454545442e-01
0.000000000000000000e+00
1.505681818181818232e-01
0.000000000000000000e+00
2.073863636363636465e-01
0.000000000000000000e+00
2.187500000000000000e-01
1.562500000000000000e-01
0.000000000000000000e+00
7.386363636363636742e-02
0.000000000000000000e+00
1.988636363636363605e-02
0.000000000000000000e+00
8.522727272727272096e-03
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
8.522727272727272096e-03
1.988636363636363605e-02
0.000000000000000000e+00
1.534090909090909116e-01
```

```
0.000000000000000000e+00
1.931818181818181768e-01
0.000000000000000000e+00
2.187500000000000000e-01
0.000000000000000000e+00
1.988636363636363535e-01
1.107954545454545442e-01
0.000000000000000000e+00
6.250000000000000000e-02
0.000000000000000000e+00
2.272727272727272790e-02
0.000000000000000000e+00
1.136363636363636395e-02
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
1.136363636363636395e-02
7.954545454545454419e-02
0.000000000000000000e+00
8.522727272727272096e-02
0.000000000000000000e+00
2.102272727272727348e-01
0.000000000000000000e+00
2.045454545454545581e-01
0.000000000000000000e+00
1.960227272727272652e-01
9.090909090909091161e-02
0.000000000000000000e+00
7.102272727272727904e-02
0.000000000000000000e+00
3.693181818181818371e-02
0.000000000000000000e+00
1.420454545454545407e-02
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
```

9.4 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.

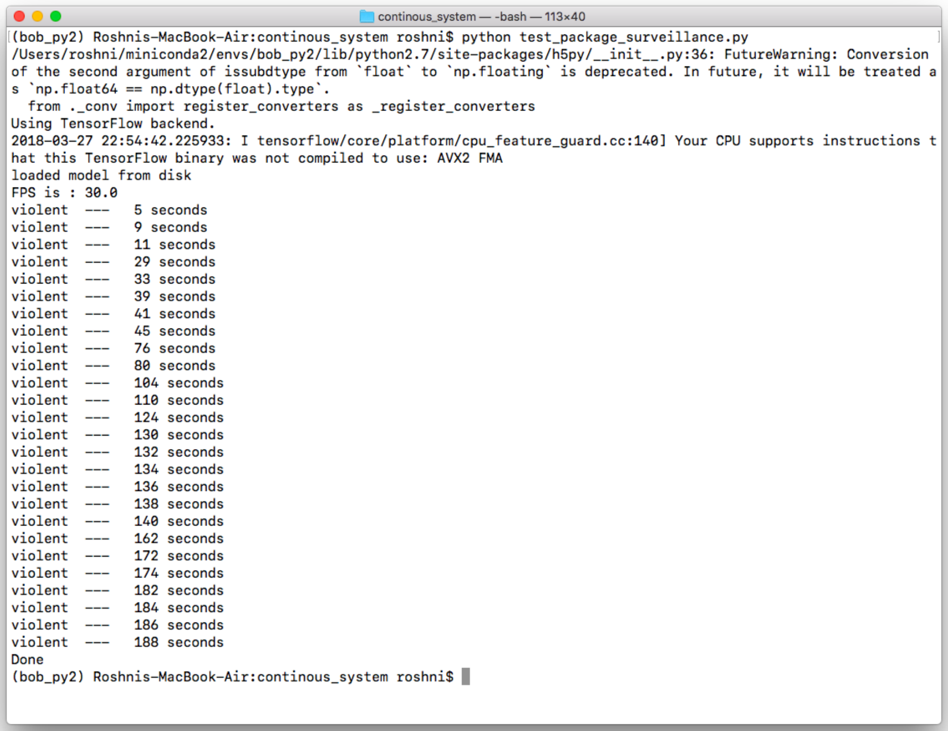


```
(bob_py2) Roshnis-MacBook-Air:keras (copy) roshni$ python n_folds.py
/Users/roshni/miniconda2/envs/bob_py2/lib/python2.7/site-packages/h5py/__init__.
py:36: FutureWarning: Conversion of the second argument of issubdtype from `floa
t` to `np.floating` is deprecated. In future, it will be treated as `np.float64
== np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-03-27 23:48:12.519722: I tensorflow/core/platform/cpu_feature_guard.cc:140]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
accuracy is : 0.947368421053
accuracy is : 0.894736842105
accuracy is : 0.825
accuracy is : 0.775
accuracy is : 0.85
accuracy is : 0.775
accuracy is : 0.815789473684
average accuracy is : 0.840413533835
(bob_py2) Roshnis-MacBook-Air:keras (copy) roshni$ 
```

Figure 9.3: Neural Net Folds Accuracy

9.5 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. So with the help of this we are able to decide the exact second where the violence occurs.

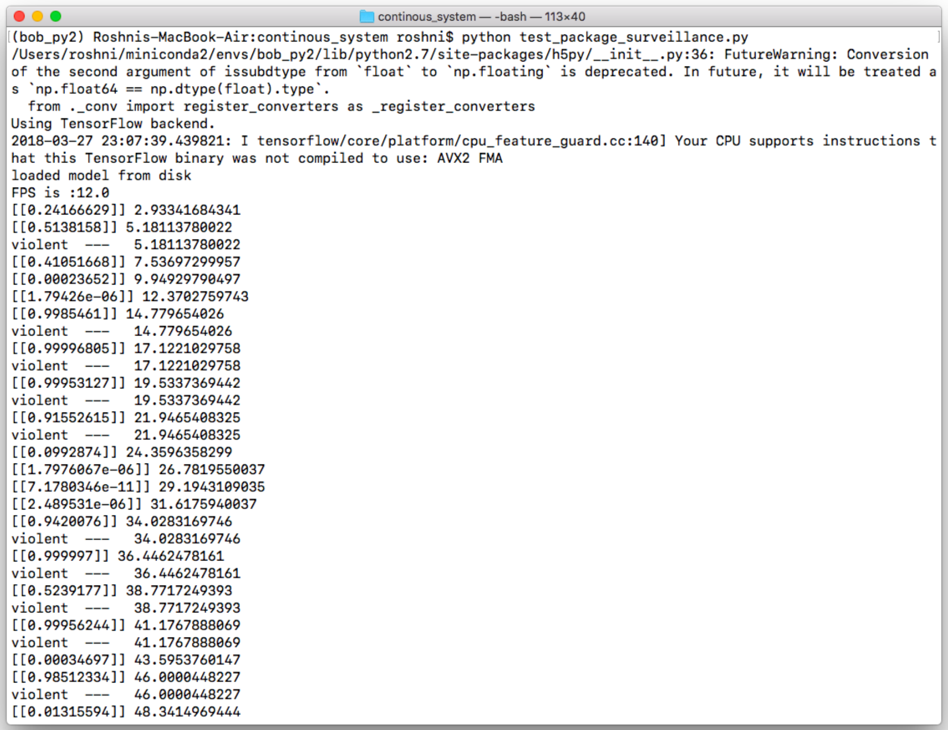


```
(bob_py2) Roshnis-MacBook-Air:continous_system roshni$ python test_package_surveillance.py
/Users/roshni/miniconda2/envs/bob_py2/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conversion
of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated a
s `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-03-27 22:54:42.225933: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions t
hat this TensorFlow binary was not compiled to use: AVX2 FMA
loaded model from disk
FPS is : 30.0
violent  ---   5 seconds
violent  ---   9 seconds
violent  ---   11 seconds
violent  ---   29 seconds
violent  ---   33 seconds
violent  ---   39 seconds
violent  ---   41 seconds
violent  ---   45 seconds
violent  ---   76 seconds
violent  ---   80 seconds
violent  ---   104 seconds
violent  ---   110 seconds
violent  ---   124 seconds
violent  ---   130 seconds
violent  ---   132 seconds
violent  ---   134 seconds
violent  ---   136 seconds
violent  ---   138 seconds
violent  ---   140 seconds
violent  ---   162 seconds
violent  ---   172 seconds
violent  ---   174 seconds
violent  ---   182 seconds
violent  ---   184 seconds
violent  ---   186 seconds
violent  ---   188 seconds
Done
(bob_py2) Roshnis-MacBook-Air:continous_system roshni$ 
```

Figure 9.4: Surveillance on Video

9.6 Real Time Surveillance on Camera Input

OpenCV 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.



```
continous_system — -bash — 113×40
(bob_py2) Roshnis-MacBook-Air:continous_system roshni$ python test_package_surveillance.py
/Users/roshni/miniconda2/envs/bob_py2/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conversion
of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated a
s `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-03-27 23:07:39.439821: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions t
hat this TensorFlow binary was not compiled to use: AVX2 FMA
loaded model from disk
FPS is :12.0
[[0.24166629]] 2.93341684341
[[0.5138158]] 5.18113780022
violent  ---   5.18113780022
[[0.41051668]] 7.53697299957
[[0.00023652]] 9.94929790497
[[1.79426e-06]] 12.3702759743
[[0.9985461]] 14.779654026
violent  ---   14.779654026
[[0.99996805]] 17.1221029758
violent  ---   17.1221029758
[[0.99953127]] 19.5337369442
violent  ---   19.5337369442
[[0.91552615]] 21.9465408325
violent  ---   21.9465408325
[[0.0992874]] 24.3596358299
[[1.7976067e-06]] 26.7819550037
[[7.1780346e-11]] 29.1943109035
[[2.489531e-06]] 31.6175940037
[[0.9420076]] 34.0283169746
violent  ---   34.0283169746
[[0.999997]] 36.4462478161
violent  ---   36.4462478161
[[0.5239177]] 38.7717249393
violent  ---   38.7717249393
[[0.99956244]] 41.1767888069
violent  ---   41.1767888069
[[0.00034697]] 43.5953760147
[[0.98512334]] 46.0000448227
violent  ---   46.0000448227
[[0.01315594]] 48.3414969444
```

Figure 9.5: Surveillance through Camera

9.7 Result Ananlysis

Title	**Methodology**	**Results**	**Merits**	**Demerits**
Violence detection using Oriented Violent Flows, 2016 [1]	AdaBoost and SVM classifier.	88.00 percent	Feature representation model, which depicts the information involving both the motion magnitude and motion orientation.	Detection point where the behaviour is changing from normal to abnormal is time consuming hence not applicable in real time scenerios.
Violent Flows:Real-Time Detection of Violent Crowd Behaviour, 2012 [2]	Global descriptors and SVM classifier	5-fold cross validation: 81.30 percent	The algorithm detected far more violent scenes correctly, compared to existing work. It was furthermore far faster to detect the violence, typically in less than a second from its outbreak	Only magnitude of the flow vectors is considered, but the direction is not.

Title	**Methodology**	**Results**	**Merits**	**Demerits**
Automatic Fight Detection in Surveillance Videos, 2016 [3]	Motion magnitude, motion acceleration and strength of motion region relationship, collectively known as motion signals	10 fold cross validation: 82.70 percent	Difference between stimulated fights and real fights. Doesnt rely on high level behaviour recognition, Thus applicable to Low quality videos.	Less accuracy is achieved when testing with real fight scenarios
Online real-time crowd behaviour detection in video sequences, 2015 [4]	Instant entropy and temporal occupancy variation	96 percent	Works without the need of training phase.	Computational speed (FPS) is varying for different datasets. Not robust with different types of data. Only able to give the percentage of violence. Not ideal for generating alerts. Crowded Scenes are considered as violent.

Title	**Methodology**	**Results**	**Merits**	**Demerits**
Proposed Algorithm	Using Global Descriptors (ViFs) along with Neural Network to accurately predict disturbance in crowd.	85.00 percent	Highly Scalable, can process multiple videos at a time using GPUs. Accuracy can be greatly imporved by increasing weights to well performing features. Input from human surveyor can continously train the model for better performance in future.	Only works for avi videos. Neural Net initial training takes time.

Table 9.1: Result Analysis

Chapter 10

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.

Chapter 11

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.

References

[1] 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

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

[3] 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

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

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

[6] Keras: `https://keras.io/`

[7] TensorFlow: `https://en.wikipedia.org/wiki/TensorFlow`

Appendix A

Code

A.1 Video Preprocessing

process.py

```python
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])
        if rescale<0.8:
            dim = (self.dim, int(frame.shape[0] * rescale))
            frame = cv2.resize(frame, dim, interpolation = cv2.
                ↪ INTER_AREA)
        return frame

    def setVideoDimension(self,dim):
        self.dim = dim

    def useCamera(self):
        self.cap = cv2.VideoCapture(0)
        self.last_frame = self.read_frame()

    def showInputFromCamera(self):
        while True:
            ret , frame = self.cap.read()
            cv2.imshow('camera_frame',frame)
            cv2.imshow('resized_camera_frame',self.resize_frame(cv2
                ↪ .cvtColor(frame,cv2.COLOR_BGR2GRAY)))

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    def read_frame(self):
        ret , frame = self.cap.read()
        return frame

    def getFPS(self):
        return self.cap.get(cv2.CAP_PROP_FPS)

    def getFramesFromVideoSource(self):
        PREV_F = self.getFrameFromIndex(self.frame_number)
        CURRENT_F = self.getFrameFromIndex(self.frame_number +
            ↪ self.MOVEMENT_INTERVAL)
        NEXT_F = self.getFrameFromIndex(self.frame_number + (2 *
            ↪ self.MOVEMENT_INTERVAL))

        frames = (PREV_F,CURRENT_F,NEXT_F,self.frame_number)
        self.frame_number += 6

        return frames

    def getFramesFromCameraSource(self):
        frame1 = self.last_frame
        for i in range(0,self.MOVEMENT_INTERVAL-1):
            self.read_frame()
        frame2 = self.read_frame()
        for i in range(0,self.MOVEMENT_INTERVAL-1):
            self.read_frame()
        frame3 = self.read_frame()
        self.last_frame = frame3

        # cv2.imshow('camera_frame',frame2)

        frame1 = self.resize_frame(frame1)
        frame2 = self.resize_frame(frame2)
        frame3 = self.resize_frame(frame3)

        frame1 = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
        frame2 = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
        frame3 = cv2.cvtColor(frame3,cv2.COLOR_BGR2GRAY)

```

```python
        # cv2.imshow('resized_camera_frame',frame2)

        frames = (frame1,frame2,frame3,time.time())

        return frames
```

A.2 Optical Flow

flow.py

```python
import bob.ip.optflow.liu.sor
import numpy as np
class OptFlow:
    def __init__(self):
        self.alpha = 0.0026
        self.ratio = 0.6
        self.minWidth = 20
        self.nOuterFPIterations = 7
        self.nInnerFPIterations = 1
        self.nSORIterations = 30
        self.flows = ()

    def sorFlow(self,frame1,frame2):
        self.flows = bob.ip.optflow.liu.sor.flow(frame1,frame2,
            ↪ self.alpha,self.ratio,self.minWidth,self.
            ↪ nOuterFPIterations,self.nInnerFPIterations,self.
            ↪ nSORIterations)
        #self.flows = bob.ip.optflow.liu.sor.flow(frame1,frame2)
        return self.flows


    def getFlowMagnitude(self,vx,vy):
        flow_magnitude = np.sqrt(np.square(vx) + np.square(vy))
        return flow_magnitude
```

A.3 Generate ViFs for Violent Videos

violent_features_VIOLENT.py

```python
from ViolentFlow import VioFlow
```

```python
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

```python
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

```python
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:
            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 < 92:
                X_train = np.vstack((X_train,vif))
                Y_train = np.append(Y_train,0)
            else:
                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 < 92:
                X_train = np.vstack((X_train, vif))
                Y_train = np.append(Y_train, 1)
            else:
                X_test = np.vstack((X_test, vif))
                Y_test = np.append(Y_test, 1)
            file_obj.close()
            count += 1
```

```python
        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(1,7):
        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)

    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))
```

A.8 Training Neural Net and Storing on disk

keras_train_100_once_test_30_multiple_store_model.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

data = range(1,130)
random.shuffle(data)
X_train = np.empty((0,336))
Y_train = np.array([])
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]))

        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]

```

```python

    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

```python
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):
        try:
            if j in test_set:
                continue
            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_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’%j
    for j in range(1,130):
        try:
            if j in test_set:
                continue
            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_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’%j

    if len(X_train) == 0:
        iters -= 1
        continue

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

    for l in range(1,5):
        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)

    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 = []
        self.bins = np.arange(0.0,1.05,0.05,dtype=np.float64)

```

```python
    def setVideoName(self,video_name):
        self.vid.read_video(video_name)

    def histc(self,X, bins):
        map_to_bins = np.digitize(X,bins)
        r = np.zeros(bins.shape,dtype=np.float64)
        for i in map_to_bins:
            r[i-1] += 1
        return r

    def getBlockHist(self,flow_video):
        flow_vec = np.reshape(flow_video,(flow_video.shape[0]*
            ↪ flow_video.shape[1],1))
        count_of_bins = self.histc(flow_vec,self.bins)
        return count_of_bins/np.sum(count_of_bins)

    def getFrameHist(self,flow_video_size):
        flow_video = np.zeros(flow_video_size,dtype=np.float64)
        for each_flow in self.temp_flows:
            flow_video = flow_video + each_flow
        flow_video = flow_video / self.index
        self.index = 0
        self.temp_flows = []
        self.height = flow_video.shape[0]
        self.width = flow_video.shape[1]
        self.B_height = int(math.floor((self.height - 11)/4))
        self.B_width = int(math.floor((self.width - 11)/4))
        frame_hist = []
        for y in range(6,self.height-self.B_height-4,self.B_height
            ↪ ):
            for x in range(6,self.width-self.B_width-4,self.B_width
                ↪ ):
                block_hist = self.getBlockHist(flow_video[y:y+self.
                    ↪ B_height,x:x+self.B_width])
                frame_hist = np.append(frame_hist,block_hist,axis =
                    ↪  0)
        return frame_hist

    def doSurveillanceFromVideo(self):
        FPS = round(self.vid.getFPS())
        print 'FPS is : '+str(FPS)
        while True:
            frames = self.vid.getFramesFromVideoSource()
            PREV_F = frames[0]
            CURRENT_F = frames[1]
            NEXT_F = frames[2]

            frame_number = frames[3]

            PREV_F = self.vid.resize_frame(PREV_F)
            CURRENT_F = self.vid.resize_frame(CURRENT_F)
            NEXT_F = self.vid.resize_frame(NEXT_F)

            (vx1,vy1,w1) = self.flow.sorFlow(PREV_F,CURRENT_F)
            (vx2,vy2,w2) = self.flow.sorFlow(CURRENT_F,NEXT_F)

            m1 = self.flow.getFlowMagnitude(vx1,vy1)
            self.index = self.index + 1
            m2 = self.flow.getFlowMagnitude(vx2,vy2)

            change_mag = abs(m2-m1)
            binary_mag = np.ones(change_mag.shape,dtype=np.float64)
            threshold = np.mean(change_mag , dtype=np.float64)
            self.temp_flows.append(np.where(change_mag < threshold ,0,binary_mag))

            if self.index>=int(FPS/3):
                vif = self.getFrameHist(CURRENT_F.shape)
                X_frame = np.empty((0,336))
                vif = np.reshape(vif, (-1, vif.shape[0]))
                X_frame = np.vstack((X_frame, vif))
                pred = self.model.predict(X_frame)
                pred = round(pred[0][0])
                if pred == 1:
                    time_violence = float(frame_number) / self.vid.fps
                    print 'violent --- '+str(int(time_violence))+' seconds'

    def doSurveillanceFromCamera(self):
        start_time = time.time()
        self.vid.useCamera()
        FPS = round(self.vid.getFPS())
        print 'FPS is :'+str(FPS)
        while True:
            frames = self.vid.getFramesFromCameraSource()
            PREV_F = frames[0]
            CURRENT_F = frames[1]
            NEXT_F = frames[2]

            time_now = frames[3]

            PREV_F = self.vid.resize_frame(PREV_F)
            CURRENT_F = self.vid.resize_frame(CURRENT_F)
            NEXT_F = self.vid.resize_frame(NEXT_F)

            (vx1,vy1,w1) = self.flow.sorFlow(PREV_F,CURRENT_F)
            (vx2,vy2,w2) = self.flow.sorFlow(CURRENT_F,NEXT_F)

            m1 = self.flow.getFlowMagnitude(vx1,vy1)
            self.index = self.index + 1
            m2 = self.flow.getFlowMagnitude(vx2,vy2)

            change_mag = abs(m2-m1)
            binary_mag = np.ones(change_mag.shape,dtype=np.float64)
            threshold = np.mean(change_mag , dtype=np.float64)
            self.temp_flows.append(np.where(change_mag < threshold
                ↪ ,0,binary_mag))

            if self.index>=int(FPS/3):
                vif = self.getFrameHist(CURRENT_F.shape)
                X_frame = np.empty((0,336))
                vif = np.reshape(vif, (-1, vif.shape[0]))
                X_frame = np.vstack((X_frame, vif))
                pred = self.model.predict(X_frame)
                print pred,time_now-start_time
                pred = round(pred[0][0])
                if pred == 1:
                    print 'violent --- '+str(time_now - start_time)
```

A.12 Surveillance System Main File

test_package_surveillance.py

```
from ContSurv import ContinousSurv
if __name__ == '__main__':
    obj = ContinousSurv()
    # doing surveillance on a video file
    # obj.setVideoName('testV2.avi')
    # obj.doSurveillanceFromVideo()
    # doing surveillance with the camera
    obj.doSurveillanceFromCamera()
```