Certificate

Declaration

Acknowledgments

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 occurs in crowd, all the people

in present 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 of detection 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.

2.5 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 screen and any disturbance caused must be manually identified.
- Much research has been done in person to person fight detection, sports violence detection, violence detection in movies and slow motion fight detection and it is automated.

2.5.1 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.

2.6 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.

2.7 Organisation of Project

This project is mainly divided into 6 modules as follows:

- The second chapter discusses about the literature survey of this project which includes an insight into the core part of our project along with the technologies used.
- The third chapter deals with the design of our proposed system. The fourth chapter deals with the implementation of our system which discusses about the algorithms used in building our system.
- The fifth chapter displays our results and discussions through a series of screenshots. The sixth chapter talks about the conclusions and the future scope of our project.

Chapter 3

Literature Survey

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.
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.
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.	

Table 3.1: Literature Survey

Chapter 4

Methodology

4.1 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.

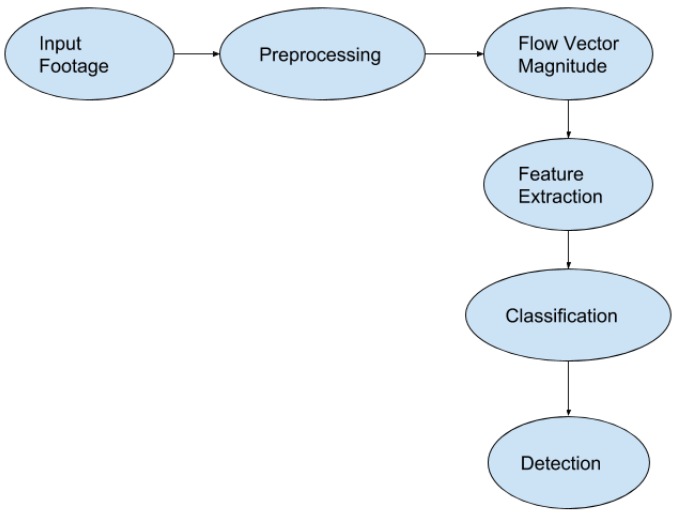

Figure 4.1: System Design

4.2 Implementation of Proposed Solution

4.2.1 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.

4.2.2 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 pixel height and respective width accordingly. Preprocessing is done in a way such that the aspect ratio of the video is not disturbed.

4.2.3 Flow Vector Magnitude

For 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, this is the most time taking part of the proposed algorithm. Since we are reducing the size of frames considerably, this is computation is quick enough to cope us with real time violence detection.

4.2.4 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 to 1.0) with the intervals of 0.05, i.e 20 bins.

4.2.5 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.

4.2.6 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 continuous learning in real-time also.

4.3 Languages and Tools

Only open source languages and tools are used in developing the project. Unix environment is being used for the development of the process.

4.3.1 Language

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:

- 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.

4.3.2 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
- Humancomputer 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

4.3.3 Other 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.

Chapter 5

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 5.1: Dataset

Chapter 6

Design

6.1 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 6.1: Data Flow Diagram Level 0

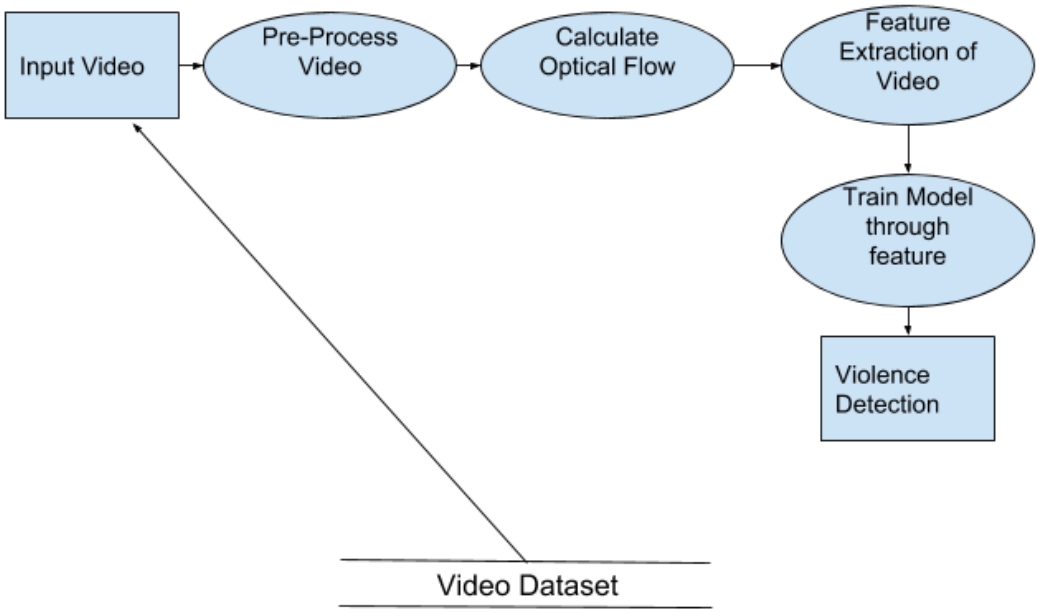

Figure 6.2: Data Flow Diagram Level 1

6.2 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 objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

6.2.1 Building blocks of UML

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

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

6.2.2 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. Structure things.
2. Behavioural things.
3. Grouping things.
4. Annotational things.

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

6.2.3 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.

6.2.4 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.

6.2.5 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

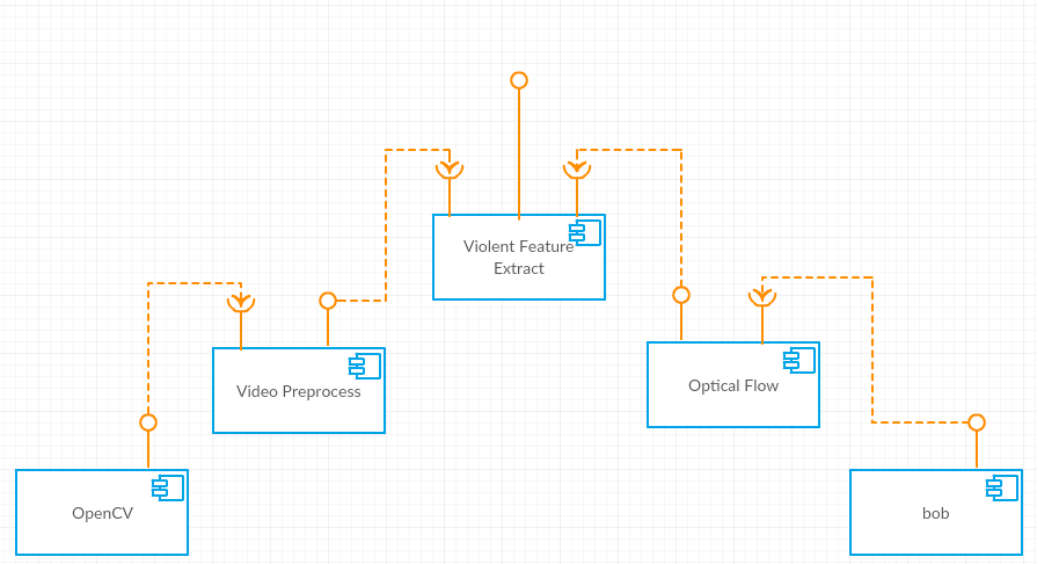

Figure 6.3: Component Diagram

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 Lius 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 by 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 , 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 7

Implementation

7.1 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 7.1: Frames before and after preprocessing respectively

7.2 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. Lius 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.

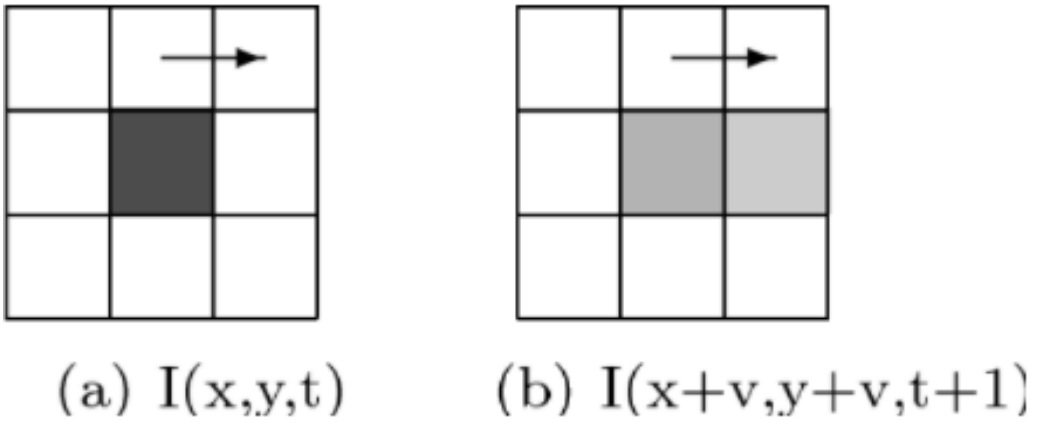

Figure 7.2: Example Optical Flow

7.3 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:-

$$b_{x,y,t} = \begin{cases} 1, & if \quad |m_{x,y,t+1} - m_{x,y,t-1}| \geq \theta \\ 0 & otherwise \end{cases} \tag{7.1}$$

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

$$\bar{b}_{x,y} = \frac{1}{T} \sum_{t} b_{x,y,t} \tag{7.2}$$

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.

7.4 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
$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$

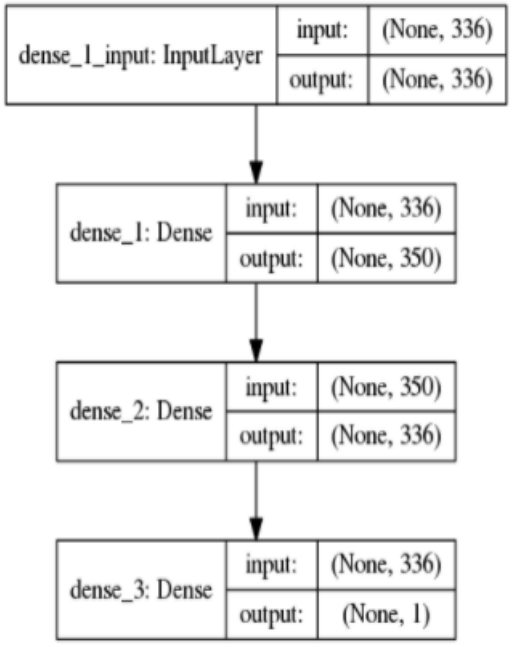

Figure 7.3: Neural Net Layer Information

7.5 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 an input to the program and it is able to detect the instance where crowd behaviour tends to violent from non_violent.

Chapter 8

Code

8.1 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):
```

```
25          self.cap = cv2.VideoCapture(video_name)
26          self.total_frames = int(self.cap.get(cv2.
                ↪ CAP_PROP_FRAME_COUNT))
27          self.fps = self.cap.get(cv2.CAP_PROP_FPS)
28          self.time = self.total_frames / self.fps
29          # self.last_frame = self.read_frame()
30
31      def getFrameFromIndex(self,frame_no):
32          #Number 2 defines flag CV_CAP_PROP_POS_FRAMES which is a
                ↪ 0-based index of the frame to be decoded/captured
                ↪ next.
33          #The second argument defines the frame number in range
                ↪ 0.0-1.0
34          self.cap.set(1,frame_no)
35          ret , img = self.cap.read()
36          if img is None:
37              sys.exit('Done')
38          img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
39          return img
40
41      def resize_frame(self,frame):
42          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
```

8.2 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
```

8.3 Generate ViFs for Violent Videos

violent_features_VIOLENT.py

```
1 from ViolentFlow import VioFlow
2 import time
3 file_vio = open('violent_list.txt')
4 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))
```

8.4 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))
```

8.5 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)
```

8.6 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
```

```
82
83      pred = []
84
85      for i in X_test:
86          pred.append(clf.predict(i.reshape(1,-1)))
87
88      count = 0
89
90      for i in range(0,len(Y_test)):
91          if pred[i][0] == Y_test[i]:
92              count = count + 1
93
94      total_accuracy += float(count)/len(Y_test)
95      print 'accuracy is : '+str(float(count)/len(Y_test))
96
97  print 'average accuracy is : ' + str(total_accuracy/iters)
```

8.7 Training Neural Net and Testing Accuracy

keras_neural_networks_training_70_30_random_20avg.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   ovr_acc = 0.0
9   start_time = time.time()
10  for j in range(1,21):
11      X_train = np.empty((0,336))
12      Y_train = np.array([])
13      X_test = np.empty((0,336))
14      Y_test = np.array([])
15      count = 0
16      data = range(1,130)
17      random.shuffle(data)
18
19      for i in data:
```

```
20          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))
```

8.8 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]))

```

```
39          X_train = np.vstack((X_train, vif))
40          Y_train = np.append(Y_train, 1)
41
42          file_obj.close()
43      except:
44          continue
45          print 'error in reading vio_%d.txt' % i
46
47  seed = 7
48  np.random.seed(seed)
49  model = Sequential()
50  model.add(Dense(350, activation="relu", kernel_initializer="
        ↪ uniform", input_dim=336))
51
52  for l in range(1,2):
53      model.add(Dense(336, activation='relu', kernel_initializer="
            ↪ uniform"))
54  model.add(Dense(1, activation="sigmoid", kernel_initializer="
        ↪ uniform"))
55
56  model.compile(loss='binary_crossentropy', optimizer='adam',
        ↪ metrics=['accuracy'])
57
58  model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)
59
60  print 'model trained'
61
62  ovr_acc = 0.0
63  start_time = time.time()
64  for j in range(1,21):
65      random.shuffle(data)
66      X_test = np.empty((0,336))
67      Y_test = np.array([])
68      count = 0
69      for i in data:
70          try:
71              file_name = 'violent_features_NON_VIOLENT/nonvio_' +
                    ↪ str(i) + '.txt'
72              file_obj = open(file_name, 'r')
73              vif = np.loadtxt(file_obj)
74              if vif.shape[0] == 630: # avoiding hd videos
75                  continue
76              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
```

```
119
120     cm = confusion_matrix(Y_test, pred)
121     print cm
122
123     accuracy = float(acc_count)/len(pred)
124     print 'accuracy is : ' + str(accuracy)
125     ovr_acc += accuracy
126
127 print 'overall : ' + str(ovr_acc/20.0)
128 print("--- %s seconds ---" % (time.time() - start_time))
129
130 model_json = model.to_json()
131
132 with open("model_100.json", "w") as json_file:
133     json_file.write(model_json)
134
135 model.save_weights("model_100.h5")
136 print("Saved model to disk")
```

8.9 Load Neural Net from Disk and Visualize

keras_visualize.py

```
1  from keras.models import model_from_json
2  from keras.utils.vis_utils import plot_model, model_to_dot
3  from IPython.display import SVG,display_svg
4
5  # load model
6  json_file = open('model_100.json', 'r')
7  loaded_model_json = json_file.read()
8  json_file.close()
9  loaded_model = model_from_json(loaded_model_json)
10
11 # load model weights
12 loaded_model.load_weights("model_100.h5")
13
14 plot_model(loaded_model, to_file='model_plot.png', show_shapes=
       ↪ True, show_layer_names=True)
15
16 display_svg(SVG(model_to_dot(loaded_model).create(prog='dot',
       ↪ format='svg')))
```

8.10 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]
```

```
37          try:
38              file_name = 'violent_features_VIOLENT/vio_'+str(
                    ↪ data_violent[j])+'.txt'
39              file_obj = open(file_name,'r')
40              vif = np.loadtxt(file_obj)
41              if vif.shape[0] == 630:# avoiding hd videos
42                  continue
43              vif = np.reshape(vif,(-1,vif.shape[0]))
44              X_test = np.vstack((X_test,vif))
45              Y_test = np.append(Y_test,1)
46              file_obj.close()
47          except:
48              continue
49              print 'error in reading nonvio_%d.txt'%data_violent[i]
50      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
97      model.compile(loss='binary_crossentropy', optimizer='adam',
            ↪ metrics=['accuracy'])
98
99      model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose
            ↪ =0)
100
101     predictions = model.predict(X_test)
102
103     pred = [round(x[0]) for x in predictions]
104
105
106     acc_count = 0
107     for k in range(0,len(pred)):
108         if pred[k] == Y_test[k]:
109             acc_count += 1
110
111     accuracy = float(acc_count)/len(pred)
112     print 'accuracy is : ' + str(accuracy)
113     ovr_acc += accuracy
114 print 'average accuracy is : ' + str(ovr_acc/iters)
```

8.11 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)

    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):
```

```
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)
```

8.12 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()
```

Chapter 9

Future Work

¡Future work here¿

Chapter 10

Conclusion

¡Conclusion here¿

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