Chapter 1

1. Introduction:

1.1. About the project:

- Pedestrian detection is a key problem in computer vision, with several applications including robotics, surveillance and automotive safety. Much of the progress of the past few years has been driven by the availability of challenging public datasets.

- Detecting people in images is a problem with a long history in the past two years there has been a surge of interest in pedestrian detection.

- Accurate pedestrian detection would have immediate and far reaching impact to applications such as surveillance, robotics, assistive technology for the visually impaired, content based indexing (e.g. Flickr, Google, and movies), advanced human machine interfaces and automotive safety, among others.

- Auto-motive applications are particularly compelling as they have the potential to save numerous lives by detecting the pedestrians.

- The main aim of Pedestrian detection is to detect the visible pedestrians in a video and locating all instances of human beings present in the video, and it has been most widely accomplished by searching all locations in the video, at all possible scales, and comparing a small area at each location with known templates or patterns of people.

Chapter 2

2. System Requirement Specification:

2.1. Software Specification:

- Operating System : Windows 7/8/10
- Coding Language : Python
- Supporting Tools : Python IDLE / Jupyter Notebook

2.2. Hardware Specification:

- Hard disk : 500 GB
- RAM : 4 GB
- Processor : Intel i3 or higher
- Input Devices : Keyboard, mouse

Chapter 3

3. Literature Survey:

3.1. Python:

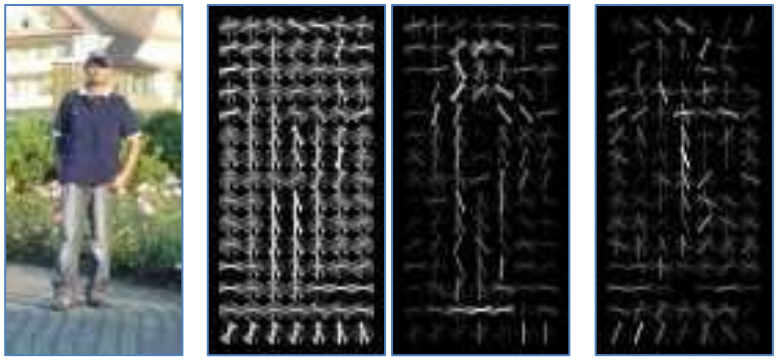
- Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

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. OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

3.3. HOG descriptor :

- The HOG descriptor focuses on the structure or the shape of an object. HOG is able to provide the edge direction as well. This is done by extracting the gradient and orientation of the edges. HOG checks directly surrounding pixels of every single pixel. The goal is to check how darker is the current pixel compared to the surrounding pixels. The algorithm draws and arrows showing the direction of the image getting darker. It repeats the process for each and every pixel in the image. At last, every pixel would be replaced by an arrow, these arrows are called Gradients. These gradients show the flow of light from light to dark. Thus detecting pedestrians in a video.



(a) (b) (c)

Figure 3.1. Our HOG detector focuses especially the head, shoulders and feet. The most active blocks are centered on the image background just outside, (a) A test image. (b) Its computed HOG descriptor. (c) The HOG descriptor weighted by respectively the positive and the negative SVM weights.

3.4. File System:

- A file system controls how data is stored and retrieved. Without a file system, data placed in a storage medium would be one large body of data with no way to tell where one block of data stops and next begins.

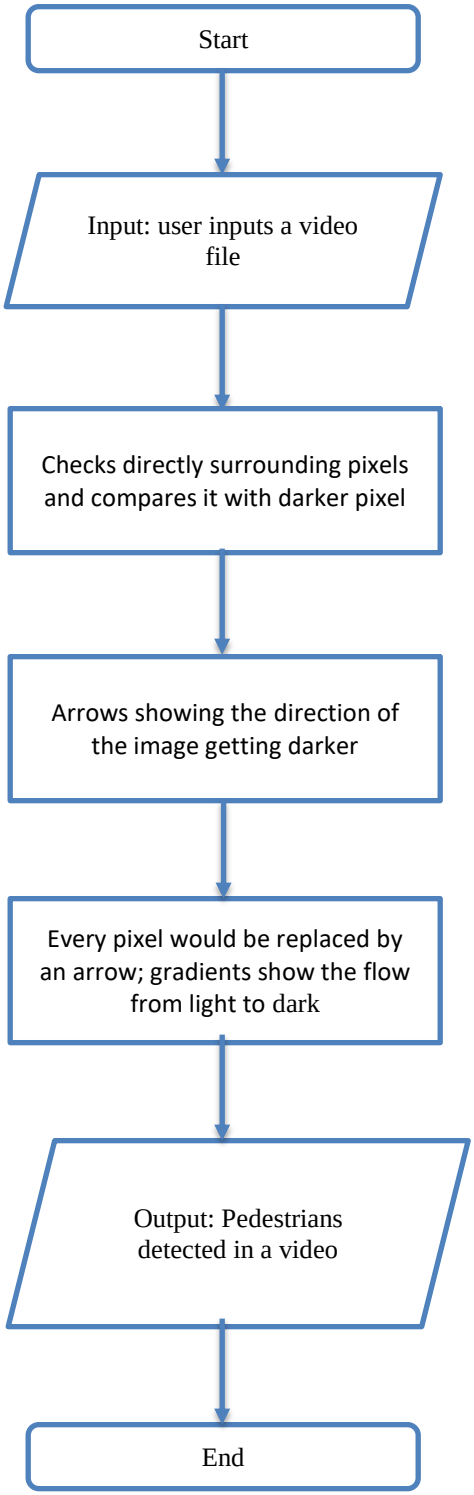
Chapter 4

4. System Design:

4.1.Modules:

- **User Input :**
  - Here user needs to input a video file through the terminal by providing the suitable path of the video file. User can select his/her desired video from their system directories
- **System Process :**
  - OpenCV has a built-in method to detect pedestrians. It has a pre-trained HOG (Histogram of Oriented Gradients) + Linear SVM model to detect pedestrians in images and video streams. It checks the surrounding pixels of every single pixel. The goal is to check how darker is the current pixel compared to the surrounding pixels. The algorithm draws and arrows showing the direction of the image getting darker. It repeats the process for each and every pixel in the image. At last, every pixel would be replaced by an arrow, these arrows are called Gradients. These gradients show the flow of light from light to dark, thus detecting pedestrians.

4.2. Block Diagram:



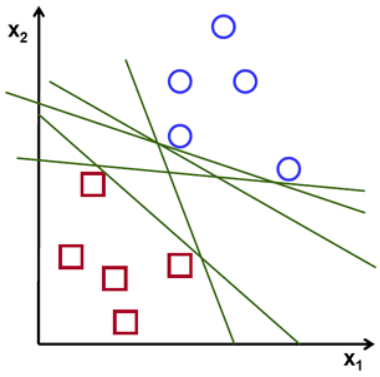
4.3.1 Algorithm:

Support Vector Machine:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyper plane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyper plane which categorizes new examples.

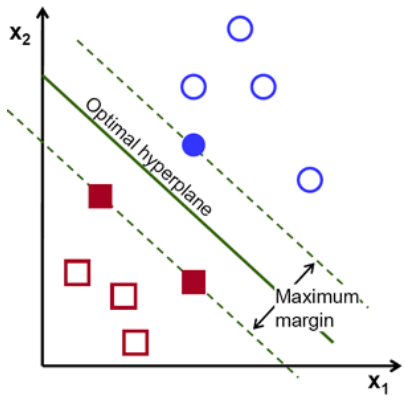
In which sense is the hyper plane obtained optimal? Let's consider the following simple problem:

For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.



In the above picture you can see that there exist multiple lines that offer a solution to the problem. Is any of them better than the others? We can intuitively define a criterion to estimate the worth of the lines: A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, our goal should be to find the line passing as far as possible from all points.

Then, the operation of the SVM algorithm is based on finding the hyper plane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyper plane maximizes the margin of the training data.



Let's introduce the notation used to define formally a hyper plane:

$$f(x)=\beta 0+\beta Tx,$$

where β is known as the *weight vector* and $\beta 0$ as the *bias*.

The optimal hyper plane can be represented in an infinite number of different ways by scaling of β and $\beta 0$. As a matter of convention, among all the possible representations of the hyper plane, the one chosen is

$$|\beta 0+\beta Tx|=1$$

Where x symbolizes the training examples closest to the hyper plane. In general, the training examples that are closest to the hyper plane are called support vectors. This representation is known as the canonical hyper plane.

Now, we use the result of geometry that gives the distance between a point x and a hyper plane $(\beta,\beta 0)$:

$$\text{distance}=|\beta 0+\beta Tx|/||\beta||.$$

In particular, for the canonical hyper plane, the numerator is equal to one and the distance to the support vectors is

$$\text{distance}_{\text{support vectors}} = |\beta_0 + \beta_T x| / ||\beta|| = 1 / ||\beta||.$$

Recall that the margin introduced in the previous section, here denoted as *M*, is twice the distance to the closest examples:

$$M = 2 / ||\beta||$$

Finally, the problem of maximizing *M* is equivalent to the problem of minimizing a function $L(\beta)$ subject to some constraints. The constraints model the requirement for the hyper plane to classify correctly all the training examples *xi*. Formally,

$$\min_{\beta,\beta_0} L(\beta) = 12 ||\beta||_2 \text{ subject to } y_i(\beta_T x_i + \beta_0) \geq 1 \ \forall i,$$

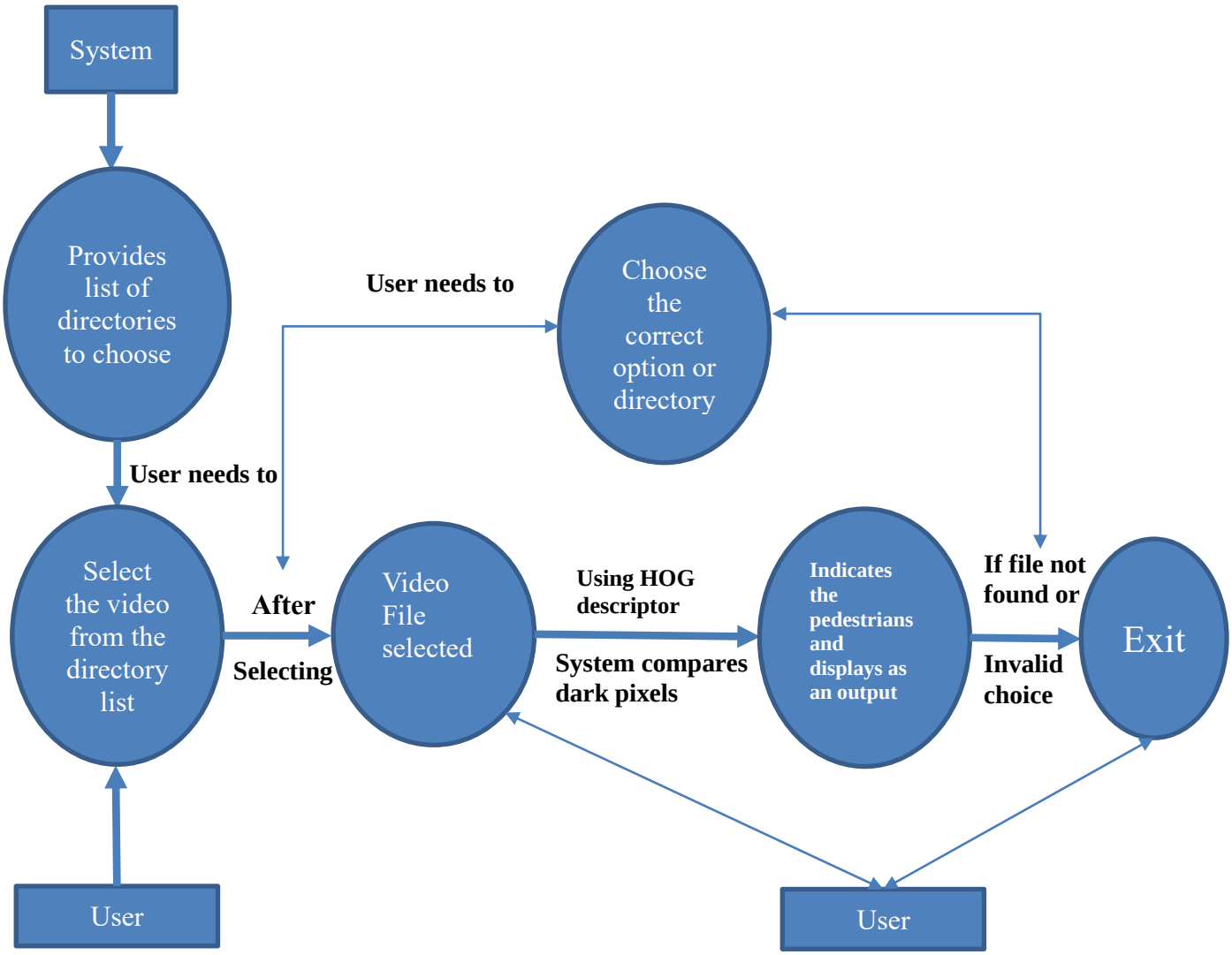
where *yi* represents each of the labels of the training examples.

This is a problem of Lagrangian optimization that can be solved using Lagrange multipliers to obtain the weight vector β and the bias $\beta 0$ of the optimal hyper plane.

Algorithm in Python:

```
svm = cv.ml.SVM_create()
svm.setType(cv.ml.SVM_C_SVC)
svm.setKernel(cv.ml.SVM_LINEAR)
svm.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
```

4.3.2 Data-Flow Diagram:



Chapter 5

5. Implementation and Screenshots:

5.1. Implementation:

```
import cv2
import imutils
import sys
import os

drives = [chr(x) + ':' for x in range(65, 90) if os.path.exists(chr(x) +':')]

def listDirectories():
    listdir = os.listdir(os.getcwd())
    for x in listdir:
        print(x)

while True:
    print("Press 1 to continue")
    result = input("Choose one of the following: ")

    if result == '1':

        print('\nQuick Access:\n1. Documents\n2.
        Videos\n3. Pictures\n4. Downloads\n')

        print('Drives: ')
        for x in range(len(drives)):
            print(str(5 + x) + '. ' + drives[x])

        while True:
            inp = input("\nEnter your Choice: ")

            if inp == '1':
                path = 'C:\\Users\\$USERNAME\\Documents'
                os.chdir(os.path.expandvars(path))
                break
```

```
        elif inp == '2':
            path = 'C:\\Users\\$USERNAME\\Videos'
            os.chdir(os.path.expandvars(path))
            break

        elif inp == '3':
            path = 'C:\\Users\\$USERNAME\\Pictures'
            os.chdir(os.path.expandvars(path))
            break

        elif inp == '4':
            path = 'C:\\Users\\$USERNAME\\Downloads'
            os.chdir(os.path.expandvars(path))
            break

        elif inp in drives:
            os.chdir(inp + '\\')
            break

        else:
            print('Error\nEnter a correct input / drive name.\n')

    while True:

        listDirectories()

        print('\n\nType "exitManager" to exit from file manager.')
        print('Type "backManager" to go up one directory.')
        res = input('\nChoose a file/folder: ')
        print('\n')

        if res in os.listdir(os.getcwd()):
            if os.path.isfile(res):

                hog = cv2.HOGDescriptor()
                hog.setSVMDetector(cv2.HOGDescriptor_getDefault
                PeopleDetector())
```

```
                cap = cv2.VideoCapture(res)

                while cap.isOpened():

                    ret, image = cap.read()

                    if ret:
                    image = imutils.resize(image,
                    width=min(600,image.shape[1]))

                    (regions, _) = hog.detectMultiScale(image,
                     winStride=(4, 4), padding=(4, 4), scale=1.05)

                        for (x, y, w, h) in regions:
                            cv2.rectangle(image, (x, y),
                            (x + w, y + h),(0, 0, 255), 2)

                        cv2.imshow("Pedestrians", image)

                        if cv2.waitKey(1) == 27:
                            break
                    else:
                        break
                cap.release()
                cv2.destroyAllWindows()

            else:
                os.chdir(res)

        elif res == 'exitManager':
            sys.exit(0)

        elif res == 'backManager':
            os.chdir('..')
        else:
            print('No file/folder exist of this name.')
else:
    print("Enter correct number or the input\n")
```

5.2 Screenshots:

```
Press 1 to continue
Choose one of the following: 1

Quick Access:
1. Documents
2. Videos
3. Pictures
4. Downloads

Drives:
5. C:
6. D:
7. E:
8. F:
9. I:

Enter your Choice: E:
$RECYCLE.BIN
car songs
Config.Msi
directors
Languages
MCA 5th sem
Mini Project
Music
PICTURES
SONGS
System Volume Information
Wallpapers
XAMPP


Type "exitManager" to exit from file manager.
Type "backManager" to go up one directory.

Choose a file/folder:
```

Fig1: access the file using file manager

```
Choose a file/folder: Mini Project


.ipynb_checkpoints
1.mp4
2.mp4
3.mp4
4.mp4
5.mp4
6.mp4
7.mp4
haarcascade_frontalface_default.xml
haarcascade_fullbody.xml
Pedestrain Detection Synopsis.docx
Pedestrian Detection final.ipynb
Pedestrian Detection.ipynb


Type "exitManager" to exit from file manager.
Type "backManager" to go up one directory.

Choose a file/folder:
```

Fig2: Selecting the particular video file from the folder

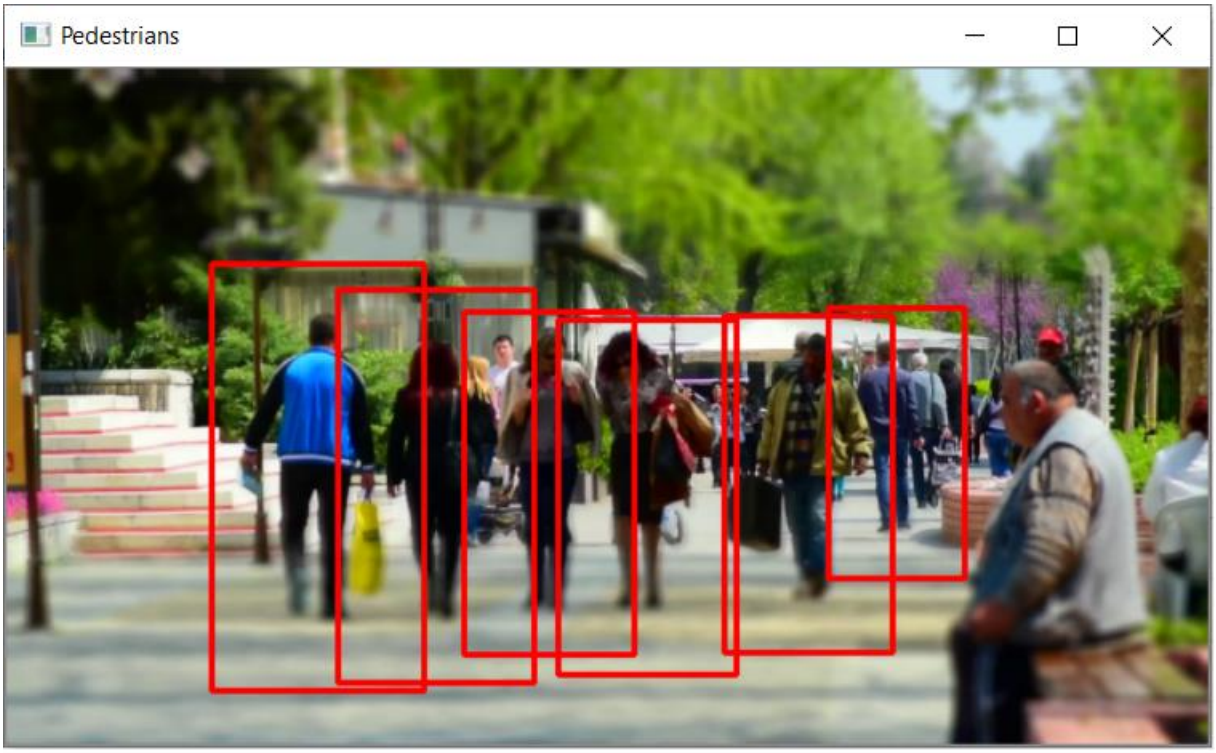


Fig3: Pedestrians detected in a video

Chapter 6

6. Conclusion:

- This project is successful in detecting pedestrians in a given video.
- We gained knowledge on several python modules such as OpenCV, HOG Descriptor which helps us to know more in depth knowledge over python language.
- The main topics are described on chapter four, where the pedestrian detection technique is being specified. This is implemented using python programming language and using a recorded video from a camera or from a surveillance system.

Chapter 7

7. Future Enhancement:

- This project can be implemented for image also.
- In the future it can be developed to provide advance driver assistance.
- Pedestrian Detection system in Traffic Surveillance.
- Increasing performance under more realistic and challenging conditions.
- Exploring two standout cases as being particularly frequent and relevant in the data gathered pedestrians at lower resolution and under partial occlusion.
- In future the entire system compatibility will be improved in such a way that a hardware implementation will be done. A radar sensor will be attached to the bumper of the vehicle for measuring distance between the pedestrian and vehicle.

References:

- https://www.geeksforgeeks.org/opencv-python-tutorial/
- https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/
- https://opencv.org/
- Programming Python: Powerful Object-Oriented Programming by Mark Lutz.
- Learning OpenCV 4 Computer Vision with Python 3 by Joe Minichino and Joseph Howse.