# CHAPTER 1 INTRODUCTION

### INTRODUCTION

In today's world CCTV surveillance is the most basic & impactful security feature a premises can have. It can be found in Hospitals, Malls, Universities etc. being the most famous way of preventing and detecting the unwanted activities. But imagine an academic campus with more than 100 CCTV in multiple buildings like Hostels, Classes, Canteen, Sports area, Auditorium etc. Manual monitoring of all the events on the CCTV camera is impossible. Even if the event had already happened, searching manually the same event in the recorded video wastes a lot of time. With increasing crime rates, it becomes a problem if they are not identified in time and necessary precautionary actions taken. Most urban and metropolitan areas have surveillance systems installed which constantly accumulates data. With the vast accumulation of surveillance data there are higher chances of suspicious activities to occur. But these tasks require human supervision to detect such activities as they are too complicated for artificial intelligence to handle and require high resources. Breaking down complicated tasks and detecting sub tasks which lead to potential crimes are one way to simplify an activity to be automated. We focus on two main potential leads to crimes which we attempt to detect through our models. The most unpredictable one is human behavior and it is very difficult to find whether it is suspicious or normal. Deep learning approach is used to detect suspicious or normal activity in an academic environment, and which sends an alert message to the corresponding authority, in case of predicting a suspicious activity. Monitoring is often performed through consecutive frames which are extracted from the video. The entire framework is divided into two parts. In the first part, the features are computed from video frames and in second part, based on the obtained features classifier predict the class as suspicious or normal.

### LITERATURE REVIEW

Deep [neural networks](https://www.sciencedirect.com/topics/physics-and-astronomy/neural-networks) are now the state-of-the-art machine learning models across a variety of areas, from image analysis to [natural language processing,](https://www.sciencedirect.com/topics/engineering/natural-language-processing) and widely deployed in academia and industry. These developments have a huge potential for medical imaging technology, medical data analysis, medical diagnostics and healthcare in general, slowly being realized. We provide a short overview of recent advances and some associated challenges in machine learning applied to medical image processing and image analysis. Long before deep learning was used, traditional machine learning methods were mainly used. Such as Decision Trees, SVM, Naive Bayes Classifier and Logistic Regression. These algorithms are also called flat algorithms. Flat here means that these algorithms cannot normally be applied directly to the raw data (such as csv, images, text, etc.). We need a pre-processing step called Feature Extraction. The result of Feature Extraction is a representation of the given raw data that can now be used by these classic machine learning algorithms to perform a task. For example, the classification of the data into several categories or classes. Feature Extraction is usually quite complex and requires detailed knowledge of the problem domain. This pre- processing layer must be adapted, tested, and refined over several iterations for optimal results. On the other side are the artificial neural networks of Deep Learning. These do not need the Feature Extraction step. The layers can learn an implicit representation of the raw data directly and on their own. Here, a more and more abstract and compressed representation of the raw data is produced over several layers of artificial neural-nets. This compressed representation of the input data is then used to produce the result. The result can be, for example, the classification of the input data into different classes.

### DIGITAL IMAGE PROCESSING

The identification of objects in an image and this process would probably start with image processing techniques such as noise removal, followed by (low-level) feature extraction to locate lines, regions and possibly areas with certain textures.

The clever bit is to interpret collections of these shapes as single objects, e.g., cars on a road, boxes on a conveyor belt or cancerous cells on a microscope slide. One reason this is an AI problem is that an object can appear very different when viewed from different angles or under different lighting. Another problem is deciding what features belong to what object and which are background or shadows etc. The human visual system performs these tasks mostly unconsciously but a computer requires skillful programming and lots of processing power to approach human performance. Manipulation of data in the form of an image through several possible techniques. An image is usually interpreted as a two- dimensional array of brightness values, and is most familiarly represented by such patterns as those of a photographic print, slide, television screen, or movie screen. An image can be processed optically or digitally with a computer.

### Basics of Image Processing: - IMAGE:

An image is a two-dimensional picture, which has a similar appearance to some subject usually a physical object or a person.

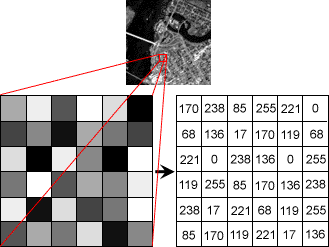
Image is a two-dimensional, such as a photograph, screen display, and as well as a three-dimensional, such as a statue. They may be captured by optical devices—such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces.

The word image is also used in the broader sense of any two-dimensional figure such as a map, a graph, a pie chart, or an abstract painting. In this wider sense, images can also be rendered manually, such as by drawing, painting, carving, rendered automatically by printing or computer graphics technology, or developed by a combination of methods, especially in a pseudo-photograph.



### Fig 1.1: Color image to grey scale Conversion Process

An image is a rectangular grid of pixels. It has a definite height and a definite width counted in pixels. Each pixel is square and has a fixed size on a given display. However different computer monitors may use different sized pixels. The pixels that constitute an image are ordered as a grid (columns and rows); each pixel consists of numbers representing magnitudes of brightness and color.



### Fig 1.2: Gray Scale Image Pixel Value Analysis

Each pixel has a color. The color is a 32-bit integer. The first eight bits determine the redness of the pixel, the next eight bits the greenness, the next eight bits the blueness, and the remaining eight bits the transparency of the pixel.



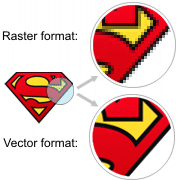
### Fig 1.3: BIT Transferred for Red, Green, and Blue plane (24bit=8bit red;8-bit green;8bit blue)

**IMAGE FILE SIZES:**

Image file size is expressed as the number of bytes that increases with the number of pixels composing an image, and the color depth of the pixels. The greater the number of rows and columns, the greater the image resolution, and the larger the file. Also, each pixel of an image increases in size when its color depth increases, an 8-bit pixel (1 byte) stores 256 colors, a 24-bit pixel (3 bytes) stores 16 million colors, the latter known as true color. Image compression uses algorithms to decrease the size of a file. High resolution cameras produce large image files, ranging from hundreds of kilobytes to megabytes, per the camera's resolution and the image-storage format capacity. High resolution digital cameras record 12-megapixel (1MP = 1,000,000 pixels / 1 million) images, or more, in true color. For example, an image recorded by a 12 MP camera; since each pixel uses 3 bytes to record true color, the uncompressed image would occupy 36,000,000 bytes of memory, a great amount of digital storage for one image, given that cameras must record and store many images to be practical. Faced with large file sizes, both within the camera and a storage disc, image file formats were developed to store such large images.

### IMAGE FILE FORMATS:

Image file formats are standardized means of organizing and storing images. This entry is about digital image formats used to store photographic and other images. Image files are composed of either pixel or vector (geometric) data that are rasterized to pixels when displayed (with few exceptions) in a vector graphic display. Including proprietary types, there are hundreds of image file types. The PNG, JPEG, and GIF formats are most often used to display images on the Internet.



### Fig1.4: Horizontal and Vertical Process

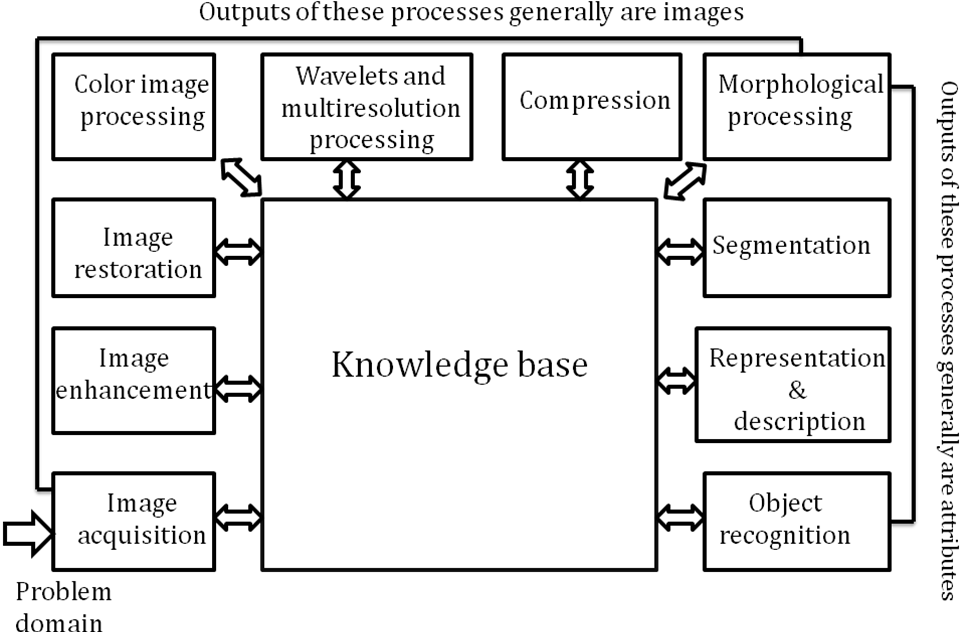
In addition to straight image formats, Metafile formats are portable formats which can include both raster and vector information. The metafile format is an intermediate format. Most Windows applications open metafiles and then save them in their own native format.

### IMAGE PROCESSING:

Digital image processing, the manipulation of images by computer, is relatively recent development in terms of man’s ancient fascination with visual stimuli. In its short history, it has been applied to practically every type of images with varying degree of success. The inherent subjective appeal of pictorial displays attracts perhaps a disproportionate amount of attention from the scientists and from the layman. Digital image processing like other glamour fields, suffers from myths, mis-connect ions, mis-understandings, and mis- information. It is vast umbrella under which fall diverse aspect of optics, electronics, mathematics, photography graphics and computer technology. It is truly multidisciplinary endeavor ploughed with imprecise jargon.

Several factors combine to indicate a lively future for digital image processing. A major factor is the declining cost of computer equipment. Several new technological trends promise to further promote digital image processing. These include parallel processing mode practical by low-cost microprocessors, and the use of charge coupled devices (CCDs) for digitizing, storage during processing and display and large low cost of image storage arrays.

**1.2.2 FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:**



**Fig.1.5:Basic steps of Image Processing**

**Image Acquisition:**

**Image Acquisition** is to acquire a digital image. To do so requires an image sensor and the capability to digitize the signal produced by the sensor. The sensor could be monochrome or color TV camera that produces an entire image of the problem domain every 1/30 sec. the image sensor could also be line scan camera that produces a single image line at a time. In this case, the objects motion past the line.



**Fig 1.6: Digital camera**

Scanner produces a two-dimensional image. If the output of the camera or other imaging sensor is not in digital form, an analog to digital converter digitizes it. The nature of the sensor and the image it produces are determined by the application.



**Fig 1.7: Mobile based Camera**

**Image Enhancement:**

**Image enhancement** is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interesting an image. A familiar example of enhancement is when we increase the contrast of an image because “it looks better.” It is important to keep in mind that enhancement is a very subjective area of image processing.

**Fig 1.8: Image enhancement process for Gray Scale Image and Color Image using Histogram Bits**



**Image restoration:**

**Image restoration** is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.



**Fig 1.9: Noise image🡪 Image Enhancement**

Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a “good” enhancement result. For example, contrast stretching is considered an enhancement technique because it is based primarily on the pleasing aspects it might present to the viewer, whereas removal of image blur by applying a deblurring function is considered a restoration technique.

**Color image processing:**

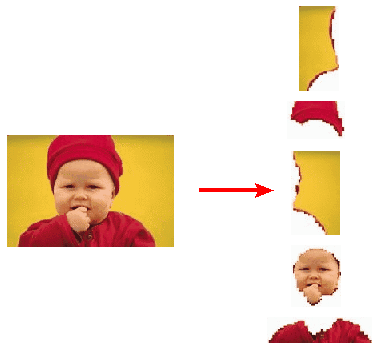
The use of color in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of color shades and intensities, compared to about only two dozen shades of gray. This second factor is particularly important in manual image analysis.



**Fig 1.10: gray Scale image 🡪 Color Image**

**Segmentation:**

**Segmentation** procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.



**Fig 1.11: Image Segment Process**

On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed.

Digital image is defined as a two-dimensional function f(x, y), where x and y are spatial (plane) coordinates, and the amplitude off at any pair of coordinates (x, y) is called intensity or grey level of the image at that point. The field of digital image processing refers to processing digital images by means of a digital computer. The digital image is composed of a finite number of elements, each of which has a particular location and value. The elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term most widely used.

**Image Compression**

Digital Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is removal of redundant data. From the mathematical viewpoint, this amounts to transforming a 2D pixel array into a statically uncorrelated data set. The data redundancy is not an abstract concept but a mathematically quantifiable entity. If n1 and n2 denote the number of information-carrying units in two data sets that represent the same information, the relative data redundancy R [2] of the first data set (the one characterized by n1) can be defined as,

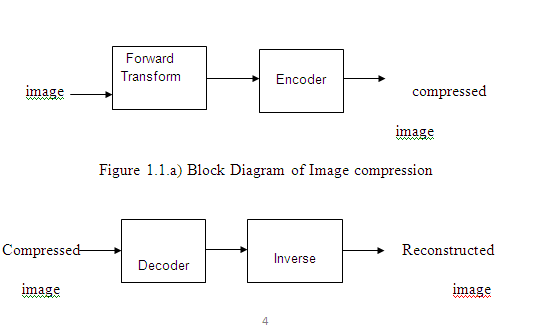
R=1-1/C

Where C called as compression ratio [2]. It is defined as

C=n1/n2

In image compression, three basic data redundancies can be identified and exploited: Coding redundancy, inter-pixel redundancy, and psychosocial redundancy. Image compression is achieved when one or more of these redundancies are reduced or eliminated. The image compression is mainly used for image transmission and storage. Image transmission applications are in broadcast television; remote sensing via satellite, air-craft, radar, or sonar; teleconferencing; computer communications; and facsimile transmission. Image storage is required most commonly for educational and business documents, medical images that arise in computer tomography (CT), magnetic resonance imaging (MRI) and digital radiology, motion pictures, satellite images, weather maps, geological surveys, and so on.

**Image Compression Model**



**Fig 1.12:Decompression Process for Image**

### 1.2.3 CLASSIFICATION OF IMAGES:

There are 3 types of images used in Digital Image Processing. They are

* + - 1. Binary Image
      2. Gray Scale Image
      3. Color Image

### BINARY IMAGE:

A binary image is a [digital image](http://en.wikipedia.org/wiki/Digital_image) that has only two possible values for each [pixel.](http://en.wikipedia.org/wiki/Pixel) Typically, the two colors used for a binary image are black and white though any two colors can be used. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.

Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit (0 or 1). This name black and white, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as [gray-scale images](http://en.wikipedia.org/wiki/Grayscale)

Binary images often arise in [digital image processing](http://en.wikipedia.org/wiki/Digital_image_processing) as [masks](http://en.wikipedia.org/w/index.php?title=Mask_(image_processing)&action=edit&redlink=1) or as the result of certain operations such as [segmentation,](http://en.wikipedia.org/wiki/Segmentation_(image_processing)) [threshold,](http://en.wikipedia.org/wiki/Thresholding_(image_processing)) and [dithering.](http://en.wikipedia.org/wiki/Dither) Some input/output devices, such as [laser printers](http://en.wikipedia.org/wiki/Laser_printer), [fax](http://en.wikipedia.org/wiki/Fax) [machines,](http://en.wikipedia.org/wiki/Fax) and bi-level [computer displays,](http://en.wikipedia.org/wiki/Visual_display_unit) can only handle bi-level images

### GRAY SCALE IMAGE

A gray-scale Image is [digital image](http://en.wikipedia.org/wiki/Digital_image) is an image in which the value of each [pixel](http://en.wikipedia.org/wiki/Pixel) is a single [sample,](http://en.wikipedia.org/wiki/Sample_(signal)) that is, it carries only [intensity](http://en.wikipedia.org/wiki/Luminous_intensity) information. Images of this sort, also known as [black-and-white,](http://en.wikipedia.org/wiki/Black-and-white) are composed exclusively of shades of Gray (0-255), varying from black (0) at the weakest intensity to white (255) at the strongest. Gray-scale images are distinct from one-bit [black-and-white](http://en.wikipedia.org/wiki/Black-and-white) images, which in the context of computer imaging are images with only the two colors, [black,](http://en.wikipedia.org/wiki/Black) and [white](http://en.wikipedia.org/wiki/White) (also called bi-level or [binary](http://en.wikipedia.org/wiki/Binary_image) [images](http://en.wikipedia.org/wiki/Binary_image)). Gray-scale images have many shades of Grey in between. Gray-scale images are also called [monochromatic,](http://en.wikipedia.org/wiki/Monochromatic) denoting the absence of any [chromatic](http://en.wikipedia.org/wiki/Chromaticity) variation.

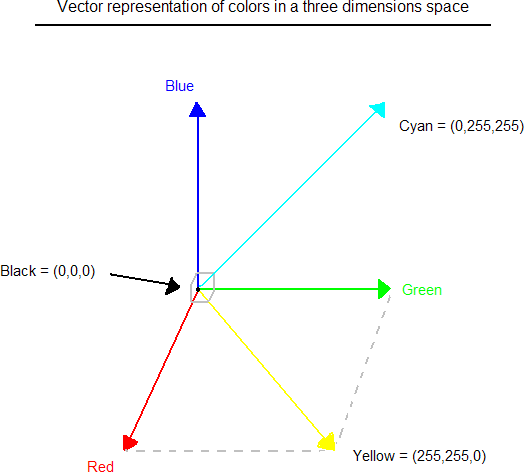
Gray-scale images are often the result of measuring the intensity of light at each pixel in a single band of the [electromagnetic spectrum](http://en.wikipedia.org/wiki/Electromagnetic_spectrum) (e.g. [infrared,](http://en.wikipedia.org/wiki/Infrared) [visible light,](http://en.wikipedia.org/wiki/Visible_spectrum) [ultraviolet,](http://en.wikipedia.org/wiki/Ultraviolet) etc.), and in such cases they are

monochromatic proper when only a given [frequency](http://en.wikipedia.org/wiki/Frequency) is captured. But also, they can be synthesized from a full color image; see the section about converting to gray-scale.

### COLOR IMAGE:

A (digital) color image is a [digital image](http://en.wikipedia.org/wiki/Digital_image) that includes color information for each [pixel.](http://en.wikipedia.org/wiki/Pixel) Each pixel has a particular value which determines its appearing color. This value is qualified by three numbers giving the decomposition of the color in the three primary colors Red, Green, and Blue. Any color visible to human eye can be represented this way. The decomposition of a color in the three primary colors is quantified by a number between 0 and 255. For example, white will be coded as R = 255, G = 255, B = 255; black will be known as (R, G, B) = (0,0,0); and say, bright pink will be: (255,0,255).

In other words, an image is an enormous two-dimensional array of color values, pixels, each of them coded on 3 bytes, representing the three primary colors. This allows the image to contain a total of 256x256x256 = 16.8 million different colors. This technique is also known as RGB encoding, and is specifically adapted to human vision



**Fig 1.13: Hue Saturation Process of RGB SCALE Image**

From the above figure, colors are coded on three bytes representing their decomposition on the three primary colors. It sounds obvious to a mathematician to immediately interpret colors as vectors in a three-dimension space where each axis stands for one of the primary colors. Therefore, we will benefit of most of the geometric mathematical concepts to deal with our colors, such as norms, scalar product, projection, rotation or distance.

# CHAPTER 2

# PROJECT OVERVIEW

## BLOCK DIAGRAM

Feature extraction

pre-processing

INPUT VIDEO

database

NN training

Suspicious activities

CNN algorithm

### Fig 2.1: Block diagram

**Video streaming:** Video streaming technology is one way to deliver video over the Internet. Using streaming technologies, the delivery of audio and video over the Internet can reach many millions of customers using their personal computers, PDAs, mobile smart-phones, or other streaming devices.

**Pre-process**: Image pre-processing are the steps taken to format images before they are used by model training and inference. This includes, but is not limited to, resizing, orienting, and color corrections.

**Train and test:** -Data-sets are split into two subsets. The first subset is known as the training data - it is a portion of our actual data-set that is fed into the machine learning model to discover and learn patterns. In this way, it trains our model. The other subset is known as the testing data.

**Feature analysis: -**Feature analysis argues that we observe individual characteristics, or features, of every object and pattern we encounter. Recognition-by-components theory maintains that we sort objects into them

component parts as a way of recognizing them. These components are understood as three-dimensional shapes called genes. It yields better results than applying machine learning directly to the raw data.

### Convolution Neural Network (CNN algorithm): -

A Convolution Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. When it comes to Machine

Learning, [Artificial Neural Networks](https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/) perform really well. Neural Networks are used in various data-sets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use [Recurrent Neural Networks](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) more precisely

an [LSTM,](https://www.geeksforgeeks.org/understanding-of-lstm-networks/) similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network, there are three types of layers:

* + 1. **Input Layers:** It is the layer in which we give input to our project. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
    2. **Hidden Layer:** The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learn-able weights of that layer and then by the addition of learn-able biases followed by activation function which makes the network nonlinear.
    3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or Soft-Max which converts the output of each class into the probability score of each class.

**2.2 Flow Chart:**

Real World Input Video

Suspicious

Activity

**Fig 2.2:Flow Chart**

Activity

Classification

Contact

Detection

Head

Motion

Detection

Motion

Tracking

Foreground

Extraction

Background

Extraction

Segmenting Video

Into Frames

# Real World Input Video: Begin with a video feed from a surveillance camera or any other source capturing human activity.

# Segmenting Video into Frames: Convert the video into a series of individual frames to process each frame separately.

# Background Extraction: Separate the background from the foreground by extracting the stationary elements of the scene.

# This step helps in identifying objects or people that are in motion against a static background.

# Foreground Extraction: Identify the moving objects or people in each frame by subtracting the background from the current frame.

# The resulting foreground represents the dynamic elements in the scene.

# Motion Tracking: Track the movement of each detected object or person across consecutive frames.

# This step helps in understanding the trajectory and direction of movement.

# Head Motion Detection: Analyze the motion patterns of the detected people's heads.

# By identifying specific head movements (e.g., nodding, shaking), it becomes possible to gather additional information about their actions.

# Contact Detection: Detect interactions or physical contacts between multiple individuals.

# This step helps identify potential confrontations or suspicious behavior based on the detected physical contact.

# Activity Classification: Analyze the overall behavior and movements of the detected individuals.

# Utilize machine learning algorithms or predefined rules to classify activities into normal or suspicious categories.

# Normal Activity: If the activity is classified as normal, no further action is required.

# Suspicious Activity: If the activity is classified as suspicious, generate an alert or notification to relevant authorities or security personnel.

# The flowchart represents a general overview of the steps involved in detecting suspicious human activity. The specific algorithms and techniques used in each step may vary depending on the system's design and the level of complexity required for accurate detection.

# CHAPTER 3 SOFTWARE DISCRIPTION

* + - 1. **Python:**

Python is an object-oriented, high-level language, interpreted, dynamic and multipurpose programming language. Python is easy to learn yet powerful and versatile scripting language which makes it attractive for application development. Python’s syntax and dynamic typing with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas. Python supports multiple programming pattern, including object-oriented programming, imperative and functional programming or procedural styles. python is not intended to work on special area such as web programming. that is why it is known as multipurpose because it can be used with web, enterprise, 3d cad etc. We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to declare an integer value in a variable. python makes the development and debugging fast because there is no compilation step included in python development and edit-test-debug cycle is very fast.

## Python Features

### Easy To Use:

Python Is Easy to Very Easy to Use And High-Level Language. Thus, It Is Programmer-Friendly Language.

### Expressive Language:

Python Language Is More Expressive. The Sense of Expressive Is the Code Is Easily Understandable.

### Interpreted Language:

Python Is an Interpreted Language I.E. Interpreter Executes the Code Line by Line At A Time. This Makes Debugging Easy and Thus Suitable for Beginners.

### Free And Open Source:

Python Language Is Freely Available (www.python. org). The Source-Code Is Also Available. Therefore, It Is Open Source.

### Object-Oriented Language:

Python Supports Object Oriented Language. Concept Of Classes and Objects Comes into Existence.

### Extensible:

It Implies That Other Languages Such As C/C++ Can Be Used to Compile the Code and Thus It Can Be Used Further In Your Python Code.

### Large Standard Library:

Python Has a Large and Broad Library.

### GUI Programming:

Graphical User Interfaces Can Be Developed Using Python.

### Integrated:

It Can Be Easily Integrated with Languages Like C, C++, Java Etc.

## Python History

* Python Laid Its Foundation in The Late 1980s.
* The Implementation of Python was Started in the December 1989 by **Guido Van Ross-um** at Cwi in Nether-land.
* ABC Programming Language is said to be the Predecessor of Python Language which was Capable of Exception Handling and Interfacing with Amoeba Operating System.
* Python is Influenced by Programming Languages like:
  + ABC Language.
  + Module-3

## Python Applications

Python As a Whole Can Be Used in Any Sphere of Development.

Let Us See What Are the Major Regions Where Python Proves to Be Handy.

### Console Based Application

Python Can Be Used to Develop Console Based Applications. For Example: I python.

### Audio Or Video Based Applications

Python Proves Handy in Multimedia Section. Some Of Real Applications Are: Tim-player, Cplay Etc.

### 3d Cad Applications

Fandango Is a Real Application Which Provides Full Features of Cad.

### Web Applications

Python Can Also Be Used to Develop Web Based Application. Some Important Developments Are: Pythonwikiengines, Po-coo, Pythonblogsoftware Etc.

### Enterprise Applications

Python Can Be Used to Create Applications Which Can Be Used Within An Enterprise Or An Organization. Some Real Time Applications Are: Opener p, Try-ton, Pica-lo Etc.

### Applications For Images

Using Python Several Application Can Be Developed for Image. Applications Developed Are: V python, Gogh, Imgseek Etc.

There Are Several Such Applications Which Can Be Developed Using Python.

# 3.5 Python Example:

Python Code Is Simple and Easy To Run. Here Is a Simple Python Code That Will Print "Welcome To Python".

A Simple Python Example Is Given Below.

>>> A="Welcome to Python"

>>> Print A

Welcome To Python

>>>

**Explanation:**

Here We Are Using Idle to Write The Python Code. Detail Explanation to Run Code Is Given In Execute Python Section.

A Variable Is Defined Named "A" Which Holds "Welcome to Python".

"Print" Statement Is Used to Print The Content. Therefore "Print A" Statement Will Print the Content Of The Variable. Therefore, The Output "Welcome to Python" Is Produced.

## Python Example:

In Python 3.4 Version, You Need to Add Parenthesis () In A String Code To Print It.

>>> A=("Welcome to Python Example")

>>> Print A

Welcome To Python Example

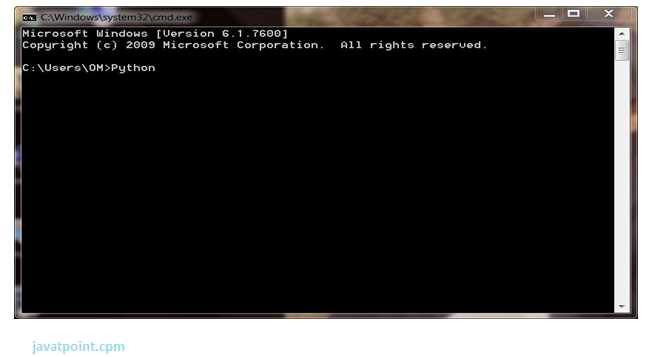
>>>

# How To Execute Python

There Are Three Different Ways of Working In Python:

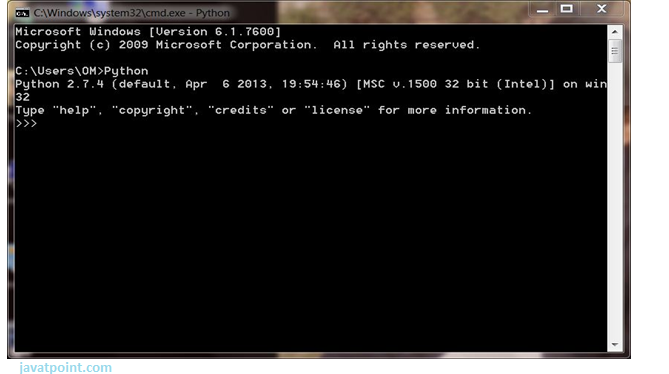
## 1) Interactive Mode:

You Can Enter Python in The Command Prompt And Start Working With Python.



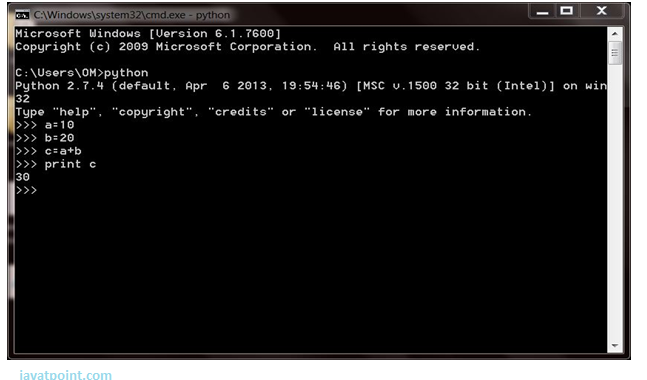
**Fig 3.1: Open python cmd**

Press Enter Key and The Command Prompt Will Appear Like:



**Fig 3.2: Command prompt**

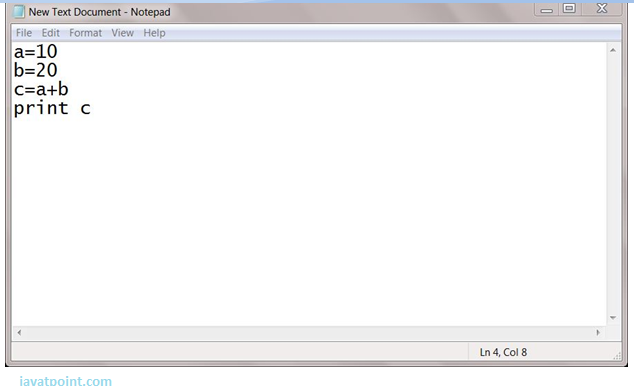
Now You Can Execute Your Python Commands.



## Fig 3.3: Output

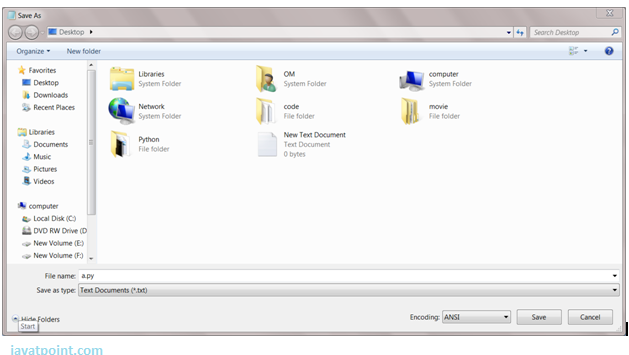
## 2) Script Mode:

Using Script Mode , You Can Write Your Python Code in A Separate File Using Any Editor Of Your Operating System.



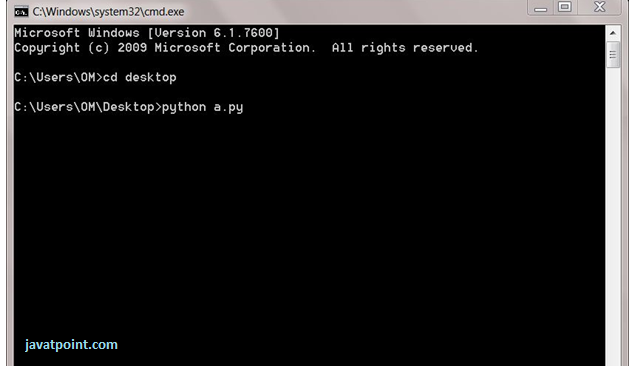
**Fig 3.4: Python code in separate file**

Save It By. Py Extension.



**Fig 3.5: Save it by .py extension**

Now Open Command Prompt and Execute It By:



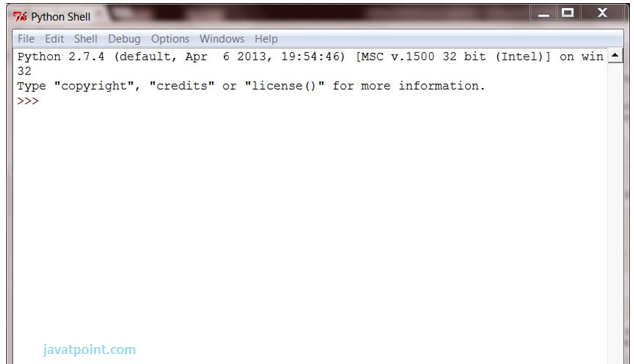
**Fig 3.6: Open cmd and execute it**

## 3) Using Ide: (Integrated Development Environment)

You Can Execute Your Python Code Using a Graphical User Interface (GUI).

All You Need to Do Is:

Click On Start Button -> All Programs -> Python -> Idle (Python GUI)

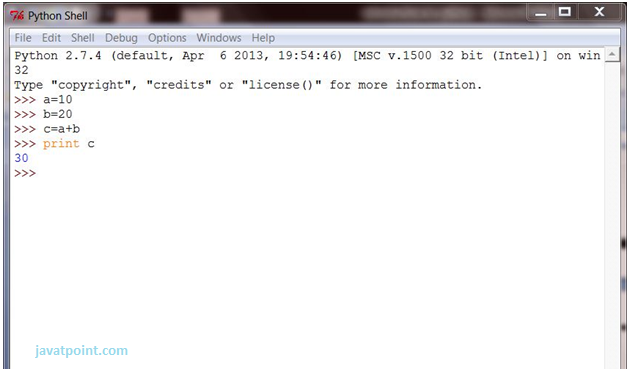


**Fig 3.7 : open python Idle**

You Can Use Both Interactive as Well as Script Mode in Ide.

**1) Using Interactive Mode:**

Execute Your Python Code on The Python Prompt and It Will Display Result Simultaneously.



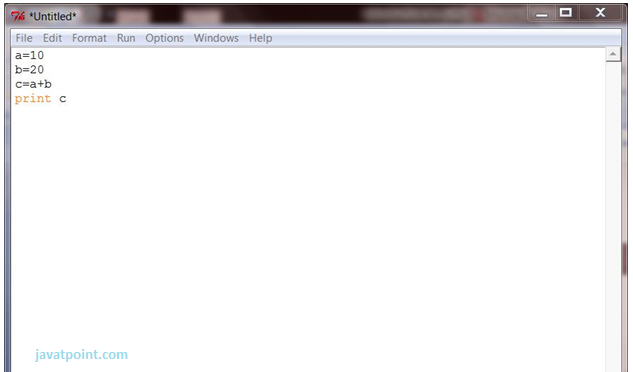
**Fig 3.8: Display Result**

**2) Using Script Mode:**

I) Click On Start Button -> All Programs -> Python -> Idle (Python Gui)

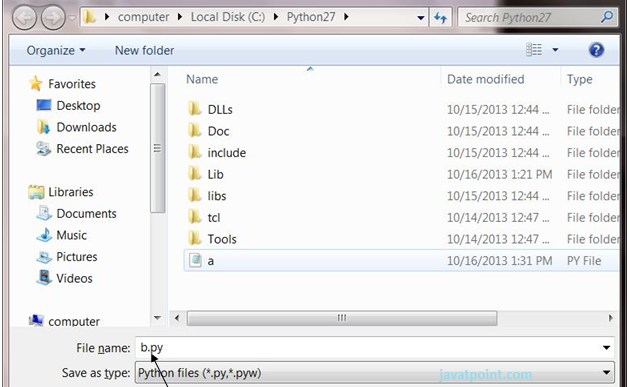
II) Python Shell Will Be Opened. Now Click on File -> New Window.

A New Editor Will Be Opened. Write Your Python Code Here.



**Fig 3.9: Write python code**

Click On File -> Save As

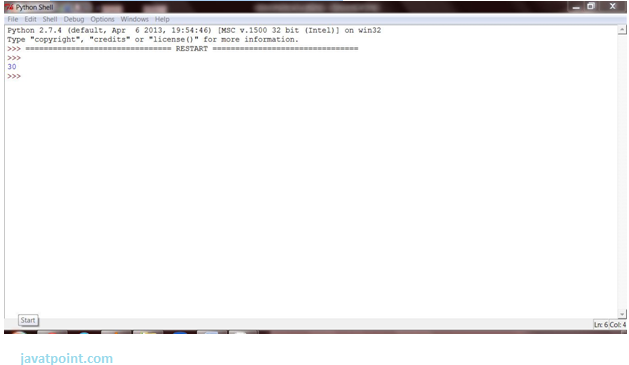


**Fig 3.10: Save the code**

Run Then Code by Clicking on Run in The Menu Bar.

Run -> Run Module

Result Will Be Displayed on A New Python Shell As:



**Fig 3.11: Result is Displayed**

**1) Using Interactive Mode:**

Execute Your Python Code on The Python Prompt and It Will Display Result Simultaneously.

**2) Using Script Mode:**

I) Click On Start Button -> All Programs -> Python -> Idle (Python GUI)

Ii) Python Shell Will Be Opened. Now Click on File -> New Window.

A New Editor Will Be Opened. Write Your Python Code Here.

Run Then Code by Clicking on Run in The Menu Bar.

Run -> Run Module

Result Will Be Displayed on A New Python Shell As:

**OpenCV:**

**Introduction To Computer Vision**

Using Software to Parse the World’s Visual Content Is as Big of a Revolution in Computing as Mobile Was 10 Years Ago, And Will Provide a Major Edge for Developers and Businesses to Build Amazing Products.

Computer Vision Is the Process of Using Machines to Understand and Analyze Imagery (Both Photos and Videos). While These Types Of Algorithms Have Been Around In Various Forms Since The 1960’s, Recent Advances In [Machine Learning](https://blog.algorithmia.com/introduction-to-machine-learning/), As Well As Leaps Forward In Data Storage, Computing Capabilities, And Cheap High-Quality Input Devices, Have Driven Major Improvements In How Well Our Software Can Explore This Kind Of Content.

**What Is Computer Vision?**

Computer Vision Is the Broad Parent Name for Any Computations Involving Visual Content – That Means Images, Videos, Icons, And Anything Else with Pixels Involved. But Within This Parent Idea, There Are A Few Specific Tasks That Are Core Building Blocks:

In **Object Classification**, You Train a Model on A Data-set Of Specific Objects, And the Model Classifies New Objects as Belonging to One Or More of Your Training Categories.

For **Object Identification**, Your Model Will Recognize a Specific Instance of An Object – For Example, Parsing Two Faces in An Image and Tagging One as Tom Cruise and One as Katie Holmes.

A Classical Application of Computer Vision Is Handwriting Recognition for Digitizing Handwritten Content (We’ll Explore More Use Cases Below). Outside Of Just Recognition, Other Methods of Analysis Include:

**Video** **Motion Analysis** Uses Computer Vision to Estimate the Velocity of Objects in A Video, Or the Camera Itself.

In **Image Segmentation**, Algorithms Partition Images into Multiple Sets of Views.

**Scene Reconstruction** Creates A 3d Model of a Scene Inputted Through Images or Video (Check Out [Selva](https://www.selva3d.com/)).

In **Image Restoration**, Noise Such as Blurring Is Removed from Photos Using Machine Learning Based Filters.

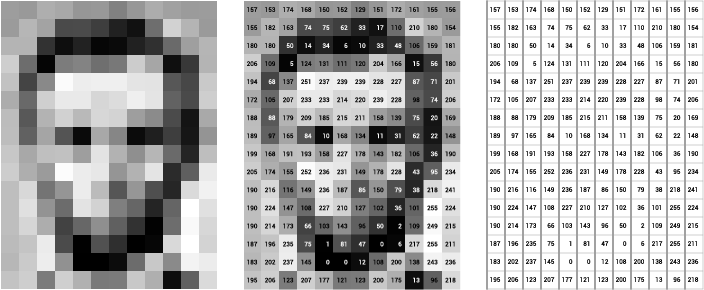
Any Other Application That Involves Understanding Pixels Through Software Can Safely Be Labeled as Computer Vision.

**How Computer Vision Works**

One Of the Major Open Questions in Both Neuroscience and Machine Learning Is: How Exactly Do Our Brains Work, And How Can We Approximate That with Our Own Algorithms? The Reality Is That There Are Very Few Working and Comprehensive Theories of Brain Computation; So, Despite the Fact That Neural Nets Are Supposed To “Mimic the Way the Brain Works,” Nobody Is Quite Sure If That’s Actually True. Jeff Hawkins Has An [Entire Book on This Topic Called on Intelligence](https://www.amazon.com/Intelligence-Understanding-Creation-Intelligent-Machines/dp/0805078533).

The Same Paradox Holds True for Computer Vision – Since We’re Not Decided on How the Brain and Eyes Process Images, It’s Difficult to Say How Well the Algorithms Used in Production Approximate Our Own Internal Mental Processes. For Example, [Studies Have Shown](https://www.technologyreview.com/s/508376/in-a-frogs-eye/) That Some Functions That We Thought Happen in The Brain of Frogs Actually Take Place in The Eyes. We’re A Far Cry from Amphibians, But Similar Uncertainty Exists in Human Cognition.

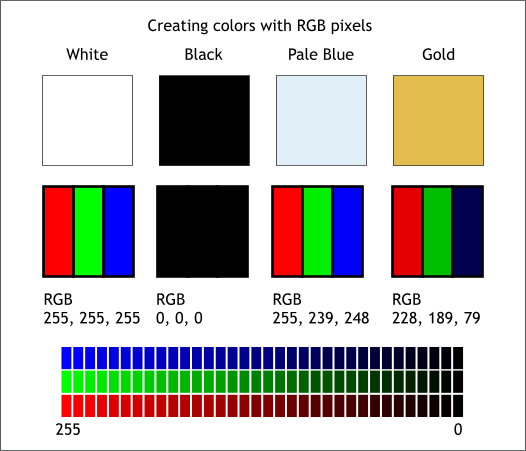
Machines Interpret Images Very Simply: As A Series of Pixels, Each with Their Own Set of Color Values. Consider The Simplified Image Below, And How Gray-scale Values Are Converted into A Simple Array of Numbers:



**Fig 3.12: Source:** [**Open-frameworks**](http://openframeworks.cc/ofBook/chapters/image_processing_computer_vision.html)

Think Of an Image as A Giant Grid of Different Squares, Or Pixels (This Image Is a Very Simplified Version of What Looks Like Either Abraham Lincoln or A Dementor). Each Pixel in An Image Can Be Represented by A Number, Usually From 0 – 255. The Series of Numbers on The Right Is What Software Sees When You Input an Image. For Our Image, There Are 12 Columns And 16 Rows, Which Means There Are 192 Input Values for This Image.

When We Start to Add in Color, Things Get More Complicated. Computers Usually Read Color as A Series Of 3 Values – Red, Green, And Blue (Rgb) – On That Same 0 – 255 Scale. Now, Each Pixel Actually Has 3 Values for The Computer to Store in Addition to Its Position. If We Were to Colorize President Lincoln (Or Harry Potter’s Worst Fear), That Would Lead To 12 X 16 X 3 Values, Or 576 Numbers.



**Fig 3.13: Source:** [**Xaraxone**](http://archive.xaraxone.com/webxealot/workbook35/page_5.htm)

For Some Perspective on How Computationally Expensive This Is, Consider This Tree:

Each Color Value Is Stored In 8 Bits.

8 Bits X 3 Colors Per Pixel = 24 Bits Per Pixel.

A Normal Sized 1024 X 768 Image X 24 Bits Per Pixel = Almost 19m Bits, Or About 2.36 Megabytes.

That’s A Lot of Memory to Require for One Image, And A Lot of Pixels for An Algorithm to Iterate Over. But To Train A Model with Meaningful Accuracy – Especially When You’re Talking About [Deep Learning](https://blog.algorithmia.com/introduction-to-deep-learning/) – You’d Usually Need Tens of Thousands of Images, And the More the Merrier. Even If You Were to Use [Transfer Learning](https://en.wikipedia.org/wiki/Transfer_learning) To Use the Insights of An Already Trained Model, You’d Still Need A Few Thousand Images to Train Yours On.

With The Sheer Amount of Computing Power and Storage Required Just to Train Deep Learning Models for Computer Vision, It’s Not Hard to Understand Why Advances in Those Two Fields Have Driven Machine Learning Forward to Such a degree.

**Business Use Cases for Computer Vision**

Computer Vision Is One of The Areas in Machine Learning Where Core Concepts Are Already Being Integrated into Major Products That We Use Every Day. [Google Is Using Maps](https://research.googleblog.com/2017/05/updating-google-maps-with-deep-learning.html) To Leverage Their Image Data and Identify Street Names, Businesses, And Office Buildings. Face book Is Using Computer Vision to Identify People in Photos, And Do several Things with That Information.

But It’s Not Just Tech Companies That Are Leverage Machine Learning for Image Applications. Ford, The American Car Manufacturer That Has Been Around [Literally Since The Early 1900’s](https://en.wikipedia.org/wiki/Ford_Motor_Company), Is [Investing Heavily In Autonomous Vehicles (Avs)](https://media.ford.com/content/fordmedia/fna/us/en/news/2016/08/16/ford-targets-fully-autonomous-vehicle-for-ride-sharing-in-2021.html). Much Of the Underlying Technology in Avs Relies on Analyzing the Multiple Video Feeds Coming into The Car and Using Computer Vision to Analyze and Pick a Path of Action.

Another Major Area Where Computer Vision Can Help Is in The Medical Field. Much Of Diagnosis Is Image Processing, Like Reading X-Rays, MRI Scans, And Other Types of Diagnostics. [Google Has Been Working with Medical Research Teams](https://research.google.com/teams/brain/healthcare/) To Explore How Deep Learning Can Help Medical Work-flows, And Have Made Significant Progress in Terms of Accuracy. To Paraphrase from Their Research Page:

“Collaborating Closely with Doctors and International Healthcare Systems, We Developed a State-Of-The-Art Computer Vision System for Reading Retinal Fundus Images for Diabetic Retinopathy and Determined Our Algorithm’s Performance Is on Par with U.S. Board-Certified Ophthalmologists. We’ve Recently Published Some Of Our Research In The [Journal Of The American Medical Association](https://research.google.com/pubs/archive/45732.pdf) And Summarized The Highlights In A [Blog Post](https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html).”

But Aside from The Groundbreaking Stuff, It’s Getting Much Easier to Integrate Computer Vision into Your Own Applications. A Number of High-Quality Third-Party Providers Like Clarifai Offer [A Simple Api for Tagging And Understanding Images](https://www.clarifai.com/), While Kairos [Provides Functionality Around Facial Recognition](https://www.kairos.com/). We’ll Dive into The Open-Source Packages Available for Use Below.

**Computer Vision on Algorithmic**

Algorithmic Makes It Easy to Deploy Computer Vision Applications as Scalable Micro-services. Our Marketplace Has A Few Algorithms to Help Get the Job Done:

* [Sal-net](https://algorithmia.com/algorithms/deeplearning/SalNet) Automatically Identifies the Most Important Parts of An Image
* [Nudity Detection](https://algorithmia.com/algorithms/sfw/NudityDetectioni2v) Detects Nudity in Pictures
* [Emotion Recognition](https://algorithmia.com/algorithms/deeplearning/EmotionRecognitionCNNMBP) Parses Emotions Exhibited in Images
* [Deep-style](https://demos.algorithmia.com/deep-style/) Transfers Next-Level Filters onto Your Image
* [Face Recognition](https://algorithmia.com/algorithms/cv/FaceRecognition)…Recognizes Faces.
* [Image Memorability](https://algorithmia.com/algorithms/deeplearning/LargescaleImageMemorability) Judges How Memorable an Image Is.

A Typical Work-flow For Your Product Might Involve Passing Images from A Security Camera into Emotion Recognition and Raising a Flag If Any Aggressive Emotions Are Exhibited, Or Using Nudity Detection to Block Inappropriate Profile Pictures on Your Web Application.

For A More Detailed Exploration of How You Can Use the Algorithmic Platform to Implement Complex and Useful Computer Vision Tasks,

### Computer Vision Resources

##### Packages And Frameworks

[**OpenCV**](https://opencv.org/)– “OpenCV Was Designed For Computational Efficiency And With A Strong Focus On Real-Time Applications. Adopted All Around The World, OpenCV Has More Than 47 Thousand People Of User Community And Estimated Number Of Downloads Exceeding 14 Million. Usage Ranges From Interactive Art, To Mines Inspection, Stitching Maps On The Web Or Through Advanced Robotics.”

[**Simple CV**](http://simplecv.org/) – “Simple CV Is An Open-Source Framework For Building Computer Vision Applications. With It, You Get Access To Several High-Powered Computer Vision Libraries Such As Open CV – Without Having To First Learn About Bit Depths, File Formats, Color Spaces, Buffer Management, Eigenvalues, Or Matrix Versus Bitmap Storage.”

[**Mahotas**](http://mahotas.readthedocs.io/en/latest/) **– “**Mahotas Is A Computer Vision And Image Processing Library For Python. It Includes Many Algorithms Implemented In C++ For Speed While Operating In Numpy Arrays And With A Very Clean Python Interface. Mahotas Currently Has Over 100 Functions For Image Processing And Computer Vision And It Keeps Growing.

**NumPy:**

Numpy, Which Stands For Numerical Python, Is A Library Consisting Of Multidimensional Array Objects And A Collection Of Routines For Processing Those Arrays. Using Numpy, Mathematical And Logical Operations On Arrays Can Be Performed. This Tutorial Explains The Basics Of Numpy Such As Its Architecture And Environment. It Also Discusses The Various Array Functions, Types Of Indexing, Etc. An Introduction To Matplotlib Is Also Provided. All This Is Explained With The Help Of Examples For Better Understanding.

**Audience**

This Tutorial Has Been Prepared For Those Who Want To Learn About The Basics And Various Functions Of Numpy. It Is Specifically Useful For Algorithm Developers. After Completing This Tutorial, You Will Find Yourself At A Moderate Level Of Expertise From Where You Can Take Yourself To Higher Levels Of Expertise.

**Prerequisites**

You Should Have A Basic Understanding Of Computer Programming Terminologies. A Basic Understanding Of Python And Any Of The Programming Languages Is A Plus.

Numpy Is A Python Package. It Stands For 'Numerical Python'. It Is A Library Consisting Of Multidimensional Array Objects And A Collection Of Routines For Processing Of Array.

**Numeric**, The Ancestor of NumPy, Was Developed By Jim Hugunin. Another Package Numarray Was Also Developed, Having Some Additional Functionalities. In 2005, Travis Oliphant Created NumPy Package By Incorporating The Features Of Numarray Into Numeric Package. There Are Many Contributors to This Open-Source Project.

## Operations Using NumPy

Using NumPy, A Developer Can Perform the Following Operations −

Mathematical And Logical Operations on Arrays.

Fourier Transforms and Routines for Shape Manipulation.

Operations Related to Linear Algebra. NumPy Has In-Built Functions for Linear Algebra And Random Number Generation.

## 

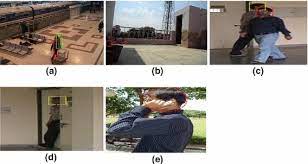
# CHAPTER 4 RESULT

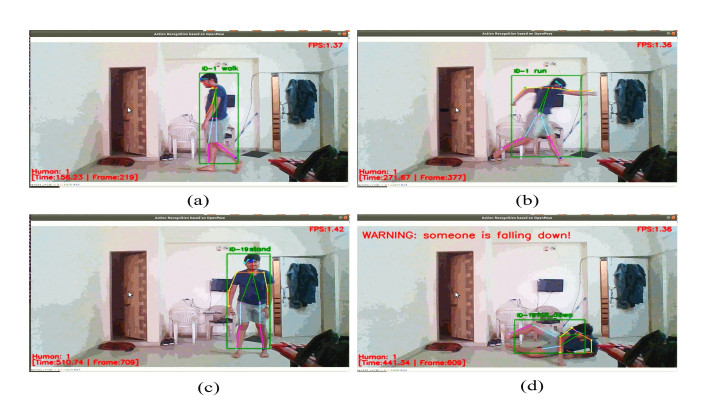
Detecting suspicious activity in video data is a challenging task. There are various difficulties, such as the complexity of the scene, the illumination of the light, the angle of the camera, and so on. In addition, suspect activities are determined by the scene and place in which they occur.

for e.g. A bag left in a classroom for more than half an hour is normal, but a bag left at a railway station for half an hour is suspicious. The standard and challenging data sets are not easily available f or testing, which is another problem. To test the proposed framework, we used standard public data sets.

In video surveillance models, computation time is a crucial factor. To calculate computation time, we divide the time that has elapsed since inference for the entire video by the number of frames rendered from the data-set.







**Fig 4.1: Result**

# CHAPTER 5 ADVANTAGES AND APPLICATIONS

* 1. **Advantages: -**

1. Less Time Consumption

2. High Accuracy

1. Cost Efficient
2. Automatic alert if any activity is detected
3. It will identify the suspect with videos

# APPLICATIONS: -

1. Airports, Bus stands and auto stands
2. Public places
3. Banks
4. Parking Places 5.Abandoned places

# CHAPTER 6 CONCLUSION AND FUTURE SCOPE

## CONCLUSION

As per the methodology discussed in this project, we analysis surveillance footage to detect suspicious human activity by considering three specific cases which include (a) human fall , (b) starting of fire and (c) gun/rifle shooting . We have created input database containing videos representing each of the three cases. To detect suspicious activity in surveillance footage we perform feature detection, CNN data set training and detect the suspicious activity using CNN algorithm. This is particularly important given the mitigation of risk to human lives if the models were integrated into existing surveillance systems. This technique can be used for any surveillance and security implementation in real time.

## FUTURE SCOPE

In our project we take videos as input and if video is large, then it will take more time to create individual frames in the output video. In future we can try to improve accuracy and make sure that it takes less time to capture human suspicious activity. In future we can add more video data set to detect suspicious human activities using deep learning.

**REFERENCES**

1. S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards RealTime Object Detection with Region Proposal Networks,” in NIPS 2015 - Advances in Neural Information Processing Systems 28. Neural Information Processing Systems Foundation, 2015
2. Pierre Sermanet David Eigen Xiang Zhang Michael Mathieu Rob Fergus Yann LeCun “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks” Courant Institute of Mathematical Sciences, New York University. Available: <https://arxiv.org/pdf/1312.6229.pdf>.
3. The IMFDB Internet Movie Firearms Database, [Online] Available: <http://www.imfdb.org/wiki/Main_Page>[Accessed Mar 20, 2019]
4. Soft Computing and Intelligence Information Systems, [Online] Available: https://sci2s.ugr.es/weapons- detection [Accessed Mar 27, 2019]
5. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in arXiv, vol. abs/1512.00567, 2015
6. Rohit Kumar Tiwari and Gyanendra K. Verma, “A Computer Vision based Framework for Visual Gun Detection using Harris Interest Point Detector”, Procedia Computer Science, vol 54, p. 703 - 712, 2015.
7. Samir K. Bandyopadhyay, Biswajita Datta, and Sudipta Roy Identifications of concealed weapon in a Human Body Department of Computer Science and Engineer, University of Calcutta, 2012
8. Claire-Hlne Demarty, et. al The MediaEval 2012 affect task: violent scenes detection Working Notes Proceedings of the MediaEval 2012 Workshop. 2012.
9. Roberto Olmos, Siham Tabik, and Francisco Herrera Automatic Handgun Detection Alarm in Videos Using Deep Learning Soft Computing and Intelligent Information Systems research group, Department of Computer Science and Artificial Intelligence, University of Granada, 2017
10. Lisa Torrey and Jude Shavlik, “Transfer Learning”, University of Wisconsin, Madison WI, USA, Available: <http://ftp.cs.wisc.edu/machine-learning/shavlikgroup/torrey.handbook09.pdf> [Accessed Mar 15, 2019]

## APPENDIX

### SOFTWARE CODE:

import tkinter as tk

from tkinter import filedialog import numpy as np

from keras.preprocessing import image from keras.models import Sequential from keras.layers import Dense

from keras.models import model\_from\_json import os

import imutils import time

import pose\_estimation\_class as pm import mediapipe as mp

import cv2

from tkinter import \*

from PIL import ImageTk, Image

import \_tkinter # with underscore, and lowercase 't'

cap=cv2.VideoCapture(0)

json\_file = open('model\_pose.json', 'r') loaded\_model\_json = json\_file.read() json\_file.close()

loaded\_model = model\_from\_json(loaded\_model\_json) # load weights into new model

loaded\_model.load\_weights("model\_pose.h5") print("Loaded model from disk")

label=['hand\_wave','squat','standing','taking\_phone','walking','yoga'] fgbg = cv2.createBackgroundSubtractorMOG2()

j = 0

detector = pm.PoseDetector() while 1:

nl,frame=cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) fgmask = fgbg.apply(gray)

contours,hierchy = cv2.findContours(fgmask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) if contours:

areas = []

for contour in contours:

ar = cv2.contourArea(contour) areas.append(ar)

max\_area = max(areas or [0])

max\_area\_index = areas.index(max\_area)

cnt = contours[max\_area\_index]

M = cv2.moments(cnt)

x, y, w, h = cv2.boundingRect(cnt)

cv2.drawContours(fgmask, [cnt], 0, (255,255,255), 3, maxLevel = 0)

if h < w: j += 1

if j > 10:

#print ("FALL")

cv2.putText(frame, 'FALL', (x, y), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (255,255,255), 2)

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)

if h > w: j = 0

cv2.putText(frame, 'normal human', (x, y), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (255,255,255), 2)

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

#cv2.imshow('fall Frame', frame)

frame, p\_landmarks, p\_connections = detector.findPose(frame, False) mp.solutions.drawing\_utils.draw\_landmarks(frame, p\_landmarks, p\_connections) lmList = detector.getPosition(frame)

cv2.imshow('pose frame',frame)

#frame = cv2.resize(frame, (256, 256) k=cv2.waitKey(1)

if k%256==27:

print('close') break

elif k%256==32: print("image saved")

cv2.imwrite('input.jpg',frame)

#test\_image = cv2.resize(frame, (256, 256)

test\_image = image.load\_img('input.jpg', target\_size = (128, 128)) test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0) result = loaded\_model.predict(test\_image) fresult=np.max(result) label2=label[result.argmax()]

print(label2) win=tk.Tk()

def b1\_click(): global path2 try:

json\_file = open('model1.json', 'r') loaded\_model\_json = json\_file.read() json\_file.close()

loaded\_model = model\_from\_json(loaded\_model\_json) # load weights into new model loaded\_model.load\_weights("model1.h5") print("Loaded model from disk")

label=["Apple Apple\_scab","Apple\_Black\_rot","Apple\_Cedar\_apple\_rust","Apple Healthy", "Corn\_(maize) Cercospora\_leaf\_spot Gray\_leaf\_spot","Corn(maize) Common\_rust", "Corn\_(maize) Healthy","Corn(maize) Northern\_Leaf\_Blight","Grape Black\_rot",

"Grape Esca(Black\_Measles)","Grape Healthy","Grape\_Leaf\_blight(Isariopsis\_Leaf\_Spot)", "Potato Early\_blight","Potato\_Healthy","Potato\_Late\_blight","Tomato Bacterial\_spot", "Tomato Early\_blight","Tomato\_Healthy","Tomato\_Late\_blight","Tomato Leaf\_Mold",

"Tomato Septoria\_leaf\_spot","Tomato\_Spider\_mites Two- spotted\_spider\_mite","Tomato Target\_Spot",

"Tomato Tomato\_Yellow\_Leaf\_Curl\_Virus","Tomato Tomato\_mosaic\_virus"] #lbl2=tk.Label(win,image=img)

#lbl2.pack(side = "bottom", fill = "both", expand = "yes") #img1=('F:/py/leaf\_disease\_final( COMPLETE )/1.jpg') #lbl2=tk.Label(win,image=img1)

#lbl2.pack(side = "bottom", fill = "both", expand = "yes") #loading image

path2=filedialog.askopenfilename() print(path2)

#img = ImageTk.PhotoImage(Image.open(path2))

#lbl2=tk.Label(win,image=img)

#lbl2.pack(side = "bottom", fill = "both", expand = "yes")

#The Label widget is a standard Tkinter widget used to display a text or image on the screen. #panel = tk.Label(win, image = img)

#panel.pack( fill = "both", expand = "yes") #imr=cv2.imread(path2) #a=cv2.imshow(imr)

#print(imr)

test\_image = image.load\_img(path2, target\_size = (128, 128)) test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0) result = loaded\_model.predict(test\_image) #print(result)

#print(result) fresult=np.max(result) label2=label[result.argmax()] print(label2) #lb2.configure(image=img) #lbl2.image=img lbl.configure(text=label2) #lbl2(ent.config(state='disabled')) win.mainloop()

except IOError: pass

#button

#labelframe = LabelFrame(win, text="Leaf Disease Detection using OPENCV") #labelframe.pack(fill="both", expand="yes")

label1 = Label(win, text="GUI For Leaf Disease Detection using OPENCV", fg ='blue')

label1.pack()

b1=tk.Button(win, text= 'browse image',width=25, height=3,fg ='red', command=b1\_click) b1.pack()

lbl = Label(win, text="Result", fg ='blue') lbl.pack()

#image =ImageTk.PhotoImage(file='a.JPG')

#img1='1.JPG'

#lb2 = Label(win,image=image) #lb2.pack()

#lbl.grid(column=0, row=0) win.geometry("550x250")

win.title("Leaf Disease Detection using OPENCV") win.bind("<Return>", b1\_click)

win.mainloop()

**Software code 2:**

# import the necessary packages

import numpy as np

import argparse

import imutils

import time

import cv2

import os

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

##ap.add\_argument("-i", "--input", required=True,

## help="path to input video")

ap.add\_argument("-c", "--confidence", type=float, default=0.5,

help="minimum probability to filter weak detections")

ap.add\_argument("-t", "--threshold", type=float, default=0.3,

help="threshold when applyong non-maxima suppression")

args = vars(ap.parse\_args())

# load the COCO class labels our YOLO model was trained on

labelsPath = os.path.sep.join(["obj.names"])

LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label

np.random.seed(42)

COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),

dtype="uint8")

# derive the paths to the YOLO weights and model configuration

weightsPath = os.path.sep.join(["yolov3.weights"])

configPath = os.path.sep.join(["yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)

# and determine only the \*output\* layer names that we need from YOLO

print("[INFO] loading YOLO from disk...")

net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

ln = net.getLayerNames()

ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

# initialize the video stream, pointer to output video file, and

# frame dimensions

fgbg = cv2.createBackgroundSubtractorMOG2()

#vs= cv2.VideoCapture("fire1.mp4")

vs= cv2.VideoCapture(0)

count=0

j=0

writer = None

(W, H) = (None, None)

# try to determine the total number of frames in the video file

try:

prop = cv2.cv.CV\_CAP\_PROP\_FRAME\_COUNT if imutils.is\_cv2() \

else cv2.CAP\_PROP\_FRAME\_COUNT

total = int(vs.get(prop))

print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total

# number of frames in the video file

except:

print("[INFO] could not determine # of frames in video")

print("[INFO] no approx. completion time can be provided")

total = -1

# loop over frames from the video file stream

while True:

# read the next frame from the file

(grabbed, frame) = vs.read()

img=frame.copy()

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

fgmask = fgbg.apply(gray)

#cv2.imshow('gray video',gray)

#human detection

#Find contours

contours,hierchy = cv2.findContours(fgmask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

areas = []

for contour in contours:

ar = cv2.contourArea(contour)

areas.append(ar)

max\_area = max(areas or [0])

max\_area\_index = areas.index(max\_area)

cnt = contours[max\_area\_index]

M = cv2.moments(cnt)

x, y, w, h = cv2.boundingRect(cnt)

if h < w:

#if h == h/2:

j += 1

if j > 5:

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)

cv2.putText(frame, ' Suspicious activity found : Falling down', (x, y), cv2.FONT\_HERSHEY\_TRIPLEX, 0.5, (0), 2)

print ("FALL")

cv2.drawContours(fgmask, [cnt], 0, (0,0,255), 3, maxLevel = 0)

if h > w:

j = 0

# if the frame was not grabbed, then we have reached the end

# of the stream

if not grabbed:

break

# if the frame dimensions are empty, grab them

if W is None or H is None:

(H, W) = frame.shape[:2]

# construct a blob from the input frame and then perform a forward

# pass of the YOLO object detector, giving us our bounding boxes

# and associated probabilities

blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),

swapRB=True, crop=False)

net.setInput(blob)

start = time.time()

layerOutputs = net.forward(ln)

end = time.time()

# initialize our lists of detected bounding boxes, confidences,

# and class IDs, respectively

boxes = []

confidences = []

classIDs = []

# loop over each of the layer outputs

for output in layerOutputs:

# loop over each of the detections

for detection in output:

# extract the class ID and confidence (i.e., probability)

# of the current object detection

scores = detection[5:]

classID = np.argmax(scores)

confidence = scores[classID]

# filter out weak predictions by ensuring the detected

# probability is greater than the minimum probability

if confidence > args["confidence"]:

# scale the bounding box coordinates back relative to

# the size of the image, keeping in mind that YOLO

# actually returns the center (x, y)-coordinates of

# the bounding box followed by the boxes' width and

# height

box = detection[0:4] \* np.array([W, H, W, H])

(centerX, centerY, width, height) = box.astype("int")

# use the center (x, y)-coordinates to derive the top

# and and left corner of the bounding box

x = int(centerX - (width / 2))

y = int(centerY - (height / 2))

# update our list of bounding box coordinates,

# confidences, and class IDs

boxes.append([x, y, int(width), int(height)])

confidences.append(float(confidence))

classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping

# bounding boxes

idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],

args["threshold"])

# ensure at least one detection exists

if len(idxs) > 0:

# loop over the indexes we are keeping

for i in idxs.flatten():

# extract the bounding box coordinates

(x, y) = (boxes[i][0], boxes[i][1])

(w, h) = (boxes[i][2], boxes[i][3])

# draw a bounding box rectangle and label on the frame

color = [int(c) for c in COLORS[classIDs[i]]]

cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

text = "{}: {:.4f}".format(LABELS[classIDs[i]],

confidences[i])

cv2.putText(frame, text, (x, y - 5),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

cv2.imshow('outputWindows',frame)

if cv2.waitKey(10) & 0xFF == ord('q'):# Press 'ESC' for exiting video

break

# release the file pointers

print("[INFO] cleaning up...")

vs.release()

cv2.destroyAllWindows()