**PROJECT REPORT:**

**SURVEILLANCE CAM USING OPEN CV**

**Team T-12: VIVEKANAND**

**Team Members:**

**Kritika Bhachawat, 2nd Year, CSE**

**Harshita Mittal, 2nd Year, CSE**

**Vartika Prajapat, 2nd Year, CSE**

**Harsh Soni, 2nd Year, IT**

**Abhijeet Singh Rajpurohit, 2nd Year, CSE**

**Objective**: To make a surveillance camera which can:

1. Enhance the quality of image

2. Detect the motion

3. Human detection

4. Face Recognition

**IDE used:** PyChrarm, VS Code

**Libraries Used:**

1. OpenCV-Python:

OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.

In OpenCV, the CV is an abbreviation form of computer vision, which is defined as a field of study that helps computers to understand the content of digital images such as photographs and videos.

The purpose of computer vision is to understand the content of the images. It extracts descriptions from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.

➢ Our project is mainly based on utilizing the OpenCV methods to achieve the objective listed above.

**Objective:**

1. **Enhancing the quality of the image**:

Used a built-in OpenCV method (convertScaleAbs) to adjust the brightness and contrast of the image. On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, and conversion to an unsigned 8-bit type.

The sequence of operations involved in convertScaleAbs();

² The first operation is to re-scale the source image by the factor-alpha.

² The second is to offset by (add) the factor-beta.

² The third is to compute the absolute value of that sum.

² The fourth is to cast that result (with saturation) to an unsigned char (8-bit).

2. **Motion Detection**:

Videos can be treated as a stack of pictures called frames. Here we compared different frames(pictures) to the first frame which should be static(No movements initially). We compare two images by comparing the intensity value of each pixel.

Steps:

Gray Scale: First we convert our incoming colour frame to gray scale and store it as a reference. Grayscale images are less complex and on which operations are easy to perform. In grey pictures, there is only one intensity value whereas in RGB(Red, Green and Blue) images there are three intensity values. So it would be easy to calculate the intensity difference in grayscale.

Gaussian Blur: The greyscale image is then converted to gaussian through gaussian blur operation. The conversion reduces the error as continue change in pixels between different frames can happen even there is no motion, gaussian blur frame reduces that error.

Threshold Frame: The difference between the pixels of initial gray and the current gaussian frame crosses a certain range, gives us indication a motion. If the intensity difference for a particular pixel is more than 30(in our case) then that pixel will be white and if the difference is less than 30 that pixel will be black.

Drawing Contours: To show that motion is detected, contours are drawn.to find contour and in built OpenCV method used, findContours(), it returns a modified image as the first of three return parameters. there are three arguments in cv.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs a modified image, the contours and hierarchy. contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object. To draw the contours, cv.drawContours function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

3. **Human detection**: To detect human, “haarcascade” human detection algorithm is used in this project. It is computationally less expensive, a fast algorithm, and gives high accuracy.

Addition explanation of algorithm is the next section (4th point).

4. **Face Recognition**: To detect faces, “haarcascade” facial detection algorithm is used in this project. It is computationally less expensive, a fast algorithm, and gives high accuracy.

Working of Algorithm:

Haar-feature selection: A Haar-like feature consists of dark regions and light regions. It produces a single value by taking the difference of the sum of the intensities of the dark regions and the sum of the intensities of light regions. It is done to extract useful elements necessary for identifying an object.

Creation of Integral Images: A given pixel in the integral image is the sum of all the pixels on the left and all the pixels above it. Since the process of extracting Haar-like features involves calculating the difference of dark and light rectangular regions, the introduction of Integral Images reduces the time needed to complete this task significantly.

AdaBoost Training: This algorithm selects the best features from all features. It combines multiple “weak classifiers” (best features) into one “strong classifier”. The generated “strong classifier” is basically the linear combination of all “weak classifiers”.

Cascade Classifier: It is a method for combining increasingly more complex classifiers like AdaBoost in a cascade which allows negative input (non-face) to be quickly discarded while spending more computation on promising or positive face-like regions. It significantly reduces the computation time and makes the process more efficient.

To capture the face, the face detection method was applied to the grayscale frame of the video. This is done using the cv2::CascadeClassifier::detectMultiScale method, which returns boundary rectangles for the detected faces. (the further process is discussed in the later section)

***OpenCV Functions Used:***

* **.videoCaputre -** cv2. VideoCapture( ) to get a video capture object for the camera.
* **.CascadeClassifier-**  It is a method for combining increasingly more complex classifiers like AdaBoost in a cascade which allows negative input (non-face) to be quickly discarded while spending more computation on promising or positive face-like regions
* **.cvtColor-** cvtColor() method is used to convert an image from one color space to another.
* **.gaussianBlur-** In the Gaussian Blur operation, the image is convolved with a Gaussian filter instead of the box filter. The Gaussian filter is a low-pass filter that removes the high-frequency components are reduced**.**
* **.read-** used to read at most n bytes from the file associated with the given file descriptor. If the end of the file has been reached while reading bytes from the given file descriptor, os. read() method will return an empty bytes object for all bytes left to be read.
* **.blur-** cv2.blur() method is used to blur an image using the normalised box filter
* **.absdiff-** absdiff is a function which helps in finding the absolute difference between the pixels of the two image arrays. By using this we will be able to extract just the pixels of the objects that are moving.
* **.threshold-** Thresholding is a method of image segmentation, in general, it is used to create binary images.
* **.dilate-** Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image
* **.findcontours:** Function that helps in extracting the contours from the image. It works best on binary images, so we should first apply thresholding techniques, Sobel edges, etc.

We see that there are three essential arguments in cv2.findContours() function. First one is source image, second is contour retrieval mode, third is contour approximation method and it outputs the image, contours, and hierarchy. ‘contours‘ is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object.

* **.contourArea:**

Syntax: cv.contourArea (contour, oriented = false)

*contour*: input vector of 2D points (contour vertices)

*oriented*: oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.

* **.THRESH\_BINARY**:

If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).

* **.RETR\_EXTERNAL:**

Gives "outer" contours, so if we have (say) one contour enclosing another (like concentric circles), only the outermost is given

* **.boundingRect :**

The cv2.boundingRect() function of OpenCV is used to draw an approximate rectangle around the binary image. This function is used mainly to highlight the region of interest after obtaining contours from an image.

* **.rectangle:**

*Syntax*: cv2.rectangle(image, start\_point, end\_point, colour, thickness)

*Parameters*:

*image*: It is the image on which the rectangle is to be drawn.

*start\_point*: It is the starting coordinates of a rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

*end\_point*: It is the ending coordinates of a rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

*color*: It is the colour of borderline of rectangle to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue colour.

*thickness*: It is the thickness of the rectangle borderline in px. Thickness of -1 px will fill the rectangle shape by the specified colour.

Return Value: It returns an image.

* **.detectMultiScale:**

*Syntax*:

> *image:* Imported image to check.

> *scaleFactor*: Parameter specifying how much the image size is reduced at each image scale. Basically, the scale factor is used to create your scale pyramid. More explanation, your model has a fixed size defined during training, which is visible in the XML. This means that this size of the face is detected in the image if present. However, by rescaling the input image, you can resize a larger face to a smaller one, making it detectable by the algorithm.

> *minNeighbors* – Parameter specifying how many neighbors each candidate rectangle should have to retain it.

This parameter will affect the quality of the detected faces. Higher value results in fewer detections but with higher quality. 3~6 is a good value for it.

* **.imshow:**

*>Syntax*: cv2.imshow(window\_name, image)

*>Parameters*:

*window\_name*: A string representing the name of the window in which image to be displayed.

*image*: It is the image that is to be displayed.

*>Return Value*: It doesn’t returns anything.

It is a function which we used to display the screen.

* **.waitKey:**

Function of Python OpenCV allows users to display a window for given milliseconds or until any key is pressed. It takes time in milliseconds as a parameter and waits for the given time to destroy the window, if 0 is passed in the argument it waits till any key is pressed.

* **.release**:

Closes video file or capturing device.

The methods are automatically called by subsequent VideoCapture::open and by VideoCapture destructor.

* **.destroyAllWindows**:

This function allows users to destroy or close all windows at any time after exiting the script. If you have multiple windows open at the same time, and you want to close, then you would use this function.

**Approach**:

1) Importing the required libraries, body and face cascade files and testing file (video file).

2) Initializing the loop and reading video frame in each loop.

3) Converting the image to gray, to gaussianblur and to blur consequently.

4) To reduce the accuracy (improve functionality) we get the difference of every 10th frame to compare and get the motion.

5) Making contours for if the motion detected.

6) If motion happens we check for human, then if human detected in motion we check for face, and draw contours.

7) The imported video will continue to function until the video ends or key (‘q’) is pressed.

**Github repository Link:**

>>