

SMART CCTV BY MACHINE LEARNING

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology Degree in

Information Technology

By

SNEHA .J(Reg.no.37120071)

VEDHANAYAKI.K(37120082)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119. TAMIL NADU

MARCH 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" Grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119.

Phone: 044 - 2450 3150 / 51 / 52 / 54 / 55 Fax: 044 - 2450 2344

www.sathyabama.ac.in



SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of SNEHA K (Reg. No.37120071) & SUPRIYA S (Reg. no.37120082) who underwent the professional training in **"Smart Cctv by Machine Learning"** under our supervision from December 2020 to March 2021.

Internal Guide

MS.VIMALI.J.S, ME.Ph.D.,

Head of the Department

Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, SNEHA (Reg. No. 37120071) & VEDHANAYAKI (37120082) hereby declare that the Professional Training Report on “**Smart cctv by Machine Learning**” done by us under the guidance of MS.VIMALI.J.S,ME., Ph.D., at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of technology degree in **Information Technology**.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to the Board of **Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing , **Dr.R.Subhashini M.E., Ph.D. , Head** of the Department of Information Technology for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **MS.VIMALI.J.S, ME.Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved the way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

CCTV systems are becoming more common these days because they can control what is going on at the place where they will install. In terms of safety and protection, tracking the behavior of staff and visitors at our place is necessary. CCTV services play a vital role in all forms of security problems. Installing a dependable CCTV camera is a practical move, and a surveillance package provides you with a proactive security advantage. If you own a company or just want to protect your family. Routine inspections and repairs must well maintain. Traditional CCTV devices can only constantly record the world; they cannot perform smart functions such as motion detection.

CHAPTER	TITLE	PAGE NO
	ABSTRACT	Iv
	LIST OF FIGURES	Viii
	LIST OF ABBREVIATIONS	Viii
1	INTRODUCTION	1
	1.1 INTRODUCTION TO THE PROJECT	1
	1.2 AIM OF THE PROJECT	1
	1.3 SCOPE OF THE PROJECT	2
2	LITERATURE SURVEY	3
	2.1 A COMPREHENSIVE SURVEY ON POSE-INVARIANT FACE RECOGNITION	3
	2.2 IMAGE CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORK	3
	2.3 LEARNING COMPACT BINARY FACE DESCRIPTOR FOR RECOGNITION	3
3	IMAGE PROCESSING IN PYTHON WITH OPEN CV	4
	3.1 IMAGE PROCESSING SYSTEM	4
	3.1.1 USES	4
	3.2 PRE REQUISITES	4
	3.2.1 INSTALLATION	5
	3.3 IMAGE THRESHOLDING	5
	3.4 OBJECT DETECTION IN VIDEO WITH DEEP LEARNING AND OPEN CV	6
	3.4.1 OBJECT DETECTION	6
	3.4.2 FEATURE EXTRACTION	8

3.4.3 OPEN CV	8
3.4.4 SETTING UP OPENCV	9
3.4.5 IMAGE BASIS IN OPENCV	9
3.4.6 HANDLING FILES AND IMAGES	10
3.4.7 CONSTRUCTING BASIC SHAPES IN OPENCV	10
3.4.8 IMAGE PROCESSING TECHNIQUES	10
3.4.9 HARR-FEATURE BASED CASCADE CLASSIFIER FOR OBJECT DETECTION	11
3.5 DEEP LEARNING WITH OPEN CV	12
3.5.1 DEEP LEARNING INSIDE OPENCV 3.3	12
3.5.2 CONFIGURE YOUR MACHINE WITH OPENCV 3.3	13
3.5.3 OPENCV DEEP LEARNING FUNCTIONS AND FRAMEWORKS	13
3.5.4 CLASSIFYING IMAGES USING DEEP LEARNING AND OPENCV	14
3.5.5 DEEP LEARNING AND OPENCV CLASSIFICATION RESULTS	15
3.6 MACHINE LEARNING IN PYTHON	15
3.7 LEARNING REPRESENTATIONS FROM DATA	16
3.8 THE “DEEP” IN DEEP LEARNING	18
3.9 PROBABLISTIC MODELLING	18
3.10 KERNEL METHODS	19
3.11 DATASETS	19
3.12 THE IRIS DATASET	19
3.13 THE 3-SCENES IMAGE DATASET	20

	3.13.1 STEPS TO PERFORM MACHINE LEARNING IN PYTHON	20
	3.14 MACHINE LEARNING PROJECT STRUCTURE	21
	3.14.1 IMPLEMENTING PYTHON MACHINE LEARNING FOR NUMERICAL DATA	21
	3.14.2 IMPLEMENTING PYTHON MACHINE LEARNING FOR IMAGES	22
	3.15 DETECTION AND RECOGNITION	22
	3.16 FACE RECOGNITION	24
	3.16.1 DATABASE	25
	3.16.2 EIGENFACES IN OPENCV	26
	3.16.3 FISHERFACES IN OPENCV	26
	3.17 FIRE DETECTION	27
	3.17.1 CREATING THE CUSTOMIZED CNN ARCHITECTURE	27
	3.17.2 CREATING CUSTOMIZED INCEPTION V3 MODEL	28
	3.17.3 REAL-TIME TESTING	29
4	METHODOLOGY	31
	4.1 SYSTEM REQUIREMENTS	31
	4.1.1 SOFTWARE REQUIREMENTS	31
	4.1.2 HARDWARE REQUIREMENTS	31
	4.2 ARCHITECTURE DIAGRAM	31
	4.2.1 SYSTEM ARCHITECTURE	32

	4.3 MODULES DESCRIPTION	32
	4.4 ANALYSING AND ACCESSING DATA	34
5	RESULT AND DISCUSSION	37
	RESULT	37
	DISCUSSION	37
6	CONCLUSION AND FUTURE WORK	38
	CONCLUSION	38
	FUTURE WORK	38
	REFERENCE	39
	APPENDIX	40
	A. SOURCE CODE	40
	B. SCREENSHOTS	42
	C. PLAGIARISM REPORT	45

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO
3.4.1	OPENCV ARCHITECTURE	8
3.5	DETECTION ALGORITHM	15
3.5.1	BLOCK DIAGRAM	16
4.2.1	SYSTEM ARCHITECTURE	24

LIST OF ABBREVIATIONS

CCTV	CLOSED CIRCUIT TELEVISION
CNN	CONVOLUTIONAL NEURAL NETWORK
HSV	HUE- SATURATION- BRIGHTNESS
DCNN	DEEP CONVOLUTIONAL NEURAL NETWORK
PIFR	POST INVARIANT FACE RECOGNITION
CV	COMPUTER VISION
GPU	GRAPHICS PROCESS UNIT
API	APPLICATION PROGRAMMING INTERFACE

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION OT THE PROJECT

Nowadays CCTV systems are emerging because CCTV systems can keep track of what is happening at the premises. The traditional CCTV systems are only capable to record the world continuously, It cannot able to perform smart functionalities like detecting motion, etc. Here the Smart CCTV comes into the picture. It can able to detect a motion, Detect a noise, and several other smart functionalities. These functionalities are achieved by AI and machine learning methods.Smart CCTV enables remote monitoring of a home or property, connecting to a device even when away from the premises.In this project, we can able to detect the motion By deploying the AI methods into the CCTV systems, it can able to perform more than our traditional CCTV systems. Our motive is to design a CCTV system that can act smarter.

By using the Machine Learning methods, we deployed several modules to make the CCTV system smarter. The modules are the CCTV system should take a snap whenever motion or noise arises in the screen. We also design the model to monitor a particular portion of the screen and other basic functions.The project Smart CCTV camera surveillance system is to enhance the CCTV camera-based security systems, which presently exist in different places. We design the Project Security System by using CCTV Camera using wireless technology. Using a surveillance system for image detection is becoming more important. We often use an embedded surveillance system for image processing of the surveillance system and also for traffic monitoring, but this configuration requires a high-performance core, which works against some advantages of embedded systems, such as low power consumption and low cost.

1.2 AIM OF THE PROJECT

CCTV systems are becoming more common these days because they can control what is going on at the place where they will install. In terms of safety and protection, tracking the behavior of staff and visitors at our place is necessary.CCTV services play a vital role in all forms of security problems. Installing a dependable CCTV camera is a practical move, and a surveillance package provides you with a proactive security advantage. If you own a company or just want to protect your family.Routine inspections and repairs must well maintain. Traditional CCTV devices can only constantly record the world; they cannot perform smart functions such as motion detection.

1.3 SCOPE OF THE PROJECT

OpenCV was started at Intel in 1999 by Gary Bradsky and the first release came out in 2000. VadimPisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge. Later its active development continued under the support of Willow Garage, with Gary Bradsky and VadimPisarevsky leading the project. Right now, OpenCV supports a lot of algorithms related to Computer Vision and Machine Learning and it is expanding day-by-day. Currently OpenCV supports a wide variety of programming languages like C++, Python, Javaetc and is available on different platforms including Windows, Linux, OS X, Android, iOS etc. Also, interfaces based on CUDA and OpenCLare also under active development for high-speed GPU operations. OpenCV-Python is the Python API of OpenCV. It combines the best qualities of OpenCV C++ API and Python language.OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

CHAPTER 2

LITERATURE SURVEY

2.1 A COMPREHENSIVE SURVEY ON POSE-INVARIANT FACE RECOGNITION(2015)

The capacity to recognize faces under varied poses is a fundamental human ability that presents a unique challenge for computer vision systems. Compared to frontal face recognition, which has been intensively studied and has gradually matured in the past few decades, pose-invariant face recognition (PIFR) remains a largely unsolved problem. However, PIFR is crucial to realizing the full potential of face recognition for real-world applications, since face recognition is intrinsically a passive biometric technology for recognizing uncooperative subjects. In this paper, we discuss the inherent difficulties in PIFR and present a comprehensive review of established techniques. Existing PIFR methods can be grouped into four categories, i.e., pose-robust feature extraction approaches, multi-view subspace learning approaches, face synthesis approaches, and hybrid approaches. The motivations, strategies, pros/cons, and performance of representative approaches are described and compared. Moreover, promising directions for future research are discussed.

2.2 IMAGE NET CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORKS(2012)

Deep convolution neural network of 1.2 million image splits in the Image Net LSVRC-2010 competition in 1000 different categories. In the experimental data, we found 37.5% of the top-1 and top-5 error ratings and 17.0% which is much better than the previous case. The neural network, comprising 60 million parameters and 650,000 neurons, consists of five specific layers, some of which are followed by integration layers, and three fully connected layers with a final 1000 soft max. To make training faster, we have used non-complementary neurons and GPU efficient use of convolution function. To reduce overcrowding in fully integrated layers we have used a newly developed “dropout” approach that has proven to be very effective. We also included variants of this type in the ILSVRC-2012 competition and found a high error error rate of 15.3%, compared to 26.2% achieved with the best entry.

2.3 LEARNING COMPACT BINARY FACE DESCRIPTOR FOR FACE RECOGNITION(2015)

Python is a general purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability. Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules.. This is how Open CV-Python works, it is a Python wrapper around original C++ implementation. Open CV is a huge open-source library for computer vision, machine learning, and image processing. Open CV supports a wide variety of programming languages .like Python,C++,Java,etc.

CHAPTER 3

IMAGE PROCESSING IN PYTHON WITH OPENCV

3.1 IMAGE PROCESSING

Image Processing is most commonly termed as 'Digital Image Processing' and the domain in which it is frequently used is 'Computer Vision'. Don't be confused - we are going to talk about both of these terms and how they connect. Both Image Processing algorithms and Computer Vision (CV) algorithms take an image as input; however, in image processing, the output is also an image, whereas in computer vision the output can be some features/information about the image.

The data that we collect or generate is mostly raw data, i.e. it is not fit to be used in applications directly due to a number of possible reasons. Therefore, we need to analyze it first, perform the necessary pre-processing, and then use it.

For instance, let's assume that we were trying to build a cat classifier. Our program would take an image as input and then tell us whether the image contains a cat or not. The first step for building this classifier would be to collect hundreds of cat pictures. One common issue is that all the pictures we have scraped would not be of the same size/dimensions, so before feeding them to the model for training, we would need to resize/pre-process them all to a standard size. This is just one of many reasons why image processing is essential to any computer vision application.

3.1.1 USES

The data that we collect or generate is mostly raw data, i.e. it is not fit to be used in applications directly due to a number of possible reasons. Therefore, we need to analyze it first, perform the necessary pre-processing, and then use it.

For instance, let's assume that we were trying to build a cat classifier. Our program would take an image as input and then tell us whether the image contains a cat or not. The first step for building this classifier would be to collect hundreds of cat pictures. One common issue is that all the pictures we have scraped would not be of the same size/dimensions, so before feeding them to the model for training, we would need to resize/pre-process them all to a standard size.

This is just one of many reasons why image processing is essential to any computer vision application.

3.2 Prerequisites

Before going any further, let's discuss what you need to know in order to follow this tutorial with ease. Firstly, you should have some basic programming knowledge in any language. Secondly, you should know what machine learning is and the basics of how it works, as we will be using some machine learning algorithms for image processing in this article. As a bonus, it would help if you have had any exposure to, or basic knowledge of, Open CV before going on with this tutorial. But this is not required.

One thing you should definitely know in order to follow this tutorial is how exactly an image is represented in memory. Each image is represented by a set of pixels i.e. a matrix of pixel values. For a grayscale image, the pixel values range from 0 to 255 and they represent the intensity of that pixel. For instance, if you have an image of 20 x 20 dimensions, it would be represented by a matrix of 20x20 (a total of 400-pixel values).

If you are dealing with a colored image, you should know that it would have three channels - Red, Green, and Blue (RGB). Therefore, there would be three such matrices for a single image.

3.2.1 Installation

Note: Since we are going to use OpenCV via Python, it is an implicit requirement that you already have Python (version 3) already installed on your workstation.

Windows

```
$ pip install opencv-python
```

MacOS

```
$ brew install opencv3 --with-contrib --with-python3
```

Linux

```
$ sudo apt-get install libopencv-dev python-opencv
```

To check if your installation was successful or not, run the following command in either a Python shell or your command prompt:

```
import cv2
```

3.3 Image Thresholding

The concept of thresholding is quite simple. As discussed above in the image representation, pixel values can be any value between 0 to 255. Let's say we wish to convert an image into a binary image i.e. assign a pixel either a value of 0 or 1. To do this, we can perform thresholding. For instance, if the Threshold (T) value is 125, then all pixels with values greater than 125 would be assigned a value of 1, and all pixels with values lesser than or equal to that would be assigned a value of 0.

3.4 Object detection in video with deep learning and OpenCV:

To build our deep learning-based real-time object detector with Open CV we'll need to

- (1) access our webcam/video stream in an efficient manner and
- (2) apply object detection to each frame.

To get the system set up, we simply install Open CV using the relevant instructions for the system .Compared to last week, we don't need the image argument since we're working with streams and videos — other than that the following arguments remain the same:

- `--prototxt`
: The path to the Caffe prototxt file.
- `--model`
: The path to the pre-trained model.
- `--confidence`
: The minimum probability threshold to filter weak detections. The default is 20%.

3.4.1 Object detection

Object Detection is a computer technology related to computer vision, image processing, and deep learning that deals with detecting instances of objects in images and videos. We will do object detection in this article using something known as haar cascades. Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features. Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier

- Positive images – These images contain the images which we want our classifier to identify.
- Negative Images – Images of everything else, which do not contain the object we want to detect.

We then initialize a class list and a color set, and then we initialize class labels and corresponding random colors. Next, let's initialize our video stream (this can be from a video file or a camera).The video stream and Fps classes are part of imultis package.Now loop over each and every frame first we read a frame from the stream, followed by resizing it then we set the blob as the input to our neural network and feed the input through the net which give as detections. At this point, we have detected objects in the input frame. It is now time to look at confidence values and determine if we should draw a box + label surrounding the object We starting by looping over our

detections keeping in mind that multiple objects can be detected in a single image. We also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display

Source code:

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2
```

the prediction in the terminal as well as draw the prediction on the image with text and a colored bounding box. Looping through our detections, first we extract the confidence value. If the confidence is above our minimum threshold, we extract the class label index and compute the bounding box coordinates around the detected object. Then, we extract the (x, y)-coordinates of the box which we will use shortly for drawing a rectangle and displaying text. We built a text label containing the class name and the confidence. Also, draw a colored rectangle around the object using our class color and previously extracted (x, y)-coordinates. In general, we want the label to be displayed above the rectangle, but if there isn't room, we'll display it just below the top of the rectangle. Finally, we overlay the colored text onto the frame using the y-value that we just calculated. The remaining steps in the frame capture loop involve

(1) displaying the frame, (2) checking for a quit key, and (3) updating our frames per second counter

Source code:

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

3.4.2 Feature Extraction

Haar feature-based cascade classifiers is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The nice thing about haar feature-based cascade classifiers is that you can make a classifier of any object you want, OpenCV already provided some classifier parameters to you, so you don't have to collect any data to train on it. Neural Network (CNN) places a green pixel map from the front-facing camera to the driving instructions. This powerful end-to-end approach means that with low-level training from people, the system learns to direct, with or without a linear curve, on both local and highways. The plan can also apply

to areas with vague hints such as parking lots or unpaved roads. The system is trained to automatically learn the internal representation of the necessary processing steps, such as finding useful road features, with a personal directional angle only as a training signal. We do not need to explicitly train it to find, for example,

End-of-life learning leads to better performance and smaller programs. Better performance results because the internal components do it themselves to increase the performance of the entire system, instead of improving the intermediate options chosen by man, e. g., route discovery. Such methods are logically preferred to translate people easily which does not automatically verify the operation of the system. Smaller networks are possible because the system learns to solve a problem with a small number of processing steps.

3.4.3 Open cv



Fig. 1.Opencv Architecture

Fig. 1. Opencv Architecture

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces,

or even the handwriting of a human. There are many different edge detection filters, such as Sobel, Scharr, Laplacian filters, or Canny-edge detector. We will use a Laplacian edge filter since it produces edges that look most similar to hand sketches compared to Sobel or Scharr, and that are quite consistent compared to a Canny-edge detector, which produces very clean line drawings but is affected more by random noise in the camera frames and the line drawings therefore often change drastically between frames. A strong bilateral filter smoothes flat regions while keeping edges sharp, and is therefore great as an automatic cartoonifier or painting filter, except that it is extremely slow (that is, measured in seconds or even minutes rather than milliseconds!).

We will therefore use some tricks to obtain a nice cartoonifier that still runs at an acceptable speed. The most important trick we can use is to perform bilateral filtering at a lower resolution. It will have a similar effect as at full resolution, but will run much faster. Let's reduce the total number of pixels by a factor of four (for example, half width and half height). We have four parameters that control the bilateral filter: color strength, positional strength, size, and repetition count. We need a temp Mat since `bilateralFilter()` can't

overwrite its input (referred to as "in-place processing"), but we can apply one filter storing a temp Mat and another filter storing back to the input. We will perform this on a grayscale image with some noise reduction, so the previous code for converting the original image to grayscale and applying a 7 x 7 Median filter should be used by again.

The system is trained to automatically learn the internal representation of the necessary processing steps, such as finding useful road features, with a personal directional angle only as a training signal. We do not need to explicitly train it to find, for example, a roadmap.

End-of-life learning leads to better performance and smaller programs. Better performance results because the internal components do it themselves to increase the performance of the entire system, instead of improving the intermediate options chosen by man, e. g., route discovery. Such methods are logically preferred to translate people easily which does not automatically verify the operation of the system. Smaller networks are possible because the system learns to solve a problem with a small number of processing steps.

3.4.4 Setting up with openCV

Mastering OpenCV 4 with Python will give you the knowledge to build projects involving Open Source Computer Vision Library (OpenCV) and Python. These two technologies (the first one is a programming language, while the second one is a computer vision and machine learning library) will be introduced. Also, you will learn why the combination of OpenCV and Python has the potential to build every kind of computer application. Finally, an introduction about the main concepts related to the content of this book will be provided.

In this chapter, you will be given step-by-step instructions to install everything you need to start programming with Python and OpenCV. This first chapter is quite long, but do not worry, because it is divided into easily assimilated sections, starting with general terminology and concepts, which assumes that the reader is new to this information. At the

end of this chapter, you will be able to build your first project involving Python and OpenCV.

3.4.5 Image Basis in OpenCV

Images are a key component in a computer vision project because they provide, in many cases, the input to work with. Therefore, understanding main image concepts is the basic knowledge you need to start coding your computer vision projects. Also, some of the OpenCV library peculiarities, such as the coordinate system or the BGR order (rather than RGB), will be introduced.

In this chapter, you will learn how to start writing your first scripts, which will introduce you to the OpenCV library. At the end of this chapter, you will have enough knowledge to start programming your first computer vision project in OpenCV and Python.

3.4.6 Handling Files and Images

In any kind of project, coping with files and images is a key aspect. In this sense, many projects should work with files as forms of data input. Additionally, the project can generate some data after any kind of processing has been done, which can be outputted in the form of files or images. In computer vision, this information flow (input-processing-output) takes special relevance due to the inherent characteristics of these types of projects (for example, images to be processed and models that are generated by machine learning algorithms).

In this chapter, we are going to see how we can handle both files and images. You will learn how to cope with files and images, which are necessary for building computer vision applications.

3.4.7 Constructing Basics shapes in OpenCV

One basic functionality offered by OpenCV is drawing basic shapes. OpenCV provides functions to draw lines, circles, rectangles, ellipses, and so on. When building a computer vision project, you usually want to modify the image by drawing some shapes. For example, if you develop a face detection algorithm, you should draw a rectangle highlighting the detected faces in the computed image. Additionally, if you develop a face recognition algorithm, you should draw a rectangle highlighting the detected faces and also write some text showing the identity of the detected faces. And finally, it is a common approach to write some **debugging** information. You could show, for example, the number of detected faces (in order to see the performance of your face detection algorithm) or the processing time. In this chapter, you are going to see how to draw basic and more advanced shapes using the OpenCV library.

3.4.8 Image Processing Techniques

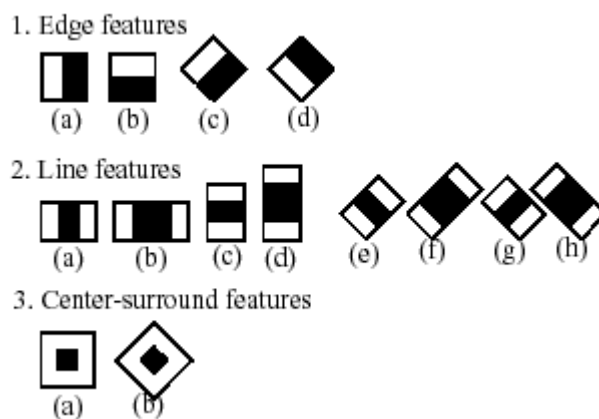
Image processing techniques are the core of your computer vision projects. They can be seen as useful key tools, which you can use to complete various tasks. In other words, image processing techniques are like building blocks that should be kept in mind when processing your images. Therefore, a basic understanding of image processing is required if you are to work with computer vision projects.

In this chapter, you will learn most of the common image processing techniques you need. These will be complemented by the other image processing techniques covered in the next three chapters of this book (histograms, thresholding techniques, contour detection, and filtering).

3.4.9 Haar-Feature based Cascade classifier for object detection

The object detector described below has been initially proposed by Paul Viola and improved by Rainer Lienhart First, a classifier is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size. After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole

image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales. The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below. The current algorithm uses the following Haar-like features:



The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position

within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular regions are calculated rapidly using integral images (see below and the integral description). There is a separate application called `opencv_traincascade` that can train a cascade of boosted classifiers from a set of samples.

Groups the object candidate rectangles.

Parameters:

- `rectList` Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.)
- `groupThreshold` Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it.
- `eps` Relative difference between sides of the rectangles to merge them into a group.

The function is a wrapper for the generic function `partition`. It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by `eps`. When `eps=0`, no clustering is done at all. If `eps→+inf`, all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to `groupThreshold` rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.

3.5 Deep Learning with OpenCV

In the past few years, artificial intelligence (AI) has been a subject of intense media hype. Machine learning, deep learning, and AI come up in countless articles, often outside of technology-minded publications. We're promised a future of intelligent chatbots, self-driving cars, and virtual assistants—a future sometimes painted in a grim light and other times as utopian, where human jobs will be scarce and most economic activity will be handled by robots or AI agents. For a future or current practitioner of machine learning, it's important to be able to recognize the signal in the noise so that you can tell world-changing developments from overhyped press releases. Our future is at stake, and it's a future in which you have an active role to play: after reading this book, you'll be one of those who develop the AI agents

In the first part, we'll discuss the OpenCV 3.3 release and the overhauled Dnn Module. We'll then write a Python script that will use OpenCV and GoogleLeNet (pre-trained on ImageNet) to classify images. Finally, we'll explore the results of our classifications.

3.5.1 Deep Learning inside OpenCV 3.3

The `dnn` module of OpenCV has been part of the `Opencv_contrib` repository since version v3.1. Now in OpenCV 3.3, it is included in the major repository. Deep Learning is a fast-growing domain of Machine Learning, and if you're working in computer vision/image processing already (or getting up to speed), it's a crucial area to explore.

With OpenCV 3.3, we can use pre-trained networks with popular deep learning frameworks. That they are pre-trained implies that we don't need to spend many hours training the network — rather we can complete a forward pass and use the output to decide within our application.

OpenCV does not (and does not intend to be) to be a tool for training networks — there are already great frameworks available for that purpose. Since a network (such as a CNN) can be a classifier, it makes logical sense that OpenCV has a Deep Learning module that we can leverage easily within the OpenCV ecosystem.

Popular network architectures compatible with OpenCV 3.3 include:

- GoogleLeNet (used in this blog post)
- AlexNet
- SqueezeNet
- VGGNet (and associated flavors)
- ResNet

The release notes for this module are available on the OpenCV repository page.

Aleksandr Rybnikov, the main contributor for this module, has ambitious plans for this module so be sure to stay on the lookout and read his release notes (in Russian, so make sure you have Google Translation enabled in your browser if Russian is not your native language). It's my opinion that the `dnn` module will have a big impact on the OpenCV community, so let's get the word out.

3.5.2 Configure your machine with OpenCV 3.3

Installing OpenCV 3.3 is on par with installing other versions. The same install tutorials can be utilized — just make sure you download and use the correct release.

Simply follow these instructions for macOS or Ubuntu while making sure to use the OpenCV and `opencv_contrib` releases for OpenCV 3.3. If you opt for the MacOS + homebrew install instructions, be sure to use the `--HEAD` switch (among the others mentioned) to get the bleeding edge version of OpenCV. If you're using virtual environments (highly recommended), you can easily install OpenCV 3.3 alongside a previous version. Just create a brand new virtual environment (and name it appropriately) as you follow the tutorial corresponding to your system.

3.5.3 OpenCV deep learning functions and frameworks

OpenCV 3.3 supports the Caffe, TensorFlow, and Torch/Approach frameworks. Keras is currently not supported (since Keras is a wrapper around backends such as TensorFlow and Theano), although I imagine it's only a matter of time until Keras is directly supported given the popularity of the deep learning library.

Using OpenCV 3.3 we can load images from disk using the following functions inside Dnn:

- `cv2.dnn.blobFromImage`
- `cv2.dnn.blobFromImages`

We can directly import models from various frameworks via the “create” methods:

- `cv2.dnn.createCaffeImporter`
- `cv2.dnn.createTensorFlowImporter`
- `cv2.dnn.createTorchImporter`

Once we have loaded a model from disk, the `.forward`` method is used to forward-propagate our image and obtain the actual classification.

3.5.4 Classifying images using deep learning and OpenCV

In this, we'll be creating a Python script that can be used to classify input images using OpenCV and GoogLeNet (pre-trained on ImageNet) using the Caffe framework. The GoogLeNet architecture (now known as “Inception” after the novel micro-architecture) was introduced by Szegedy et al. in their 2014 paper, Going deeper with convolutions. It also supported other architectures with OpenCV 3.3 including AlexNet, ResNet, and SqueezeNet — we'll be examining these architectures for deep learning with OpenCV in a future blog post. In the meantime, let's learn how we can load a pre-trained Caffe model and use it to classify an image using OpenCV. To begin, open up a new file, name it `deep_learning_with_opencv.py`. We can create an argument parser followed by establishing four required command line arguments

- `--image`
: The path to the input image.
- `--prototxt`
: The path to the Caffe “deploy” prototxt file.
- `--model`
: The pre-trained Caffe model (i.e., the network weights themselves).
- `--labels`
: The path to ImageNet labels (i.e., “syn-sets”).

Now that we've established our arguments, we parse them and store them in a variable, `args`, for easy access later. We load the image from disk via `cv2.imread`. As you can see, we have a unique identifier followed by a space, some class labels, and a new-line. Parsing this file line-by-line is straightforward and efficient using Python. First, we load the class label rows from disk into a list. To do this we strip whitespace from the beginning and end of each line while using the new-line (`'\n'`) as the row delimiter. Second, we use list comprehension to extract the relevant class labels from rows by looking for the space (`' '`) after the ID, followed by delimiting class labels with a comma (`','`).

Since we've opted to use Caffe, we utilize `cv2.dnn.readNetFromCaffe` to load our Caffe model definition `prototxt` and pre-trained model from disk. If you are familiar with Caffe, you'll recognize the `prototxt` file as a plain text configuration which follows a JSON-like structure. I recommend that you open `bvlc_googlenet.prototxt` from the "Downloads" section in a text editor to inspect it.

It is important to note at this step that we aren't training a CNN — rather, we are making use of a pre-trained network. Therefore we are just passing the blob through the network (i.e., forward propagation) to obtain the result (no back-propagation). First, we specify blob as our input. Second, we make a start timestamp, followed by passing our input image through the network and storing the predictions. Finally, we set an end timestamp so we can calculate the difference and print the elapsed time. Using NumPy, we can easily sort and extract the top five predictions. The idea for this loop is to

- (1) draw the top prediction label on the image itself and
- (2) print the associated class label probabilities to the terminal. Lastly, we display the image to the screen and wait for the user to press a key before exiting

3.5.5 Deep learning and OpenCV classification results

Now that we have implemented our Python script to utilize deep learning with OpenCV, let's go ahead and apply it to a few example images. Make sure you use the "Downloads" section of this blog post to download the source code + pre-trained GoogLeNet architecture + example images. In the above example, we have Jemma, the family beagle. Using OpenCV and GoogLeNet we have correctly classified this image as "beagle". Furthermore, inspecting the top-5 results we can see that the other top

predictions are also relevant, all of them of which are dogs that have similar physical appearances as beagles. Taking a look at the timing we also see that the forward pass took < 1 second, even though we are using our CPU. Keep in mind that the forward pass is substantially faster than the backward pass as we do not need to compute the gradient and backpropagate through the network.

3.6 Machine Learning in Python

Machine learning is an application of artificial intelligence that provides computers (and other systems with the certain capability of computation) with the capability to automatically make predictions or decisions from experience without being explicitly programmed to carry out the task. The concept of machine learning has been around for a long time, but it has been gaining momentum over the last several years mainly due to three key factors:

- There is enormously increased data.
- There are significantly improved algorithms.
- There is substantially more powerful computer hardware. Virtual personal assistants (for example, smart speakers or mobile apps), predictions while commuting (traffic predictions

or navigation services), video systems (surveillance camera systems or license plate recognition systems), and e-commerce applications (recommendation systems or automatic price-comparison applications) are only a few examples of machine learning applications...

To Install the required Python machine learning libraries Before we can get started with this, you first need to make sure your system is configured for machine learning. code requires the following libraries:

- **NumPy:** For numerical processing with Python.
- **PIL:** A simple image processing library.
- **scikit-learn:** Contains the machine learning algorithms we'll cover today (we'll need version 0.20+ which is why you see the --upgrade the flag below).
- **Keras and TensorFlow:** For deep learning. The CPU version of TensorFlow is fine, for today's example.
- **OpenCV:** While we aren't using OpenCV for this blog post, **imutils** depend upon it (next bullet). Because of this, you can simply use pip to install OpenCV, just remember you won't have the full install of OpenCV and you can't customize it.
- **imutils:** package of image processing/computer vision convenience functions

3.7 Learning Representation from data

To define deep learning and understand the difference between deep learning and other machine-learning approaches, first we need some idea of what machine-learning algorithms do. I just stated that machine learning discovers rules to execute a data-processing task, given examples of what's expected. So, to do machine learning, we need three things

- Input data points— For instance, if the task is speech recognition, these data points could be sound files of people speaking. If the task is image tagging, they could be pictures.
- Examples of the expected output— In a speech-recognition task, these could be human-generated transcripts of sound files. In an image task, expected outputs could be tags such as “dog,” “cat,” and so on.
- A way to measure whether the algorithm is doing a good job— This is necessary in order to determine the distance between the algorithm's current output and its expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call learning.

A machine-learning model transforms its input data into meaningful outputs, a process that is “learned” from exposure to known examples of inputs and outputs. Therefore, the central problem in machine learning and deep learning is to meaningfully transform data: in other words, to learn useful representations of the input data at hand—representations that get us closer to the expected output. Before we go any further: what's a representation? At its core, it's a different way to look at data—to represent or encode data. For instance, a color image can be encoded in the RGB format (red-green-blue) or in the HSV format (hue-saturation-value): these are two different representations of the same data. Some tasks that may be difficult with one representation can become easy with another. For example, the task “select all red pixels in the image” is simpler in the RGB

format, whereas “make the image less saturated” is simpler in the HSV format. Machine-learning models are all about finding appropriate representations for their input data—transformations of the data that make it more amenable to the task at hand, such as a classification task.

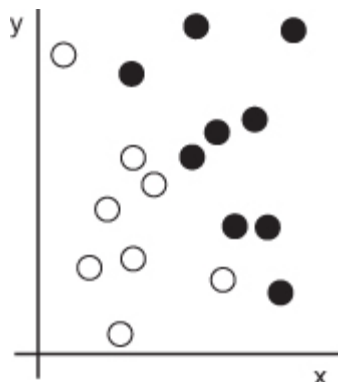


Fig 3.7 Some sample data

As you can see, we have a few white points and a few black points. Let’s say we want to develop an algorithm that can take the coordinates (x, y) of a point and output whether that point is likely to be black or to be white.

- The inputs are the coordinates of our points.
- The expected outputs are the colors of our points.
- A way to measure whether our algorithm is doing a good job could be, for instance, the percentage of points that are being correctly classified.

What we need here is a new representation of our data that cleanly separates the white points from the black points. One transformation we could use, among many other possibilities, would be a coordinate change. In this new coordinate system, the coordinates of our points can be said to be a new representation of our data. And it’s a good one! With this representation, the black/white classification problem can be expressed as a simple rule: “Black points are such that $x > 0$,” or “White points are such that $x < 0$.” This new representation basically solves the classification problem. In this case, we defined the coordinate change by hand. But if instead we tried systematically searching for different possible coordinate changes, and used as feedback the percentage of points being correctly classified, then we would be doing machine learning. Learning, in the context of machine learning, describes an automatic search process for better representations.

All machine-learning algorithms consist of automatically finding such transformations that turn data into more-useful representations for a given task. These operations can be coordinate changes, as you just saw, or linear projections (which may destroy information), translations, nonlinear operations (such as “select all points such that $x > 0$ ”), and so on. Machine-learning algorithms aren’t usually creative in finding these transformations; they’re merely searching through a predefined set of operations, called a hypothesis space. So that’s what machine learning is, technically: searching for useful

representations of some input data, within a predefined space of possibilities, using guidance from a feedback signal. This simple idea allows for solving a remarkably broad range of intellectual tasks, from speech recognition to autonomous car driving. Now that you understand what we mean by learning, let's take a look at what makes deep learning special

3.8 The “Deep” In Deep Learning

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations. The deep in deep learning isn't a reference to any kind of deeper understanding achieved by the approach; rather, it stands for this idea of successive layers of representations. How many layers contribute to a model of the data is called the depth of the model. Other appropriate names for the field could have been layered representations learning and hierarchical representations learning. Modern deep learning often involves tens or even hundreds of successive layers of representations—and they're all learned automatically from exposure to training data. Meanwhile, other approaches to machine learning tend to focus on learning only one or two layers of representations of the data; hence, they're sometimes called shallow learning.

In deep learning, these layered representations are (almost always) learned via models called neural networks, structured in literal layers stacked on top of each other. The term neural network is a reference to neurobiology, but although some of the central concepts in deep learning were developed in part by drawing inspiration from our understanding of the brain, deep-learning models are not models of the brain. There's no evidence that the brain implements anything like the learning mechanisms used in modern deep-learning models. You may come across pop-science articles proclaiming that deep learning works like the brain or was modeled after the brain, but that isn't the case. It would be confusing

and counterproductive for newcomers to the field to think of deep learning as being in any way related to neurobiology; you don't need that shroud of “just like our minds” mystique and mystery, and you may as well forget anything you may have read about hypothetical links between deep learning and biology. For our purposes, deep learning is a mathematical framework for learning representations from data

3.9 Probabilistic model

Probabilistic modeling is the application of the principles of statistics to data analysis. It was one of the earliest forms of machine learning, and it's still widely used to this day. One of the best-known algorithms in this category is the Naive Bayes algorithm.

Naive Bayes is a type of machine-learning classifier based on applying Bayes' theorem while assuming that the features in the input data are all independent (a strong, or “naive” assumption, which is where the name comes from). This form of data analysis predates computers and was applied by hand decades before its first computer implementation (most likely dating back to the 1950s). Bayes' theorem and the foundations of statistics date back to the eighteenth century, and these are all you need to start using Naive Bayes classifiers.

A closely related model is the logistic regression (logreg for short), which is sometimes considered to be the “hello world” of modern machine learning. Don’t be misled by its name—logreg is a classification algorithm rather than a regression algorithm. Much like Naive Bayes, logreg predates computing by a long time, yet it’s still useful to this day, thanks to its simple and versatile nature. It’s often the first thing a data scientist will try on a dataset to get a feel for the classification task at hand.

3.10 Kernel Methods

As neural networks started to gain some respect among researchers in the 1990s, thanks to this first success, a new approach to machine learning rose to fame and quickly sent neural nets back to oblivion: kernel methods. Kernel methods are a group of classification algorithms, the best known of which is the support vector machine (SVM). The modern formulation of an SVM was developed by Vladimir Vapnik and Corinna Cortes in the early 1990s at Bell Labs and published in 1995, although an older linear formulation was published by Vapnik and Alexey Chervonenkis as early as 1963. Vladimir Vapnik and Corinna Cortes, “Support-Vector Networks,” *Machine Learning* 20, no. 3 (1995): 273–297. Vladimir Vapnik and Alexey Chervonenkis, “A Note on One Class of Perceptrons,” *Automation and Remote Control* 25 (1964).

SVMs aim at solving classification problems by finding good decision boundaries between two sets of points belonging to two different categories. A decision boundary can be thought of as a line or surface separating your training data into two spaces corresponding to two categories. To classify new data points, you just need to check which side of the decision boundary they fall on.

3.11 Datasets

To help you gain experience performing machine learning in Python, we’ll be working with two separate datasets. The first one, the Iris dataset, is the machine learning practitioner’s equivalent of “*Hello, World!*” (likely one of the first pieces of software you wrote when learning how to program). The second dataset, 3-scenes, is an example image dataset I put together — this dataset will help you gain experience working with image data, and most importantly, learn what techniques work best for numerical/categorical datasets vs. image datasets. Let’s go ahead and get a more intimate look at these datasets.

3.12 The Iris dataset

The Iris dataset is arguably one of the most simplistic machine learning datasets — it is often used to help teach programmers and engineers the fundamentals of machine learning and pattern recognition.

We call this dataset the “*Iris dataset*” because it captures attributes of three Iris flower species:

1. *Iris Setosa*
2. *Iris Versicolor*
3. *Iris Virginica*

Each species of flower is quantified via four numerical attributes, all measured in centimeters:

1. Sepal length
2. Sepal width
3. Petal length
4. Petal width

Our goal is to train a machine learning model to correctly predict the *flower species* from the measured attributes. It's important to note that one of the classes is linearly separable from the other two — the latter are *not* linearly separable from each other. In order to correctly classify these the flower species, we will need a non-linear model. It's extremely common to need a non-linear model when performing machine learning with Python in the real world — the rest of this tutorial will help you gain this experience and be more prepared to conduct machine learning on your own datasets.

3.13 The 3-scenes image dataset

The second dataset we'll be using to train machine learning models is called the 3-scenes dataset and includes 948 total images of 3 scenes:

- Coast (360 of images)
- Forest (328 of images)
- Highway (260 of images)

The 3-scenes dataset was created by sampling the 8-scenes dataset from Oliva and Torralba's 2001 paper, Modeling the shape of the scene: a holistic representation of the spatial envelope.

3.13.1 Steps to perform machine learning in Python

Whenever you perform machine learning in Python I recommend starting with a simple 5-step process:

1. Examine your problem
2. Prepare your data (raw data, feature extraction, feature engineering, etc.)
3. Spot-check a set of algorithms
4. Examine your results
5. Double-down on the algorithms that worked best

This pipeline will evolve as your machine learning experience grows, but for beginners, this is the machine learning process I recommend for getting started. The more you apply machine learning in Python, the more experience you will gain. Based on your previous experience you may already know an algorithm that works well. From there, you need to prepare your data. Typically this step involves loading your data from disk, examining it, and deciding if you need to perform feature extraction or feature engineering. Feature

extraction is the process of applying an algorithm to quantify your data in some manner. For example, when working with images we may wish to compute histograms to summarize the distribution of pixel intensities in the image — in this manner, we can characterize the color of the image. Feature engineering, on the other hand, is the process of transforming your raw input data into a representation that better represents the underlying problem. Feature engineering is a more advanced technique and one I recommend you explore once you already have some experience with machine learning and Python. the following machine learning algorithms in your toolbox:

- A linear model (ex. Logistic Regression, Linear SVM),
- A few non-linear models (ex. RBF SVMs, SGD classifiers),
- Some tree and ensemble-based models (ex. Decision Trees, Random Forests).
- A few neural networks, if applicable (Multi-layer Perceptrons, Convolutional Neural Networks)

Try to bring a robust set of machine learning models to the problem — your goal here is to gain experience on your problem/project by identifying which machine learning algorithms performed well on the problem and which ones did not.

3.14 machine learning project structure

The Iris dataset is built into scikit-learn. The 3-scenes dataset, however, is not. I've included it in the 3scenes/directory and as you can see there are three subdirectories (classes) of images.

We'll be reviewing four Python machine learning scripts today:

- `classify_iris.py`
: Loads the Iris dataset and can apply any one of seven machine learning algorithms with a simple command line argument switch.
- `classify_images.py`
: Gathers our image dataset (3-scenes) and applies any one of seven Python machine learning algorithms
- `nn_iris.py`
: Applies a simple multi-layer neural network to the Iris dataset
- `basic_cnn.py`
: Builds a Convolutional Neural Network (CNN) and trains a model using the 3-scenes dataset

3.14.1 Implementing Python machine learning for numerical data

The first script we are going to implement is `classify_iris.py` this script will be used to spot-check machine learning algorithms on the Iris dataset. Once implemented, we'll be able to use `classify_iris.py` to run a suite of machine learning algorithms on the Iris dataset, look at the results, and decide on which algorithm works best for the project.

- Our Python machine learning methods from scikit-learn
- A dataset splitting method used to separate our data into training and testing subsets
- The classification report utility from scikit-learn which will print a summarization of our machine learning results
- Our Iris dataset, built into scikit-learn A tool for command line argument parsing called Argparse

3.14.2 Implementing Python machine learning for images

`classify_images.py` , is used to train the same suite of machine learning algorithms above, only on the 3-scenes image dataset. It is very similar to our previous Iris dataset classification script, so be sure to compare the two as you follow along. Most machine learning algorithms perform very poorly on raw pixel data. Instead, we perform feature extraction to characterize the contents of the images.

Here we seek to quantify the color of the image by extracting the mean and standard deviation for each color channel in the image. Given three channels of the image (Red, Green, and Blue), along with two features for each (mean and standard deviation), we have $3 \times 2 = 6$ total features to quantify the image. We form a feature vector by concatenating the values.

3.15 Detection and Recognition

Face processing is a hot topic in artificial intelligence because a lot of information can be automatically extracted from faces using computer vision algorithms. The face plays an important role in visual communication because a great deal of non-verbal information, such as identity, intent, and emotion, can be extracted from human faces. Face processing is a really interesting topic for computer vision learners because it touches on different areas of expertise, such as object detection, image processing, and landmark detection or object tracking.

In this chapter, you will be introduced to the main topics related to face processing using state-of-the-art algorithms and techniques in order to achieve impressive results.

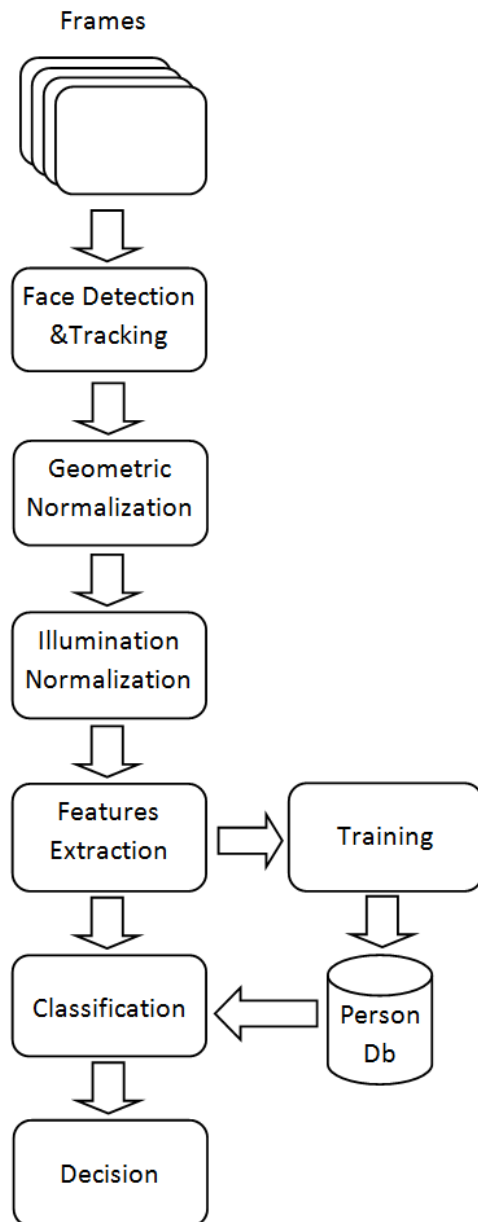


Fig. 2. Face Detection Algorithm

Simple colour thresholds dependent on HSV (Hue-Saturation-Brightness) or RGB (Red-Green-Blue) values, as well as colour histogram estimation and re-projection, as well as advanced Mixture model machine-learning algorithms including CIE Lab colour space camera tuning and offline preparation for multiple sample faces, are used to detect skin regions. Also the most sophisticated approaches, however, do not always work effectively in a variety of camera and lighting conditions, as well as for different skin types. We can use a basic skin-detection system because We want our skin detection to work without any calibration or planning on a mobile device, and we're just using it as a "fun" image filter. However, since the colour response of mobile device camera sensors varies significantly, we need something more powerful than basic colour thresholds to identify skin for people of any skin colour without calibration.

3.12 Face Recognition

With the development of computer vision, machine learning, and deep learning, face recognition has become a hot topic. Face recognition can be applied to a wide range of uses, including crime prevention, surveillance, forensic applications, biometrics, and, more recently, in social networks. Automatic face recognition has various challenges, such as occlusions, appearance variations, expression, aging, and scale variations. Following its success with object recognition, CNNs have been widely used for face recognition. In this chapter, we will see the functionality that OpenCV offers in connection with face recognition, and will also explore some deep learning approaches, which can be easily integrated into your computer vision projects to perform state-of-the-art face recognition results.

The AT&T Face collection, also known as the ORL Faces Database, features ten separate photographs of each of the 40 subjects. The photographs of those participants were captured at different times of the day, with different lighting, facial expressions (open vs. closed eyes, smile vs. no smile), and facial characteristics (glasses vs. no glasses). The models were photographed in a frontal, erect position against a dark, uniform background.

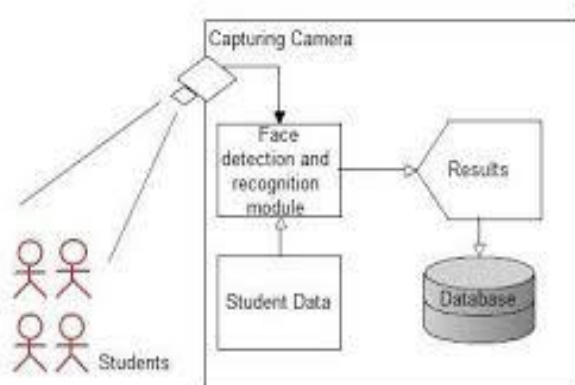


Fig. 3. Block Diagram of face recognition

Fisherfaces and Eigenfaces use a more An all-encompassing approach to facial recognition. In high-dimensional image space, the data is viewed as a vector. Since high dimensionality is negative, a lower-dimensional subspace is located where (hopefully) valuable knowledge can be retained. Where the variation is caused from an external source, the Eigenfaces solution maximises overall scatter, which can cause issues.

We used a Fisherfaces process and optimised it with To hold any discriminative results, use a Linear Discriminant Analysis. At least in our model's constrained case, the Fisherfaces method performed admirably.

Now, life isn't always fine. You obviously cannot guarantee perfect lighting or ten separate images of an individual in your images. So, what if each individual only has one image? Our covariance estimates for the subspace, as well as the identification, could be terribly incorrect. On the AT&T Facedatabase, the Eigenfaces approach had a 96 percent recognition score. How many photographs do you need to provide some precise estimates? The Rank-1 identification the Eigenfaces' rates and Fisher faces methods on AT&T Facedatabase, a quick image database, are as follows:

The query module retrieves site information stored in indexes to process all user queries. Some famous web pages may be retrieved directly from the page repository if their information is not stored in indexes. The rating module mentioned in the following subsection will filter the results shown on the user's computer screen.

3.12.1 Face Database

- AT&T Facedatabase The AT&T Facedatabase, sometimes also referred to as ORL Database of Faces, contains ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).
- Yale Facedatabase A, also known as Yalefaces. The AT&T Facedatabase is good for initial tests, but it's a fairly easy database. The Eigenfaces method already has a 97% recognition rate on it, so you won't see any great improvements with other algorithms. The Yale Facedatabase A (also known as Yalefaces) is a more appropriate dataset for initial experiments, because the recognition problem is harder. The database consists of 15 people (14 male, 1 female) each with 11 grayscale images sized 320×243 pixel. There are changes in the light conditions (center light, left light, right light), facial expressions (happy, normal, sad, sleepy, surprised, wink) and glasses (glasses, no-glasses).
- The original images are not cropped and aligned. Please look into the Appendix for a Python script, that does the job for you.
- Extended Yale Facedatabase B The Extended Yale Facedatabase B contains 2414 images of 38 different people in its cropped version. The focus of this database is set on extracting features that are robust to illumination, the images have almost no variation in emotion/occlusion/ I personally think, that this dataset is too large for the experiments I perform in this document. You better use the AT&T Facedatabase for initial testing. A first version of the Yale Facedatabase B was used in to see how the Eigenfaces and Fisherfaces method perform under heavy illumination changes. used the same setup to take 16128 images of 28 people. The Extended Yale Facedatabase B is the merge of the two databases, which is now known as Extended Yalefacedatabase B.

3.12.2 Eigenfaces in OpenCV

The problem with the image representation we are given is its high dimensionality. Two-dimensional $p \times q$ grayscale images span a $m=pq$ -dimensional vector space, so an image with 100×100 pixels lies in a 10,000-dimensional image space already. The question is: Are all dimensions equally useful for us? We can only make a decision if there's any variance in data, so what we are looking for are the components that account for most of the information. The Principal Component Analysis (PCA) was independently proposed by Karl Pearson (1901) and Harold Hotelling (1933) to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

The Eigenfaces method then performs face recognition by:

- Projecting all training samples into the PCA subspace.
- Projecting the query image into the PCA subspace.
- Finding the nearest neighbor between the projected training images and the projected query image.

Still there's one problem left to solve. Imagine we are given 400 images sized 100×100 pixel. The Principal Component Analysis solves the covariance matrix $S=XX^T$, where $\text{size}(X)=10000 \times 400$ in our example. You would end up with a 10000×10000 matrix, roughly 0.8GB. Solving this problem isn't feasible, so we'll need to apply a trick. From your linear algebra lessons you know that a $M \times N$ matrix with $M > N$ can only have $N-1$ non-zero eigenvalues. The resulting eigenvectors are orthogonal, to get orthonormal eigenvectors they need to be normalized to unit length. I don't want to turn this into a publication, so please look into [this](#) for the derivation and proof of the equations.

3.12.3 Fisherfaces in OpenCV

The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information may be lost when throwing components away. Imagine a situation where the variance in your data is generated by an external source, let it be the light. The components identified by a PCA do not necessarily contain any discriminative information at all, so the projected samples are smeared together and a classification becomes impossible

The Linear Discriminant Analysis performs a class-specific dimensionality reduction and was invented by the great statistician He successfully used it for classifying flowers in his 1936 paper The use of multiple measurements in taxonomic problems In order to find the combination of features that separates best between classes the Linear Discriminant Analysis maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation. This was also recognized by Belhumeur, Hespanha and Kriegman and so they applied a Discriminant Analysis to face recognition .

3.13 Fire detection

Recent advancements in embedded processing have allowed vision-based systems to detect fire using Convolutional Neural Networks during surveillance. In this, we have implemented two custom CNN models for a cost-effective fire detection CNN architecture for surveillance videos. The first model is a customized basic CNN architecture. We will implement and see its output and limitations and create a customized InceptionV3 model. To balance the efficiency and accuracy, the model is fine-tuned considering the target problem and fire data. We are going to use three different datasets for training our models.

3.13.1 Creating the customized CNN architecture

We are going to use TensorFlow API Keras for building our model. Let's first create our ImageDataGenerator for labeling our data. [1] and [2] datasets are used here. Finally, we will have 980 images for training and 239 images for validation. We are going to use data augmentation as well. Now, we will create our CNN model. The model contains three Conv2D-MaxPooling2D layer pairs followed by 3 Dense layers. To overcome the problem of over-fitting, we will also add dropout layers. The last layer is the Softmax layer, which will give us the probability distribution for both the classes — Fire and Nonfire.

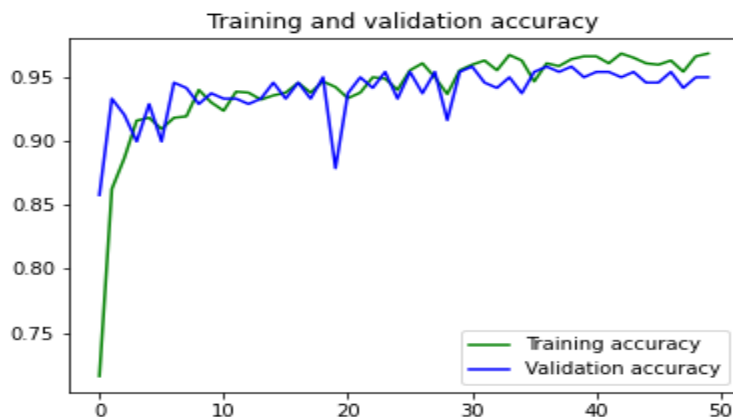


Fig 3.13.1 training and validation accuracy



Fig 3.13.1 training and validation loss

3.13.2 Creating customized InceptionV3 model

We will use a different dataset this time, the one which contains outdoor and indoor fire images. I have trained our previous CNN model in this dataset and the result was that it over-fitted, as it could not handle this comparatively larger dataset and learn complex features from the images.

Let's start with creating the ImageDataGenerator for our customized InceptionV3. The dataset contains 3 classes, but for this article, we will only use 2 classes. It contains 1800 images for training and 200 images for validation. Also, I added 8 images of my living room to add some noise to the dataset.

To make training even more accurate we can use data augmentation techniques. In the above code, 2 data augmentation techniques are applied — horizontal flipping and zooming.

Let's import our InceptionV3 model from the Keras API. We will add our layers at the top of the InceptionV3 model as shown below. We will add a global spatial average pooling layer followed by 2 dense layers and 2 dropout layers to ensure that our model does not overfit. At last, we will add a softmax activated dense layer for 2 classes.

Next, we will first train only the layers that we added and are randomly initialized. We will use RMSprop as an optimizer here.

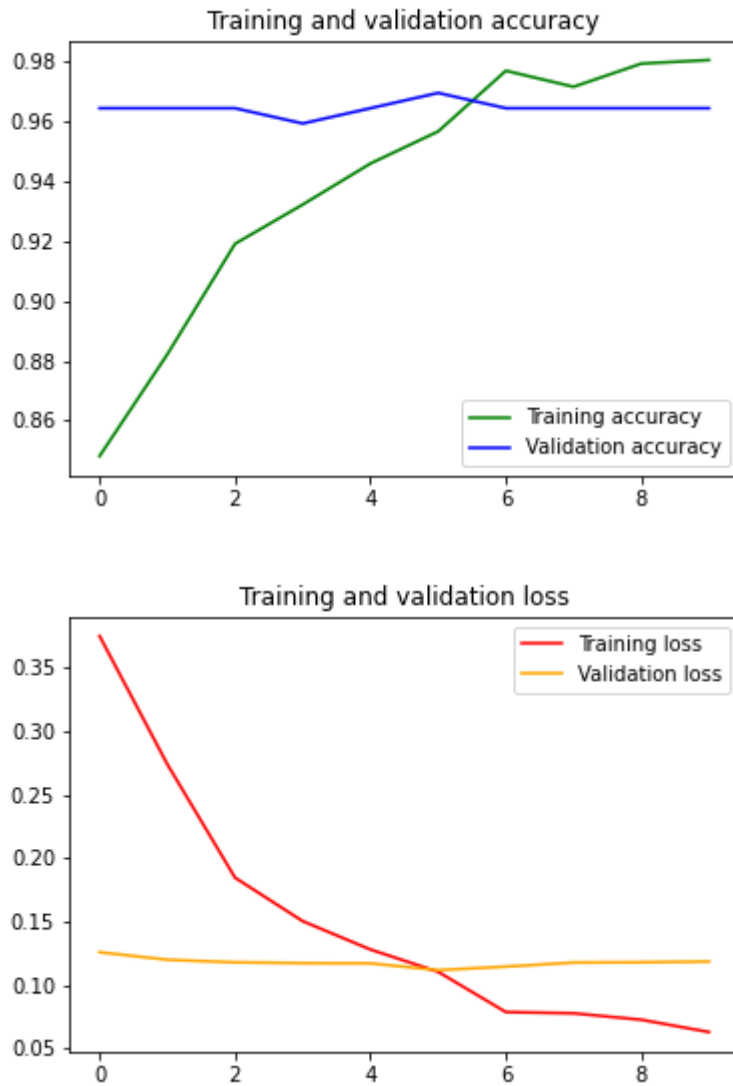


Fig 13.13.2 Creating customized InceptionV3 model

3.13.3 Real-time testing

Now, our model is ready to be tested with real scenarios. Below is the sample code for using OpenCV to access our webcam and predicting whether each frame contains fire or not. If a frame contains fire in it, we want to change the color of that frame to B&W. Using smart cameras you can identify various suspicious incidents such as collisions,

medical emergencies, and fires. Of such, fire is the most dangerous abnormal occurrence, because failure to control it at an early stage can lead to huge disasters, leading to human, ecological and economic losses. Inspired by the great potential of CNNs, we can detect fire from images or videos at an early stage. This article shows two custom models for fire detection. Considering the fair fire detection accuracy of the CNN model, it can be of assistance to disaster management teams in managing fire disasters on time, thus preventing huge losses.

CHAPTER 4

METHODOLOGY

4.1 SYSTEM REQUIREMENTS:

4.1.1 Software Requirements:

Software requirements deals with defining resources requirements and prerequisites that needs to be installed on a computer to provide functioning of an application the minimal software requirements are as follows,

- Front end : python
- IDE : Python 3.7
- OS :Windows 10,7.

4.1.2 Hardware Requirements:

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The minimal hardware requirements are as follows.

- Max 5mp camera

4.2 ARCHITECTURE DIAGRAM

An architectural diagram is a diagram of a system that is used to abstract the overall outline of the software system and the relationships, constraints, and boundaries between components. It is an important tool as it provides an overall view of the physical deployment of the software system and its evolution roadmap.

4.2.1 System Architecture:

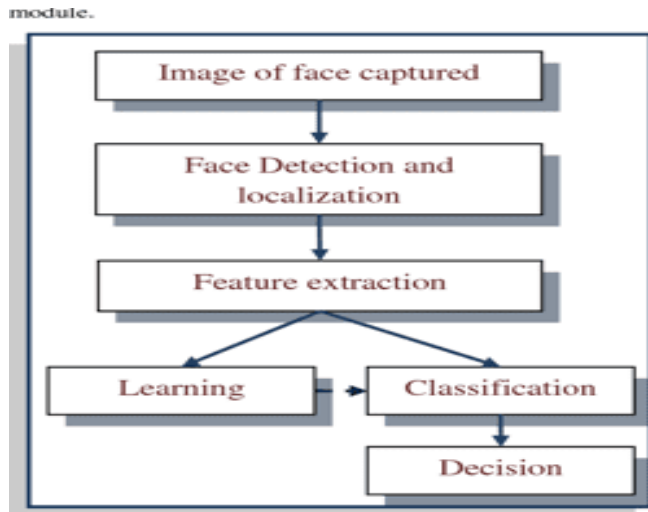


Fig. 1: Basic architecture of an artificial face recognition

Fig 4.2.1 architecture diagram

4.3 MODULES DESCRIPTION

Monitor :

As effective as having a manned security guard on-site—when threats are spotted, they are dealt with Track every move the potential intruder or criminal makes. You'll miss less and convict more. Peace of mind. An online portal is installed to see what the external monitoring station—peace of mind knowing the job took actions is being done correctly The operators then use pan and zoom cameras to look at the alarm's cause. The operator will take appropriate action if suspicious activity is detected. They will contact the emergency services and your site's key holder.

In-out :

Alarm inputs are for sensors like motion detectors and magnets even allow the lens to be zoomed closer or farther. It takes the snap whoever enters which is detected by In/Out module and saves it into our local PC

Recording :

The record module is used to record the video, and it will save the recorded video on a local PC. It will record and able to store the particular data until it restore for the new recording

Fire detection :

The feature detects the fire and alerts the user by a sound alarm in the next few minutes. It detects the fire occurrence, one or more detectors will operate. This action closes the circuit which the fire control panel recognizes as an emergency condition. The panel will then activate one or more signaling circuits to sound building alarms and summon emergency help. The panel may also signal another alarm panel so that it can be monitored from a remote point.

Rectangle:

It is the feature used to detect the motion in a particular portion of the area. The user can choose the portion if there is any motion in that portion of it gives the signal to the admin. And also it detects and captures the motion and saves the image on the local PC.

4.4 ANALYSING AND ACCESSING DATA

1. Training the Model

```
Last login: Tue Feb 23 07:44:47 on ttys000
akashnandagopal@Akashs-MacBook-Pro ~ % cd Desktop
akashnandagopal@Akashs-MacBook-Pro Desktop % cd pro
akashnandagopal@Akashs-MacBook-Pro pro % python3 train.py
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/tensorflow/python/compat/v1_orflow/python_ops_variable_scope.py:110: tf.nn.conv2d is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/tensorflow/python/util/dispatch.py:100: ops.nn_ops with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use 'rate' instead of 'keep_prob'. Rate should be set to 'rate = 1 - keep_prob'.
2021-02-23 07:46:16.288571: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI
owing CPU instructions in performance-critical operations: AVX2 FMA
To enable then in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-02-23 07:46:16.991321: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:258] None of the MLIR optimization passes
Epoch: 0, Step: 0, Loss: 10.2352
Epoch: 0, Step: 10, Loss: 6.68868
Epoch: 0, Step: 20, Loss: 6.57787
Epoch: 0, Step: 30, Loss: 6.58564
Epoch: 0, Step: 40, Loss: 6.21181
Epoch: 0, Step: 50, Loss: 6.36095
Epoch: 0, Step: 60, Loss: 6.7569
Epoch: 0, Step: 70, Loss: 6.18133
Epoch: 0, Step: 80, Loss: 6.87573
Epoch: 0, Step: 90, Loss: 5.98587
Epoch: 0, Step: 100, Loss: 6.07628
Epoch: 0, Step: 110, Loss: 5.92587
Epoch: 0, Step: 120, Loss: 6.83154
Epoch: 0, Step: 130, Loss: 5.98298
Epoch: 0, Step: 140, Loss: 5.74228
Epoch: 0, Step: 150, Loss: 5.66672
Epoch: 0, Step: 160, Loss: 5.56228
Epoch: 0, Step: 170, Loss: 5.66396
Epoch: 0, Step: 180, Loss: 5.64383
```

Fig. 5. Training the Model (At beginning)

```
Epoch: 29, Step: 3120, Loss: 0.143451
Epoch: 29, Step: 3130, Loss: 0.224523
Epoch: 29, Step: 3140, Loss: 0.141103
Epoch: 29, Step: 3150, Loss: 0.138307
Epoch: 29, Step: 3160, Loss: 0.141191
Epoch: 29, Step: 3170, Loss: 0.145803
Epoch: 29, Step: 3180, Loss: 0.151027
Epoch: 29, Step: 3190, Loss: 0.14036
Epoch: 29, Step: 3200, Loss: 0.143089
Epoch: 29, Step: 3210, Loss: 0.155034
Epoch: 29, Step: 3220, Loss: 0.14069
Epoch: 29, Step: 3230, Loss: 0.144465
Epoch: 29, Step: 3240, Loss: 0.172037
Epoch: 29, Step: 3250, Loss: 0.142371
Epoch: 29, Step: 3260, Loss: 0.178216
Epoch: 29, Step: 3270, Loss: 0.138012
Epoch: 29, Step: 3280, Loss: 0.143052
Epoch: 29, Step: 3290, Loss: 0.135708
Epoch: 29, Step: 3300, Loss: 0.21534
Epoch: 29, Step: 3310, Loss: 0.141223
Epoch: 29, Step: 3320, Loss: 0.141223
Epoch: 29, Step: 3330, Loss: 0.137741
Epoch: 29, Step: 3340, Loss: 0.134547
Epoch: 29, Step: 3350, Loss: 0.142568
Model saved in file: ./save/model.ckpt
Run the command line:
--> tensorboard --logdir=./logs
Then open http://0.0.0.0:6006/ into your web browser
akashnandagopal@Akashs-MacBook-Pro pro %
```

Fig. 6. Training the Model (At end)

In fig. 5, It starts training the model and we can see the training accuracy value. The loss values in higher in the beginning of training the model, which means the accuracy is lesser.

In fig. 6, the loss value is reduced which means the accuracy is increased after repeating the process.

The process is repeated till 30 epoch and finally the model gets trained and the model is stored at the following directory. After training we can visualize the graph using tensorboard Fig. 7 .

2. GraphVisualization

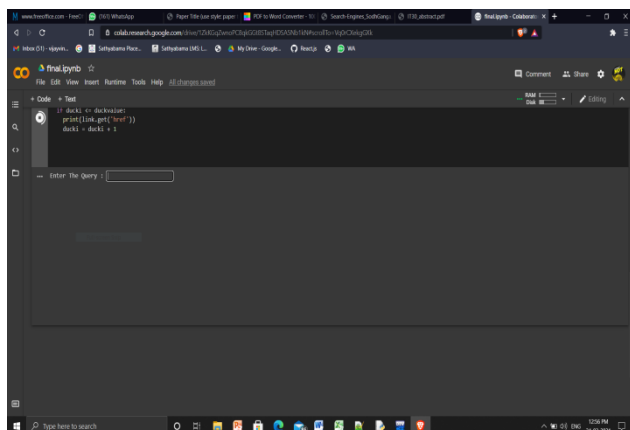


Fig. 7.Tensorboard:TensorFlow’s visualization toolkit

So in order to get good recognition rates you'll need at least 8(+1) images for each person and the Fisherfaces method doesn't really help here. The above experiment is a 10-fold cross validated result carried out with the facerec framework at: <https://github.com/bytewidth/facerec>. This is not a publication, so I won't back these figures with a deep mathematical analysis. Please have a look into [141] for a detailed analysis of both methods, when it comes to small training datasets.

3. Running the model through dataset

The trained network is used to generate steering commands from a single front-facing camera.

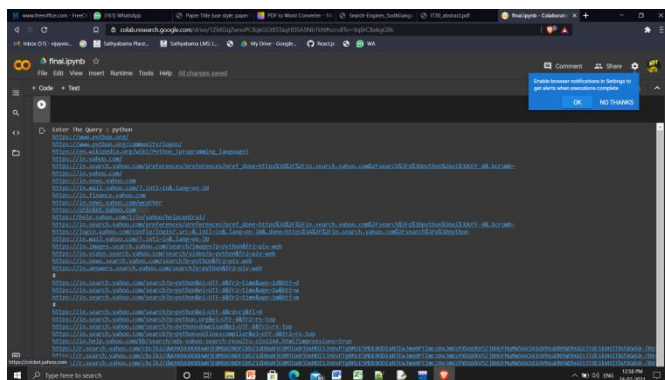


Fig. 8. Running the model through data set

4. Running the model through live web camfeed

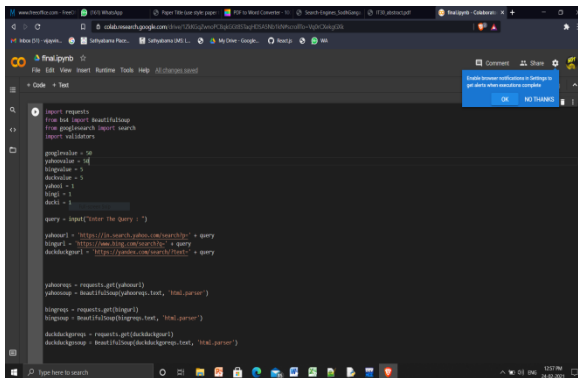


Fig. 9. Running the model through live web cam feed

CHAPTER 5

RESULT AND DISCUSSION

RESULT

The project is done with the GUI interface for the user so that the user can interact with the application easier. The model is getting trained with an open cv module in machine learning and deploy into the model By using the Machine Learning methods we deployed several modules to make the CCTV system smarter. The modules are the CCTV system should take a snap whenever motion arises on screen. We also design the model to monitor a particular portion of the screen and other basic functions.

DISCUSSION

Smart CCTV systems have the potential to play a substantial role in the management of urban Space through intelligence-led policing. In applying. We must devote this smart technology to attention to associated risks, such as the potential for false alarms and privacy breaches.Smart CCTV itself has the potential to help generate this evidence base of urban human behavior through a systematic program of research preceding the rollout of the technology.Smart CCTV shows a substantial increase in the power and sensitivity of traditional CCTV technology and the potential to transform both the management and design of urban space.This power needs to be guided by a technological vision of city-wide data management within a sophisticated legal framework cognisant of the culturally specificnature

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

CONCLUSION

Smart cameras are increasingly becoming a household essential. More than capturing images and recording footage, these devices come with additional features like face tag, motion detection, GPS, communication, real-time viewing, and cloud storage to ensure that you get the most out of your security system. Face recognition is an effortless task for humans. Experiments have shown that even one to three-day-old babies can distinguish between known faces. The query module process all queries made by users by retrieving web information stored in indexes

We have carried a series of experiments out on the proposed smart home. These experiments show how to detect fire and motion. Also, how to detect any intruder

FUTURE WORK

There are a variety of enhancements that could be made to this system to achieve greater accuracy in sensing and detection. There are a lot of other sensors that can increase the security and control of the home, like a pressure sensor that can be put outside the home to detect that someone will enter the home. Changing the way of automated notifications by using the GSM module to make this system more professional.

If we placed many cameras in a different location each of the cameras in the certain limits mapped in the system, the user will know the exact location from where the images were taken. with the up-gradation of the existing CCTV network with the new software, we intend to monitor mask

REFERENCE

- [1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back propagation applied to handwritten zip code recognition. *NeuralComputation*,1(4):541–551.
- [2] Alex Krizhevsky, IlyaSutskever, and Geoffrey E. Hinton. Imagenet classification by deep convolutional neural networks. KuF. Pereira, C.J.
C. Burges, L. Bottou, noK. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097-1105. Curran Associates, Inc.
- [3] Learning Compact Binary Face Descriptor for Face Recognition, Jiwen Lu Department of Automation, Tsinghua University, Beijing, China
- [4] Multi-Task Pose-Invariant Face Recognition, Changxing Ding
Centre for Quantum Computation and Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology at Sydney, Sydney, NSW,
- [5] APA: Adaptive Pose Alignment for Pose-Invariant Face Recognition, Zhanfu An Pattern Recognition and Intelligent System Laboratory, Beijing University of Posts and Telecommunications, Beijing, China
- [6] Tuning up the search engine, Joseph Williamsis Practice Manager, Americas, for Sun Microsystems' Advanced Internet Practice group. The information presented here represents the author's opinions and not necessarily those of Sun Microsystems Inc. Contact Williams at joseph.williams@sun.com.
- [7] Learning Compact Binary Face Descriptor for Face Recognition, Jiwen Lu
Department of Automation, Tsinghua University, Beijing, China
- [8] Multi-Task Pose-Invariant Face Recognition, Changxing Ding
Centre for Quantum Computation and Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology at Sydney, Sydney, NSW
- [9] APA: Adaptive Pose Alignment for Pose-Invariant Face Recognition, Zhanfu An
Pattern Recognition and Intelligent System Laboratory, Beijing University of Posts and Telecommunications, Beijing, China

APPENDIX

A.SOUCE CODE

```
import tkinter as tk
import tkinter.font as font
from motion import in_out
from motion import noise
from motion import fire

from rect_noise import rect_noise
from record import record
from PIL import Image, ImageTk

window = tk.Tk()
window.title("Smart cctv")
window.iconphoto(False, tk.PhotoImage(file='mn.png'))
window.geometry('1080x760')

frame1 = tk.Frame(window)

label_title = tk.Label(frame1, text="Smart cctv Camera")
label_font = font.Font(size=35, weight='bold', family='Helvetica')
label_title['font'] = label_font
label_title.grid(pady=(10,10), column=2)

icon = Image.open('icons/spy.png')
icon = icon.resize((150,150), Image.ANTIALIAS)
icon = ImageTk.PhotoImage(icon)
label_icon = tk.Label(frame1, image=icon)
label_icon.grid(row=1, pady=(5,10), column=2)

btn1_image = Image.open('icons/lamp.png')
btn1_image = btn1_image.resize((50,50), Image.ANTIALIAS)
btn1_image = ImageTk.PhotoImage(btn1_image)

btn2_image = Image.open('icons/rectangle-of-cutted-line-geometrical-shape.png')
btn2_image = btn2_image.resize((50,50), Image.ANTIALIAS)
btn2_image = ImageTk.PhotoImage(btn2_image)

btn5_image = Image.open('icons/exit.png')
btn5_image = btn5_image.resize((50,50), Image.ANTIALIAS)
btn5_image = ImageTk.PhotoImage(btn5_image)

btn3_image = Image.open('icons/security-camera.png')
btn3_image = btn3_image.resize((50,50), Image.ANTIALIAS)
btn3_image = ImageTk.PhotoImage(btn3_image)

btn6_image = Image.open('icons/incognito.png')
btn6_image = btn6_image.resize((50,50), Image.ANTIALIAS)
```

```

btn6_image = ImageTk.PhotoImage(btn6_image)

btn4_image = Image.open('icons/recording.png')
btn4_image = btn4_image.resize((50,50), Image.ANTIALIAS)
btn4_image = ImageTk.PhotoImage(btn4_image)

# ----- Button -----#
btn_font = font.Font(size=25)
btn1 = tk.Button(frame1, text='Monitor', height=90, width=180, fg='green', command=noise,
image=btn1_image, compound='left')
btn1['font'] = btn_font
btn1.grid(row=3, pady=(20,10))

btn2 = tk.Button(frame1, text='Rectangle', height=90, width=180, fg='orange',
command=rect_noise, compound='left', image=btn2_image)
btn2['font'] = btn_font
btn2.grid(row=3, pady=(20,10), column=3, padx=(20,5))

btn_font = font.Font(size=25)
btn3 = tk.Button(frame1, text='Fire Detection', height=90, width=180, fg='green',
command=fire, image=btn3_image, compound='left')
btn3['font'] = btn_font
btn3.grid(row=5, pady=(20,10))

btn4 = tk.Button(frame1, text='Record', height=90, width=180, fg='orange',
command=record, image=btn4_image, compound='left')
btn4['font'] = btn_font
btn4.grid(row=5, pady=(20,10), column=3)

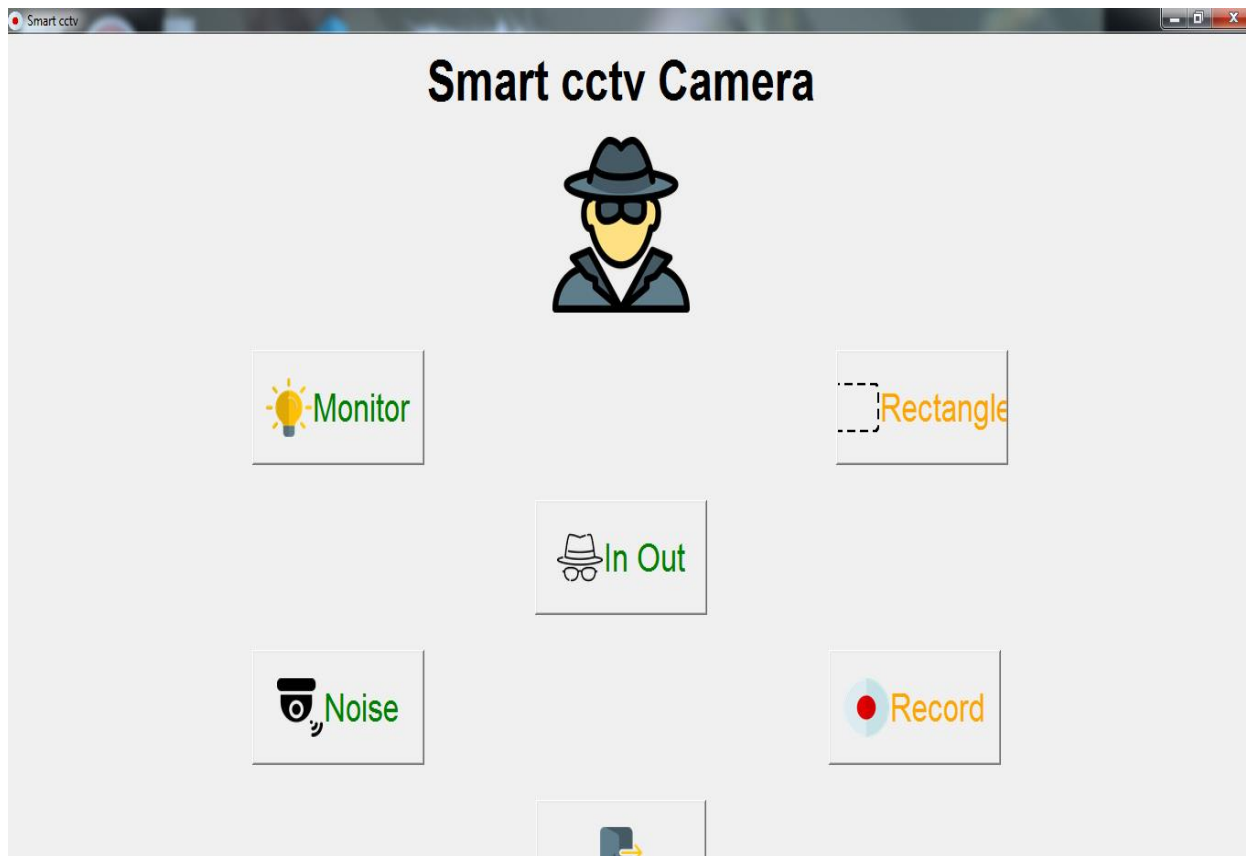
btn6 = tk.Button(frame1, text='In Out', height=90, width=180, fg='green', command=in_out,
image=btn6_image, compound='left')
btn6['font'] = btn_font
btn6.grid(row=4, pady=(20,10), column=2)

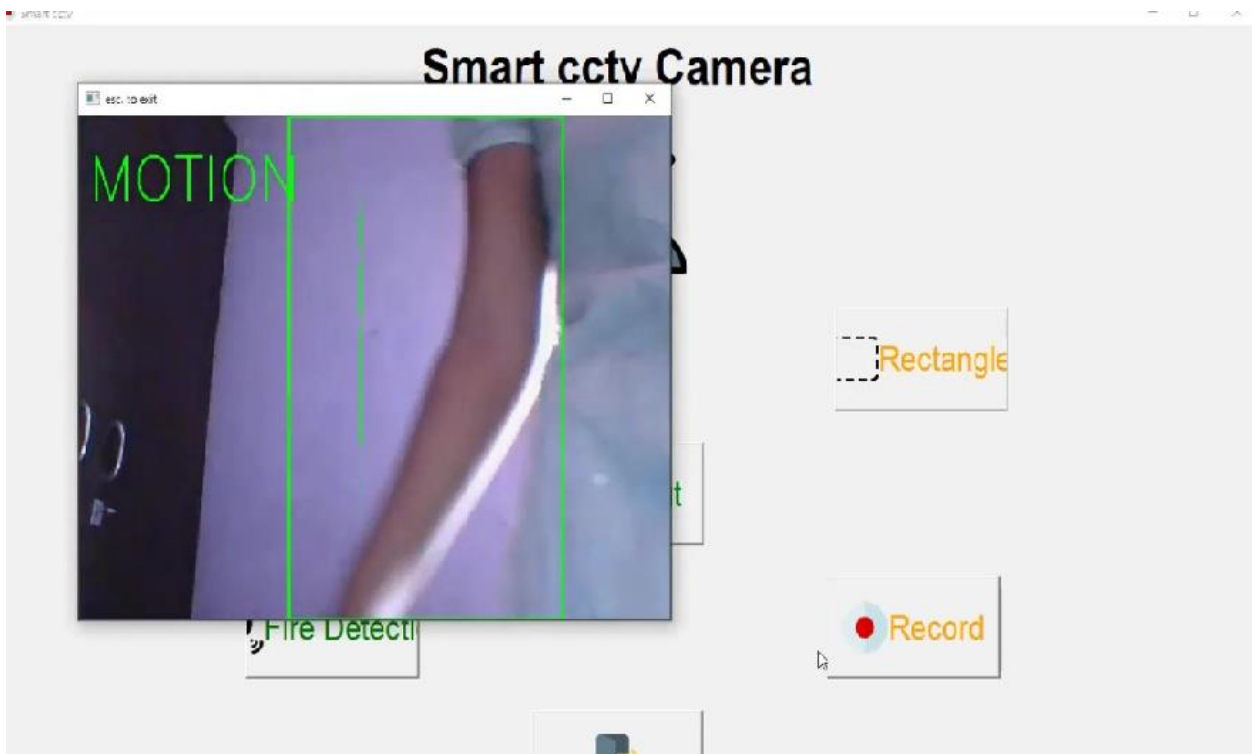
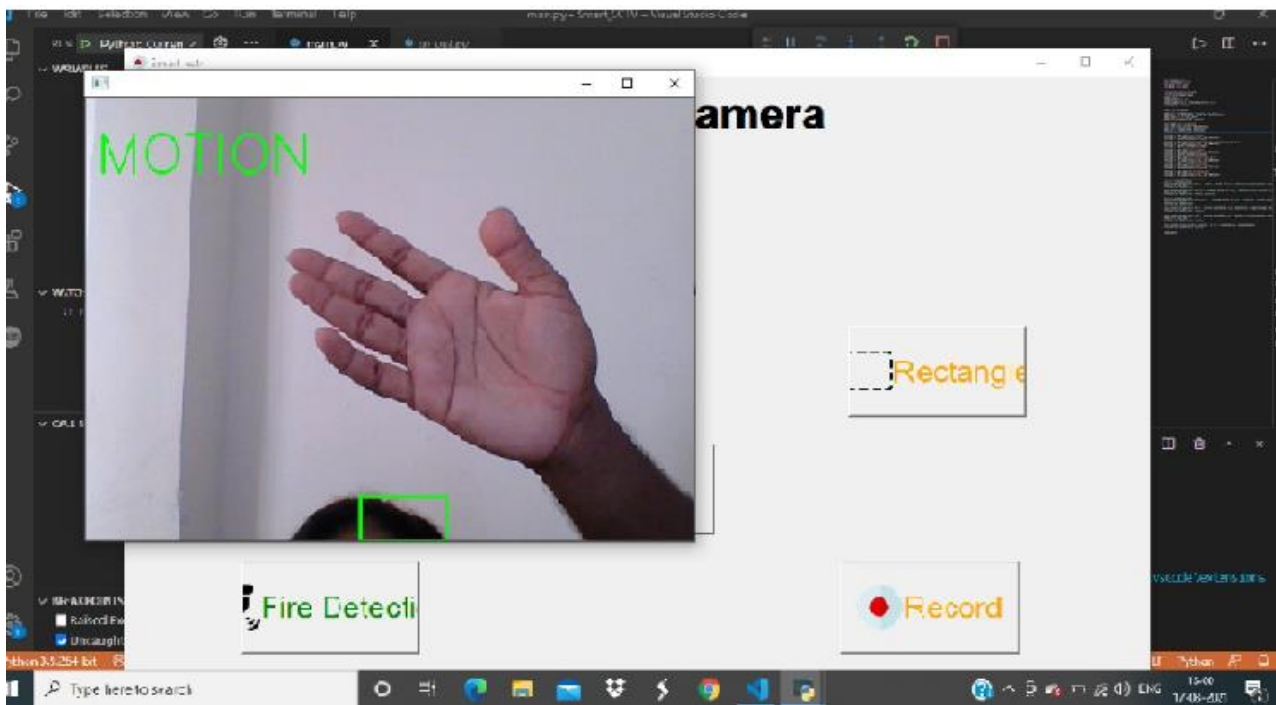
btn5 = tk.Button(frame1, height=90, width=180, fg='red', command=window.quit,
image=btn5_image)
btn5['font'] = btn_font
btn5.grid(row=6, pady=(20,10), column=2)

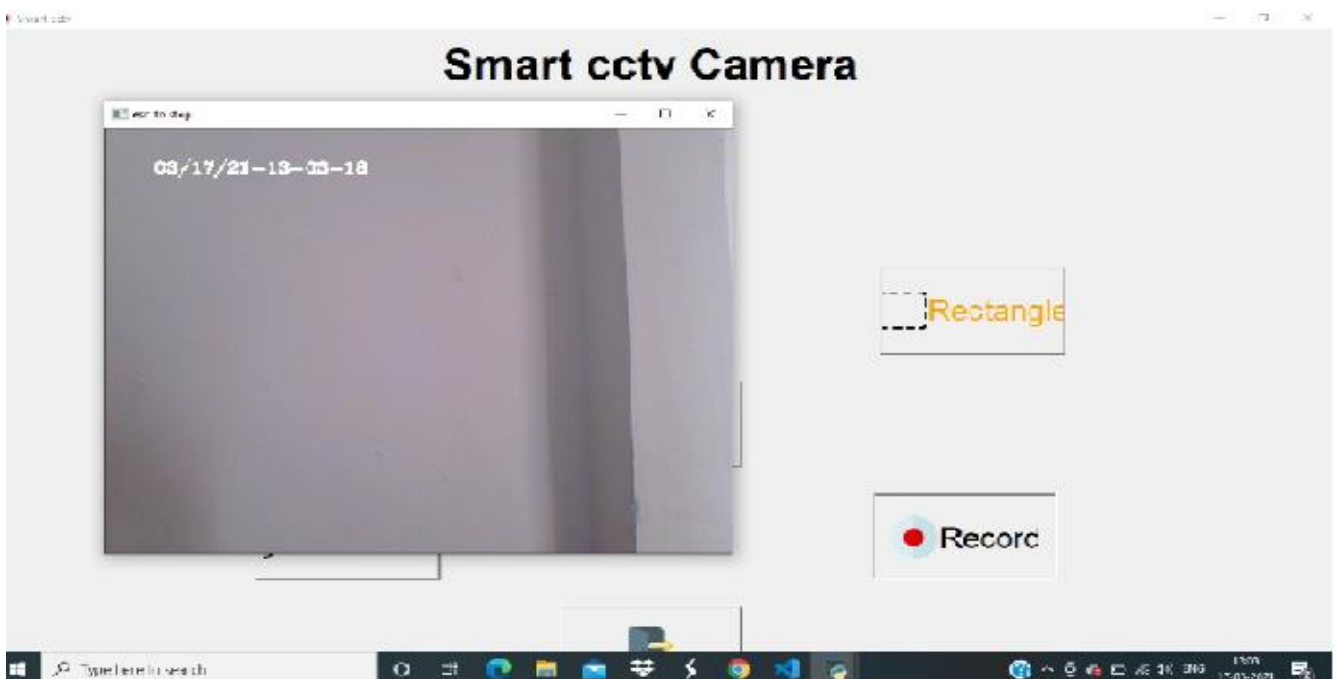
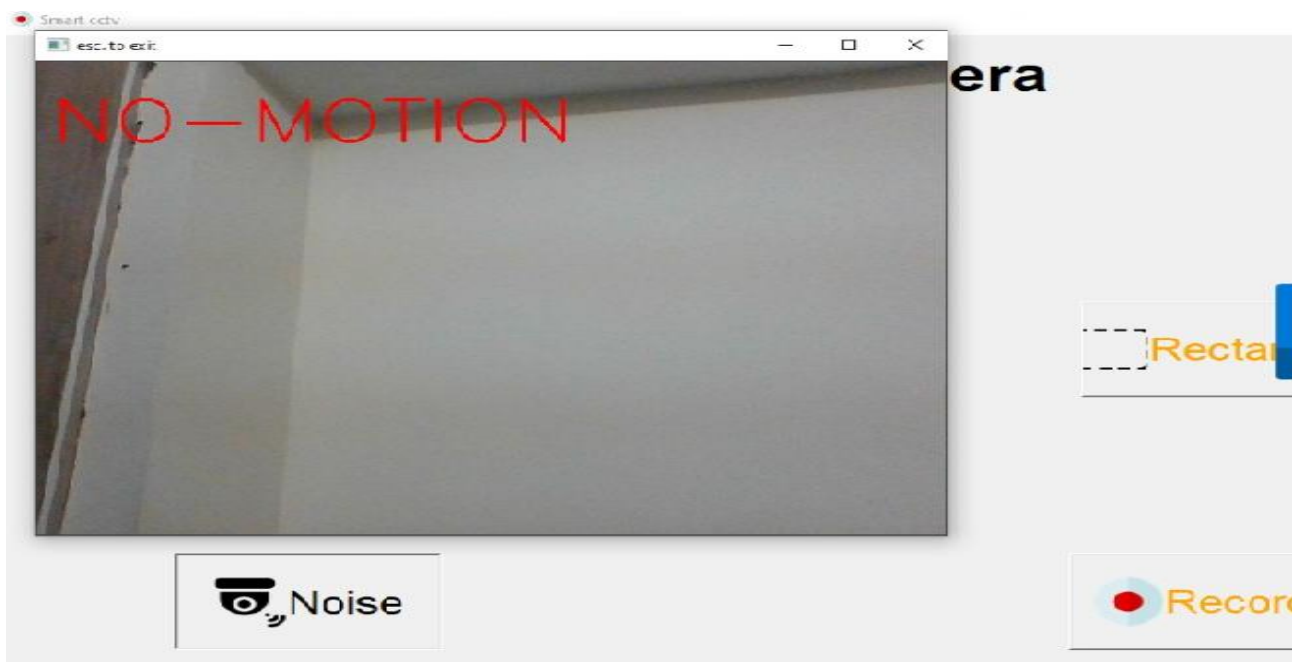
frame1.pack()
window.mainloop()

```

SCREENSHOTS







Smart cctv by machine learning

Miss.Sneha¹, Miss. Vedhanayaki², Ms. Vimali J.S³

^{1,2}U.G Scholar, Department of Information Technology

³Assistant Professor, Department of Information Technology

Sathyabama Institute of Science and Technology, Chennai, India

¹snehapeter9007@gmail.com, ³Vimalijs.bala@gmail.com, ²vedharaichandran238@gmail.com

Abstract : CCTV systems are becoming more common these days because they can control what is going on at the place where they will install. In terms of safety and protection, tracking the behavior of staff and visitors at our place is necessary. CCTV services play a vital role in all forms of security problems. Installing a dependable CCTV camera is a practical move, and a surveillance package provides you with a proactive security advantage. If you own a company or just want to protect your family. Routine inspections and repairs must well maintain. Traditional CCTV devices can only constantly record the world; they cannot perform smart functions such as motion detection.

Keywords : OpenCV, Smart CCTV, Python.

I. INTRODUCTION

Gary Bradsky began working on OpenCV in 1999 at Intel, and the first version was released in the year 2000. Vadim Pisarevsky has joined Gary Bradsky as the manager of Intel's Russian OpenCV tech unit. The 2005 DARPA Grand Challenge winning vehicle, Stanley, used OpenCV in 2005. Later, Willow Garage supported the project's active growth, which was led by Vadim Pisarevsky and Gary Bradsky. OpenCV currently supports a wide range of algorithms in the fields of computer vision and machine learning, and it continues to develop. OpenCV currently supports several programming languages, including It runs on a number of platforms, including Windows, Linux, OS X, Android, and iOS, and is written in C++, Python, and Java. GPU activities at fast speeds, interfaces based on CUDA and Open Cl are also being actively developed. OpenCV's Python API is known as OpenCV-Python. It incorporates the best features of the Python programming language and the OpenCV C++ API

I. LITERATURE SURVEY

Y. LeCun [1] Providing questions from the workplace will significantly improve the opportunity to learn how to do better. This paper shows how network architecture can incorporate such problems into a back-end network. One learns the entire function of recognition, from the character's typical picture to the last level.

Alex Krizhevsky [2] A neural network with deep convolution In the ImageNet LSVRC-2010 competition, there were 1.2 million image splits into 1000 different categories. We found 37.5 percent of the top-1 and top-5 error scores in the experimental results, as well as 17.0 percent, which is significantly better than the previous case. The neural network has 650,000 connections and 60 million parameters.

There are five distinct layers in neurons, some of which are accompanied by integration layers, and three entirely connected layers with a final Softmax of 1000. We used non-complementary neurons and GPU-efficient convolution functions to speed up preparation. We used a newly developed "dropout" strategy to minimize overcrowding in fully integrated layers, which has proven to be very successful. In the ILSVRC-2012 competition, we included variants and found a high error rate of 15.3 percent, compared to 26.2 percent with the best entry. INet Scale Technologies Inc. [3] Guido van Rossum created Python, a general-purpose programming language. that has rapidly gained popularity due to its ease of use and It enables the programmer to represent himself in fewer lines of code while still keeping readability. Python is slower than C/C++ and other programming languages. Python can be conveniently expanded with C/C++, which is an advantageous function We may use this feature to bundle computationally intensive C/C++ code in a Python wrapper. and use it as a Python module. This helps us in two ways: first, our code is as swift as the original C/C++ code (due to the background C++ code), and second, Python is incredibly simple to code. OpenCV-Python is a Python extension for the original C++ implementation.

II. METHODOLOGY

In Haar feature-based cascade classifiers, machine learning is used to train a cascade function from many positive and negative images. After that, it's used to find things in other pictures. The benefit of Haar feature-based cascade classifiers is that you can build one for any object you like, and OpenCV has already given some classifier parameters, so you don't need to collect any data to train on them. A green pixel map from the front-facing camera to the driving instructions is generated by a Neural Network (CNN). This successful end-to-end approach ensures that the system learns to guide on both local and highways, with or without a linear curve, using only low-level human training. We may also employ the technique in situations with few hints. For example, parking lots and unpaved roads.

Using only a personal directional angle as a teaching signal, The computer is designed to learn the internal representation of the necessary processing steps, such as locating helpful road features, automatically. We don't have to specifically train it to find a roadmap, for example.

End-of-life learning improves efficiency and reduces curriculum size. Better performance is achieved when

internal components work together to improve the overall system's performance rather than enhancing man-made intermediate choices, such as route discovery. Such methods are theoretically favoured to quickly translate people, but they do not automatically validate the system's operation. Smaller networks are feasible when the computer learns to solve problems with a minimal number of processing moves.

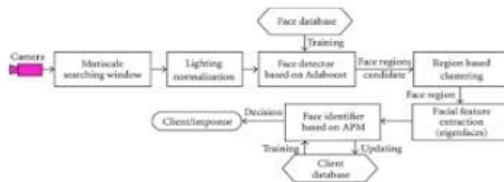


Fig. 1. OpenCV Architecture

Sobel, Scharr, Laplacian, and Canny-edge detectors are among the many edge detection filters available. We'll use a Laplacian edge filter because it produces edges that resemble hand sketches and is more consistent than a Canny-edge detector, which produces very tidy line drawings but is more influenced by random noise in the camera frames, causing the line drawings to differ considerably between frames. A solid bilateral filter smooths flat regions while keeping sharp edges. Except that it is incredibly slow (measured in seconds or even minutes rather than milliseconds!), rendering it ideal for use as an automatic cartoonifier or painting filter. As a result, we'll use a few techniques to build a smooth-running cartoonifier. The most powerful trick we have is to use Filtering in all directions at a lower resolution. It would have the same result as maximum resolution, but it will happen even sooner. Let's halve the cumulative number of pixels (half width and half height, for example). The bilateral filter is regulated by four variables: colour strength, positional capacity, scale, and repetition count. We'll need a temporary Mat so that bilateral Filter can work () doesn't corrupt the data (this is known as "in-place processing"), and we can use one filter to store the temp Mat and another to store the information. The previous coding for grayscale conversion of the original image and inserting a 7 x 7 Median filter can be reused because we'll be dealing with a grayscale image for noise reduction.

With only a personal directional angle as a training signal, The device is designed to learn the internal representation of the necessary processing steps, such as identifying useful road features, on its own. We don't have to specifically train it to find a roadmap, for example.

End-of-life learning improves efficiency and reduces curriculum size. Better performance is achieved when internal components work together to improve the overall system's performance rather than enhancing man-made intermediate choices, such as route discovery. Such methods are theoretically favoured to quickly translate people, but they do not automatically validate the system's operation. Smaller networks are feasible when the machine learns to solve problems with a minimal number of processing moves.

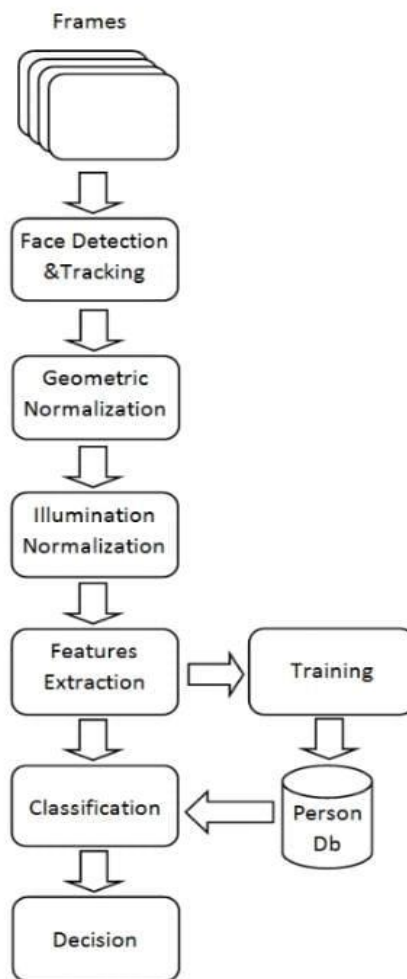


Fig. 2. Face Detection Algorithm

Simple colour thresholds dependent on HSV (Hue-Saturation-Brightness) or RGB (Red-Green-Blue) values, as well as colour histogram estimation and re-projection, as well as advanced Mixture model machine-learning algorithms including CIE Lab colour space camera tuning and offline preparation for multiple sample faces, are used to detect skin regions. Also the most sophisticated approaches, however, do not always work effectively in a variety of camera and lighting conditions, as well as for different skin types. We can use a basic skin-detection system because We want our skin detection to work without any calibration or planning on a mobile device, and we're just using it as a "fun" image filter. However, since the colour response of mobile device camera sensors varies significantly, we need something more powerful than basic colour thresholds to identify skin for people of any skin colour without calibration. The AT&T Face collection, also known as the ORL Faces Database, features ten separate photographs of each of the 40 subjects. The photographs of those participants were captured at different times of the day, with different lighting, facial expressions (open vs. closed eyes, smile vs.

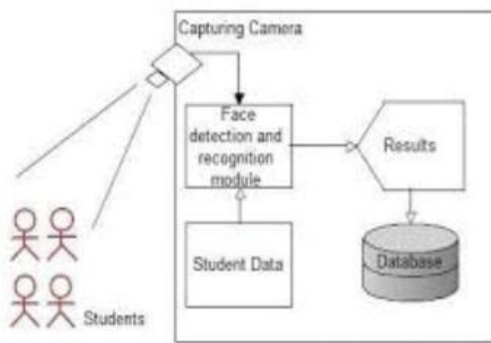


Fig. 3. Block Diagram of face recognition

Fisherfaces and Eigenfaces use a more An all-encompassing approach to facial recognition. In high-dimensional image space, the data is viewed as a vector. Since high dimensionality is negative, a lower-dimensional subspace is located where (hopefully) valuable knowledge can be retained. Where the variation is caused from an external source, the Eigenfaces solution maximises overall scatter, which can cause issues. We used a Fisherfaces process and optimised it with To hold any discriminative results, use a Linear Discriminant Analysis. At least in our model's constrained case, the Fisherfaces method performed admirably.

Now, life isn't always fine. You obviously cannot guarantee perfect lighting or ten separate images of an individual in your images. So, what if each individual only has one image? Our covariance estimates for the subspace, as well as the identification, could be terribly incorrect. On the AT&T Facedatabase, the Eigenfaces approach had a 96 percent recognition score. How many photographs do you need to provide some precise estimates? The Rank-1 identification the Eigenfaces/rates and Fisher faces methods on AT&T Facedatabase, a quick image database, are as follows:

The query module retrieves site information stored in indexes to process all user queries. Some famous web pages may be retrieved directly from the page repository if their information is not stored in indexes. The rating module mentioned in the following subsection will filter the results shown on the user's computerscreen.

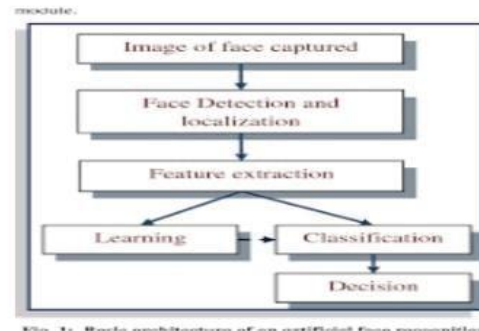


Fig. 4. System Architecture

III. RESULTS AND DISCUSSION

1. Training the Model

[illegible]

Fig. 5. Training the Model (At beginning)

```
Epoch: 29, Step: 3120, Loss: 0.143451
Epoch: 29, Step: 3140, Loss: 0.143623
Epoch: 29, Step: 3160, Loss: 0.141180
Epoch: 29, Step: 3180, Loss: 0.138307
Epoch: 29, Step: 3200, Loss: 0.140029
Epoch: 29, Step: 3179, Loss: 0.145883
Epoch: 29, Step: 3180, Loss: 0.152627
Epoch: 29, Step: 3199, Loss: 0.145829
Epoch: 29, Step: 3200, Loss: 0.143889
Epoch: 29, Step: 3218, Loss: 0.150536
Epoch: 29, Step: 3219, Loss: 0.147629
Epoch: 29, Step: 3239, Loss: 0.144165
Epoch: 29, Step: 3240, Loss: 0.147287
Epoch: 29, Step: 3259, Loss: 0.150812
Epoch: 29, Step: 3276, Loss: 0.150812
Epoch: 29, Step: 3280, Loss: 0.178256
Epoch: 29, Step: 3276, Loss: 0.150812
Epoch: 29, Step: 3299, Loss: 0.150812
Epoch: 29, Step: 3299, Loss: 0.155788
Epoch: 29, Step: 3300, Loss: 0.22526
Epoch: 29, Step: 3319, Loss: 0.150812
Epoch: 29, Step: 3320, Loss: 0.142123
Epoch: 29, Step: 3330, Loss: 0.137541
Epoch: 29, Step: 3349, Loss: 0.137541
Epoch: 29, Step: 3350, Loss: 0.142568
Model saved in file: ./model_checkpoint/weights/epoch_29_3350.h5
Enter the command line:
> tensorboard --logdir=./logs
Then open http://0.0.0.0:6006/ into your web browser
to watch the training process. You can find the
```

Fig. 6. Training the Model (At end)

In fig. 5, It starts training the model and we can see the training accuracy value. The loss values in higher in the beginning of training the model, which means the accuracy is lesser.

In fig. 6, the loss value is reduced which means the accuracy is increased after repeating the process.

The process is repeated till 30 epoch and finally the model gets trained and the model is stored at the following directory. After training we can visualize the graph using tensorboard Fig. 7 .

2. GraphVisualization camfeed

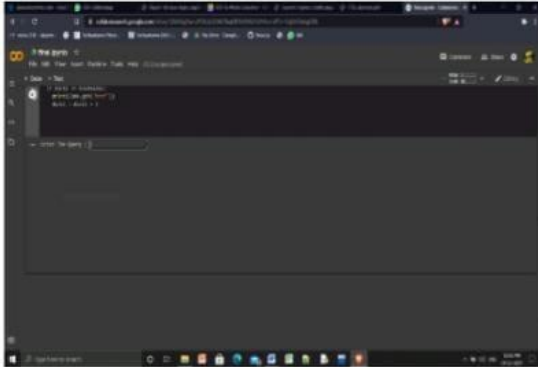


Fig. 7.Tensorboard:TensorFlow's visualization toolkit

As a result, you'll need at least 8(+1) images for each user to get decent recognition rates, and the Fisherfaces approach won't help. The facerec platform, which can be found at <https://github.com/bytedfish/facerec>, was used for the aforementioned experiment. I won't have a thorough mathematical analysis to back up these statements because this isn't a publication. For a comprehensive overview of all methods for small training datasets, see [141].

3. Running the model through dataset

A single front-facing camera is used to produce steering commands using the qualified network.

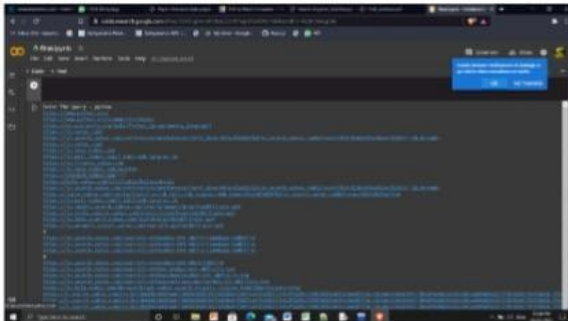


Fig. 8. Running the model through data set

4. Running the model through live web

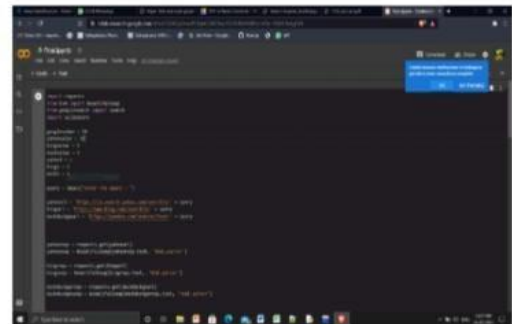


Fig. 9. Running the model through live web cam feed

IV. CONCLUSION

For humans, face recognition is a simple process. Experiments have shown that babies as young as one to three days old can differentiate between familiar faces.

The query module process all queries made by users by retrieving web information stored in indexes

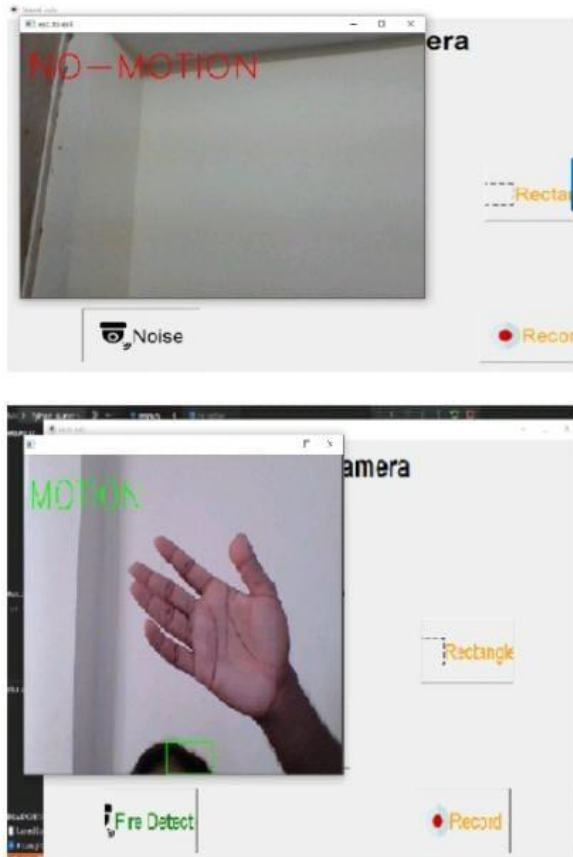
The plan can also apply to areas with vague hints such as good search results.



Monitor :

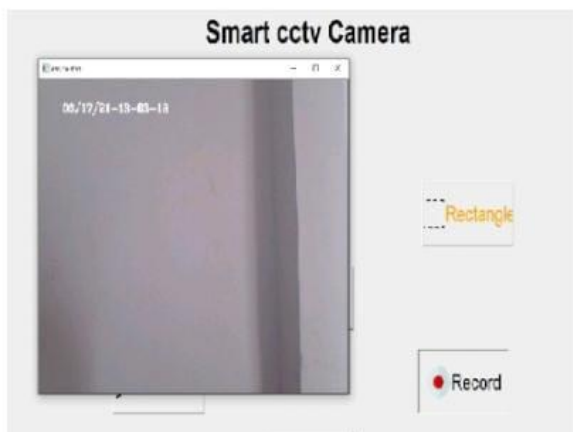
The operators then use pan and zoom cameras to look at the alarm's cause. The operator will take appropriate action if suspicious activity is detected. They will contact the emergency services and your site's key holder.

In-out :



Alarm inputs are for sensors like motion detectors and magnets. It takes the snap whoever enters which is detected by In/Out module and saves it into our local PC

Recording :



The record module is used to record the video, and it will save the recorded video on a local PC.

Fire detection :

The feature detects the fire and alerts the user by a sound alarm in the next few minutes

Rectangle:

It is the feature used to detect the motion in a particular portion of the area. The user can choose the portion, If there is any motion in that particular portion of it gives the signal to the admin

References

- [1]LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Backpropagation applied to handwritten zip code recognition. *Neural computation*. 1989 Dec;1(4):541-51.
- [2]Alex Krizhevsky, IlyaSutskever, and Geoffrey E. Hinton. Imagenet classification by deep convolutional neural networks. KuF. Pereira, C.J.C. Burges, L. Bottou, noK. Q. Weinberger, editors, *Advances in Neural Information Processing Systems* 25, pages 1097-1105. Curran Associates, Inc.
- [3]Learning Compact Binary Face Descriptor for Face Recognition, Jiwen Lu Department of Automation, Tsinghua University, Beijing, China
- [4]Ding, Changxing, Chang Xu, and Dacheng Tao. "Multi-task pose-invariant face recognition." *IEEE Transactions on Image Processing* 24.3 (2015): 980-993.
- [5]APA: Adaptive Pose Alignment for Pose-Invariant Face Recognition, Zhanfu An Pattern Recognition and Intelligent System Laboratory, Beijing University of Posts and Telecommunications, Beijing, China
- [6]Williams, Joseph, and Ravi Starzl. "Tuning up the search engine." *IT Professional* 3.1 (2001): 60-62.
- [7] Sivasangari, A., P. Ajitha, Immanuel Rajkumar, and S. Poonguzhali. "Emotion recognition system for autism disordered people." *Journal of Ambient Intelligence and Humanized Computing* (2019): 1-7.
- [8] Ajitha, P., and G. Gunasekaran. "An approach of opinion mining for online marketing using sentiment thesaurus and concept search engine." In 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), pp. 208-213. IEEE, 2015.
- [9] Sivasangari, A., P. Ajitha, and R. M. Gomathi. "Light weight security scheme in wireless body area sensor network using logistic chaotic scheme." *International Journal of Networking and Virtual Organisations* 22, no. 4 (2020): 433-444.
- [10] Ray, Sourav, and A. Sivasangari. "Design of Mobile Robot Teleportation System using Virtual Reality." In 2020 International Conference on Communication and Signal Processing (ICCCSP), pp. 1173-1175. IEEE, 2020.
- [11] Jinila, Y. B., Rajalakshmi, V., Gladence, L. M., & Anu, V. M. (2020). Food Consumption Monitoring and Tracking in Household Using Smart Container. In *Proceedings of the Third International Conference on Computational Intelligence and Informatics* (pp.

- 693-700). Springer, Singapore.
- [12] Jinila, Y. B., Alam, M. S., & Singh, P. D. (2019). Cloud-Based Scheme for Household Garbage Collection in Urban Areas. In *Advances in Big Data and Cloud Computing* (pp. 539-546). Springer, Singapore.
- [13] Vimali, J. S., Senduru Srinivasulu, J. Jabez, and S. Gowri. "Hand Gesture Recognition Control for Computers Using Arduino." In *Data Intelligence and Cognitive Informatics*, pp. 569-578. Springer, Singapore, 2021.
- [14] Vimali, J. S., S. Srinivasulu, and S. Gowri. "IoT based bank security system." *Int J Recent Technol Eng* (2019): 2324-2327.
- [15] Gowri, S. and Divya, G., 2015, February. Automation of garden tools monitored using mobile application. In *International Confernce on Innovation Information in Computing Technologies* (pp. 1-6). IEEE.
- [16] Akshaya, R., N. Niroshma Raj, and S. Gowri. "Smart Mirror-Digital Magazine for University Implemented Using Raspberry Pi." In *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, pp. 1-4. IEEE, 2018.
- [17] J. Jose and T. Sasipraba, "Indoor air quality monitors using IOT sensors and LPWAN," *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, India, 2019, pp. 633-637
- [18] "An Efficient System to Predict and Analyze Stock Data using Hadoop Techniques", Jithina Jose, SujaCherukullapurathMana, B.KeerthiSamhitha *International Journal of Recent Technology and Engineering (IJRTE)* ISSN: 2277-3878, Volume-8 Issue-2, July 2019

ORIGINALITY REPORT

8%

SIMILARITY INDEX

7%

INTERNET SOURCES

3%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1

www.da6nci.com

Internet Source

2%

2

Submitted to Indian Institute of Technology, Madras

Student Paper

2%

3

es.scribd.com

Internet Source

1%

4

docs.opencv.org

Internet Source

1%

5

link.springer.com

Internet Source

1%

6

docs.neu.edu.tr

Internet Source

<1%

7

ir.uiowa.edu

Internet Source

<1%

8

Pengfei Dou, Yuhang Wu, Shishir K. Shah, Ioannis A. Kakadiaris. "Monocular 3D facial shape reconstruction from a single 2D image with coupled-dictionary learning and sparse

<1%

coding", Pattern Recognition, 2018

Publication

9

Zhimin Du, Xinqiao Jin, Xuebin Yang. "A robot fault diagnostic tool for flow rate sensors in air dampers and VAV terminals", Energy and Buildings, 2009

Publication

<1%

10

cubic.wlu.ca

Internet Source

<1%

