IOT based Face Recognition system

by

1) Akash Manish Lad-18BCE1357
2) Ram Kumar-18BCE1273
3) Chris Nikhil Nixen-18BCE1324
4) Sudharshan Raghu-18BME1312

A project report submitted to

Prof. Dr. Rajesh Kumar

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the course of

CSE3009 – Internet of Things

in

B.Tech. COMPUTER SCIENCE AND ENGINEERING

VIT CHENNAI

Vandalur – Kelambakkam Road

Chennai – 600127

MAY 2020

BONAFIDE CERTIFICATE

Certified that this project report entitled "Face Recognition for Attendance System Using Artificial Intelligence Concept**"** is a bonafide work of

1) Akash Manish Lad (18BCE1357)
2) Sudarshan Raghu (18BME1312)
3) Chris Nikhil Nixen (18BCE1324)
4) Ram Kumar (18BCE1273)

who carried out the Project work under my supervision and guidance for:

CSE3009 – Internet of Things

Prof. Rajesh Kumar

Assistant Professor

School of Computer Science and Engineering (SCOPE),

VIT Chennai

Chennai – 600 127.

ABSTRACT

Student Attendance System is essential in all learning institutes for checking the performance of students. In most learning organizations, student attendances are physically taken by the utilization of attendance sheets issued by the institution heads as a component of regulation.

The students sign in these sheets which are then filled or physically signed in to a PC for future verification analysis.

This strategy additionally makes it hard to track the attendance of individual students in a huge classroom environment.

In this project, we propose the plan and utilization of a face detection and recognition framework to consequently recognize students going to an address in a classroom and stamp their attendance by perceiving their faces.

In these, the recognized face in a picture (gotten from the camera) will be contrasted and the already stored faces caught at the time of enrolment.

TABLE OF CONTENTS

INTRODUCTION

OBJECTIVES AND GOALS

- Designing the hardware setup for this project (Raspberry pi 3B+, Sd card, display and pi-Camera)
- Installing the required software (Raspbian OS, AWS API and other 3 required packages for the raspberry pi ; etcher and AWS credentials for windows)
- Training the images of each students and thus programming to match it with the corresponding dataset during attendance.
- Mark the attendance in the .csv file of each student (present or absent in the sheet) and also student in time and out time in the cloud.
- Creating and interface which to show every detail of each student's attendance.

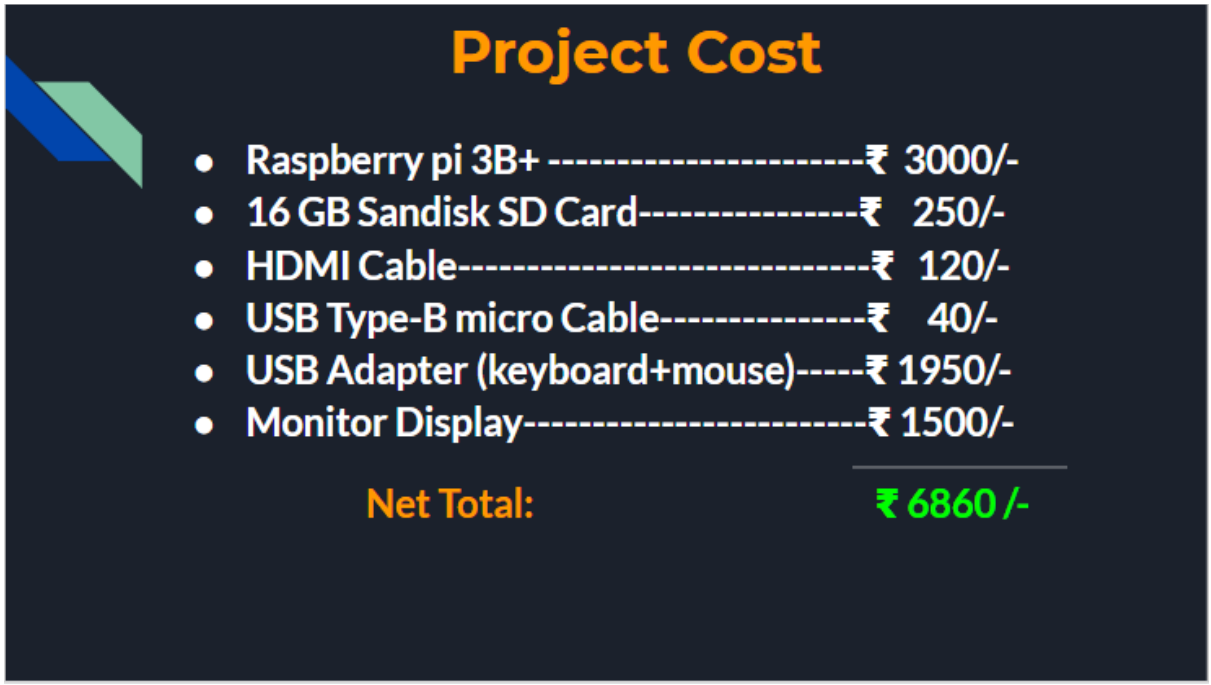
BENEFITS

IoT is growing rapidly; this is fuelled by providing supporting systems for the classrooms, especially those in which less time is available for attendance. This project details the overall design of a wireless IoT system which has been built and implemented. The automation centres on the microprocessor Raspberry-pi and uses communication modules to the cloud for analytics of attendance details of the students. The system has been tested and verified.

FEATURES

- Initially, the pi-Camera is trained by clicking pictures of each student in a classroom to store as the dataset.
- When the camera is turned on as students enter the classroom, it recognises the face and compares it with the dataset in a folder (Using machine learning neural networks concept).
- It then makes the change in the digital attendance sheet in the cloud.
- The attendance details of the student can view through an interface.

PROJECT COST



WIRELESS IOT SETUP

BLOCK DIAGRAM

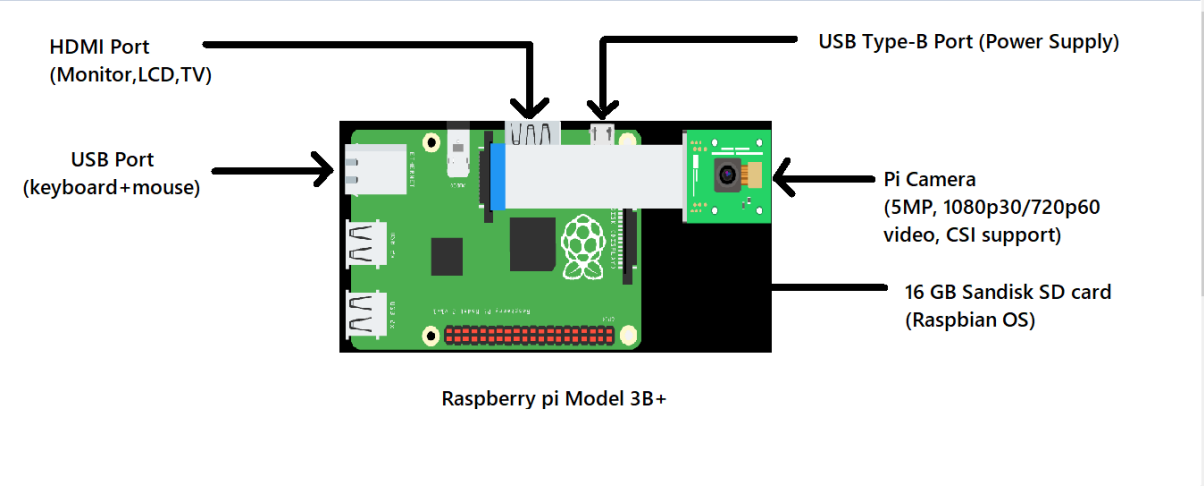


Figure 1: Overall block diagram of the function of Face recognition attendance system

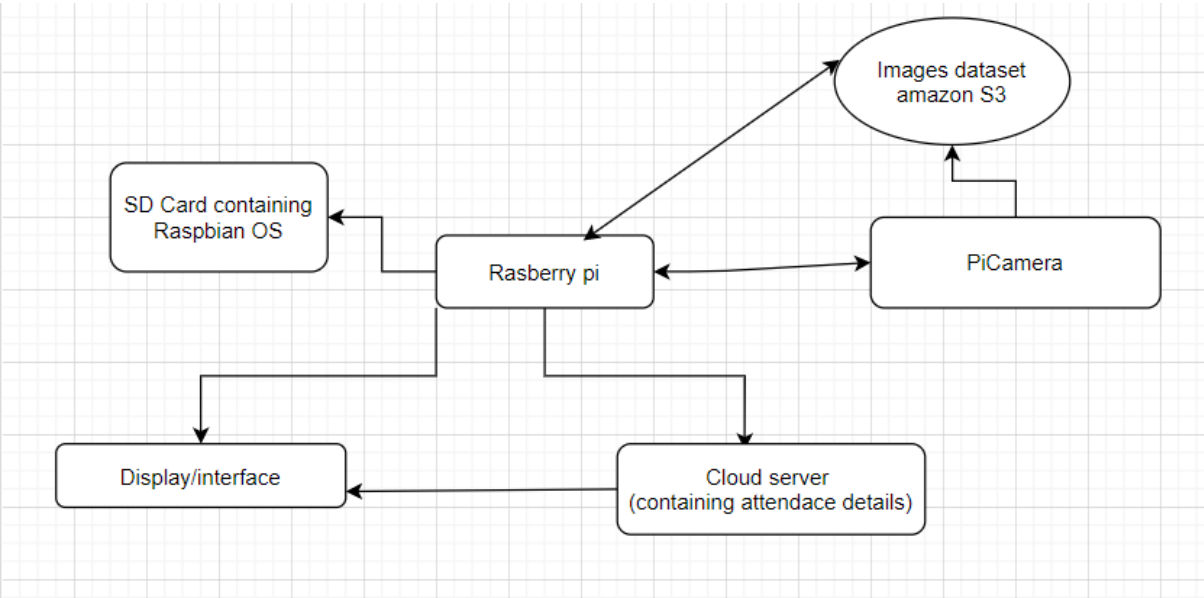


Figure 2: Secondary block diagram with all the components used

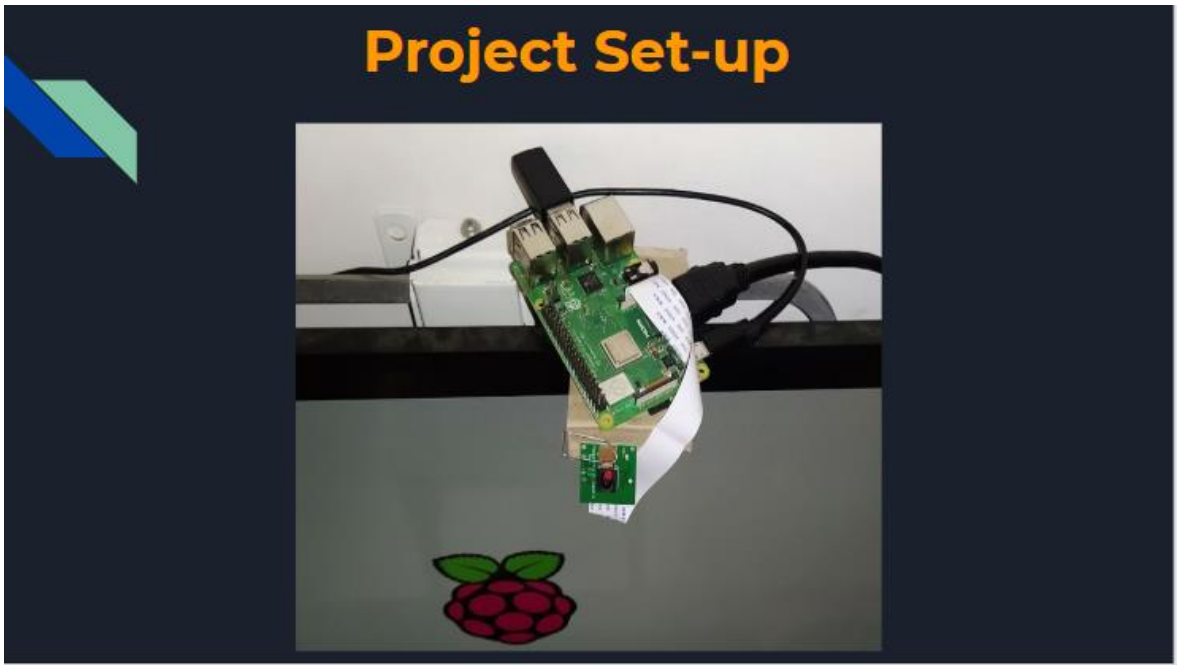
HARDWARE ANALYSIS

In this project Raspberry pi will be used as an IoT device to get local data by constructing a face recognition architecture.

Part 1 will be an IoT device setup, then
Part 2 will be face recognition setup.
Part 3 is Analytics on AWS Cloud.

Part 1 – IoT Component set-up

- Raspberrypi Model 3B+
- 16GB class10 SD card
- HDMI cable
- PiCamera for capturing images
- 3.5 inch touch LCD display

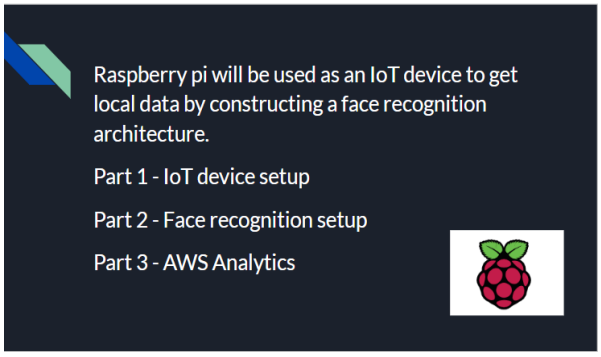


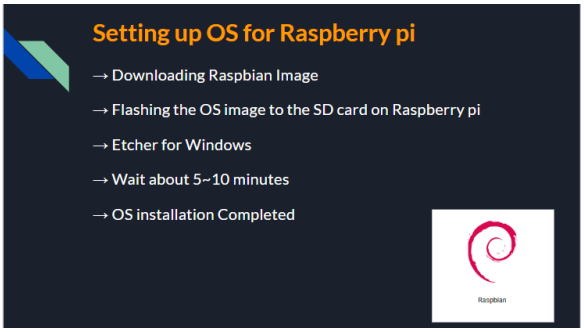
Part 2 – Image recognition Set-up

Step by step procedure:
- Downloading Raspbian Image
- Flashing the OS image to the SD card on Raspberry pi
- Etcher for Windows
- Wait time for about 5 to10 minutes
- Completion of OS installation for your Raspberry pi.

Raspberry Pi Packages Installation
- Install packages (3 minutes)
- sudo pip install awscli boto3 imutils
- Install dlib (1 hour) Follow this paragraph
- Face_Recognition (5 minutes)
- sudo pip2 --no-cache-dir install face_recognition

Setup AWS Credential On Your Raspberry Pi
- Find out your AWS Credential (Access key)
- Setup you AWS Credential in terminal
- aws configure
- AWS Access Key ID [None]: Type Your Access Key ID
- AWS Secret Access Key [None]: Type Your Secret Access Key
- Default region name [None]: us-east-1
- Default output format [None]: json





SOFTWARE IMPLEMENTATION

Program Code (train.py)

Performs Image recognition by training, taking images and hosting on interface developed using tkinter, numpy, pandas, opencv etc..

```
import tkinter as tk
from tkinter import Message ,Text
import cv2,os
import shutil
import csv
import numpy as np
from PIL import Image, ImageTk
import pandas as pd
import datetime
import time
import tkinter.ttk as ttk
import tkinter.font as font

window = tk.Tk()
window.title("Face_Recogniser")

dialog_title = 'QUIT'

#window.geometry('1280x720')
window.configure(background='blue')

#window.attributes('-fullscreen', True)

window.grid_rowconfigure(0, weight=1)
window.grid_columnconfigure(0, weight=1)

#path = "profile.jpg"

#Creates a Tkinter-compatible photo image, which can be used everywhere
Tkinter expects an image object.
#img = ImageTk.PhotoImage(Image.open(path))

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


#panel.pack(side = "left", fill = "y", expand = "no")

#cv_img = cv2.imread("img541.jpg")
#x, y, no_channels = cv_img.shape
#canvas = tk.Canvas(window, width = x, height =y)
```

```
#canvas.pack(side="left")
#photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(cv_img))
# Add a PhotoImage to the Canvas
#canvas.create_image(0, 0, image=photo, anchor=tk.NW)

#msg = Message(window, text='Hello, world!')

# Font is a tuple of (font_family, size_in_points, style_modifier_string)

# message = tk.Label(window, text="Face-Recognition-Based-Attendance-
Management-System" ,bg="Green"  ,fg="white"  ,width=50
,height=3,font=('times', 30, 'italic bold underline'))

# message.place(x=200, y=20)

lbl = tk.Label(window, text="Enter ID",width=20  ,height=2  ,fg="red"
,bg="yellow" ,font=('times', 15, ' bold ') )
lbl.place(x=400, y=200)

txt = tk.Entry(window,width=20  ,bg="yellow" ,fg="red",font=('times', 15, ' bold
'))
txt.place(x=700, y=215)

lbl2 = tk.Label(window, text="Enter Name",width=20  ,fg="red"  ,bg="yellow"
,height=2 ,font=('times', 15, ' bold '))
lbl2.place(x=400, y=300)

txt2 = tk.Entry(window,width=20  ,bg="yellow"  ,fg="red",font=('times', 15, '
bold ')  )
txt2.place(x=700, y=315)

lbl3 = tk.Label(window, text="Notification : ",width=20  ,fg="red"  ,bg="yellow"
,height=2 ,font=('times', 15, ' bold underline '))
lbl3.place(x=400, y=400)

message = tk.Label(window, text="" ,bg="yellow"  ,fg="red"  ,width=30
,height=2, activebackground = "yellow" ,font=('times', 15, ' bold '))
message.place(x=700, y=400)

lbl3 = tk.Label(window, text="Attendance : ",width=20  ,fg="red"  ,bg="yellow"
,height=2 ,font=('times', 15, ' bold  underline'))
lbl3.place(x=400, y=650)


message2 = tk.Label(window, text="" ,fg="red"   ,bg="yellow",activeforeground
= "green",width=30  ,height=2  ,font=('times', 15, ' bold '))
message2.place(x=700, y=650)

def clear():
    txt.delete(0, 'end')
    res = ""
    message.configure(text= res)
```

```
def clear2():
    txt2.delete(0, 'end')
    res = ""
    message.configure(text= res)

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        pass

    try:
        import unicodedata
        unicodedata.numeric(s)
        return True
    except (TypeError, ValueError):
        pass

    return False

def TakeImages():
    Id=(txt.get())
    name=(txt2.get())
    if(is_number(Id) and name.isalpha()):
        cam = cv2.VideoCapture(0)
        harcascadePath = "haarcascade_frontalface_default.xml"
        detector=cv2.CascadeClassifier(harcascadePath)
        sampleNum=0
        while(True):
            ret, img = cam.read()
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = detector.detectMultiScale(gray, 1.3, 5)
            for (x,y,w,h) in faces:
                cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
      #incrementing sample number
                sampleNum = sampleNum+1
      #saving the captured face in the dataset folder TrainingImage
                cv2.imwrite("TrainingImage\ "+name +"."+Id +'.'+ str(sampleNum)
+ ".jpg", gray[y:y+h,x:x+w])
                #display the frame
                cv2.imshow('frame',img)
            #wait for 100 miliseconds
            if cv2.waitKey(100) & 0xFF == ord('q'):
                break
            # break if the sample number is morethan 100
            elif sampleNum>60:
                break
        cam.release()
        cv2.destroyAllWindows()
        res = "Images Saved for ID : " + Id +" Name : "+ name
```

```
        row = [Id , name]
        with open('StudentDetails\StudentDetails.csv','a+') as csvFile:
            writer = csv.writer(csvFile)
            writer.writerow(row)
        csvFile.close()
        message.configure(text= res)
    else:
        if(is_number(Id)):
            res = "Enter Alphabetical Name"
            message.configure(text= res)
        if(name.isalpha()):
            res = "Enter Numeric Id"
            message.configure(text= res)

def TrainImages():
    recognizer = cv2.face.LBPHFaceRecognizer_create()#recognizer =
cv2.face.LBPHFaceRecognizer_create()#$cv2.createLBPHFaceRecognizer()
    harcascadePath = "haarcascade_frontalface_default.xml"
    detector =cv2.CascadeClassifier(harcascadePath)
    faces,Id = getImagesAndLabels("TrainingImage")
    recognizer.train(faces, np.array(Id))
    recognizer.save("TrainingImageLabel\Trainner.yml")
    res = "Image Trained"#+",".join(str(f) for f in Id)
    message.configure(text= res)




def getImagesAndLabels(path):
    #get the path of all the files in the folder
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    #print(imagePaths)

    #create empth face list
    faces=[]
    #create empty ID list
    Ids=[]
    #now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        #loading the image and converting it to gray scale
        pilImage=Image.open(imagePath).convert('L')
        #Now we are converting the PIL image into numpy array
        imageNp=np.array(pilImage,'uint8')
        #getting the Id from the image
        Id=int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces.append(imageNp)
        Ids.append(Id)
    return faces,Ids

def TrackImages():
```

```
    recognizer = cv2.face.LBPHFaceRecognizer_create()
#cv2.createLBPHFaceRecognizer()
    recognizer.read("TrainingImageLabel\Trainner.yml")
    harcascadePath = "haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(harcascadePath);
    df=pd.read_csv("StudentDetails\StudentDetails.csv")
    cam = cv2.VideoCapture(0)
    font = cv2.FONT_HERSHEY_SIMPLEX
    col_names =  ['Id','Name','Date','Time']
    attendance = pd.DataFrame(columns = col_names)
    while True:
        ret, im =cam.read()
        gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        faces=faceCascade.detectMultiScale(gray, 1.2,5)
        for(x,y,w,h) in faces:
            cv2.rectangle(im,(x,y),(x+w,y+h),(225,0,0),2)
            Id, conf = recognizer.predict(gray[y:y+h,x:x+w])
            if(conf < 50):
                ts = time.time()
                date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
                timeStamp =
datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
                aa=df.loc[df['Id'] == Id]['Name'].values
                tt=str(Id)+"-"+aa
                attendance.loc[len(attendance)] = [Id,aa,date,timeStamp]

            else:
                Id='Unknown'
                tt=str(Id)
            if(conf > 75):
                noOfFile=len(os.listdir("ImagesUnknown"))+1
                cv2.imwrite("ImagesUnknown\Image"+str(noOfFile) + ".jpg",
im[y:y+h,x:x+w])
            cv2.putText(im,str(tt),(x,y+h), font, 1,(255,255,255),2)
        attendance=attendance.drop_duplicates(subset=['Id'],keep='first')
        cv2.imshow('im',im)
        if (cv2.waitKey(1)==ord('q')):
            break
    ts = time.time()
    date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
    timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
    Hour,Minute,Second=timeStamp.split(":")
    fileName="Attendance\Attendance_"+date+"_"+Hour+"-"+Minute+"-
"+Second+".csv"
    attendance.to_csv(fileName,index=False)
    cam.release()
    cv2.destroyAllWindows()
    #print(attendance)
    res=attendance
    message2.configure(text= res)
```

```
clearButton = tk.Button(window, text="Clear", command=clear  ,fg="red"
,bg="yellow"  ,width=20  ,height=2 ,activebackground = "Red" ,font=('times',
15, ' bold '))
clearButton.place(x=950, y=200)

clearButton2 = tk.Button(window, text="Clear", command=clear2  ,fg="red"
,bg="yellow"  ,width=20  ,height=2, activebackground = "Red" ,font=('times',
15, ' bold '))
clearButton2.place(x=950, y=300)

takeImg = tk.Button(window, text="Take Images", command=TakeImages
,fg="red"  ,bg="yellow"  ,width=20  ,height=3, activebackground = "Red"
,font=('times', 15, ' bold '))
takeImg.place(x=200, y=500)

trainImg = tk.Button(window, text="Train Images", command=TrainImages
,fg="red"  ,bg="yellow"  ,width=20  ,height=3, activebackground = "Red"
,font=('times', 15, ' bold '))
trainImg.place(x=500, y=500)

trackImg = tk.Button(window, text="Track Images", command=TrackImages
,fg="red"  ,bg="yellow"  ,width=20  ,height=3, activebackground = "Red"
,font=('times', 15, ' bold '))
trackImg.place(x=800, y=500)

quitWindow = tk.Button(window, text="Quit", command=window.destroy
,fg="red"  ,bg="yellow"  ,width=20  ,height=3, activebackground = "Red"
,font=('times', 15, ' bold '))

quitWindow.place(x=1100, y=500)

copyWrite = tk.Text(window, background=window.cget("background"),
borderwidth=0,font=('times', 30, 'italic bold underline'))
copyWrite.tag_configure("superscript", offset=10)

copyWrite.configure(state="disabled",fg="red"  )
copyWrite.pack(side="left")
copyWrite.place(x=800, y=750)

window.mainloop()
```

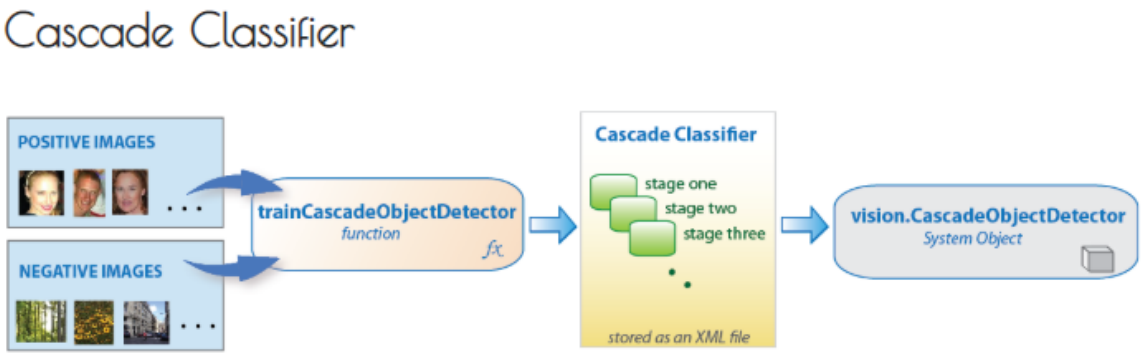
Explanation of Image recognition code (train.py) using haar-cascade classifier

The cascade classifier consists of a collection of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called decision stumps. Each stage is trained using a technique called boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners.

Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

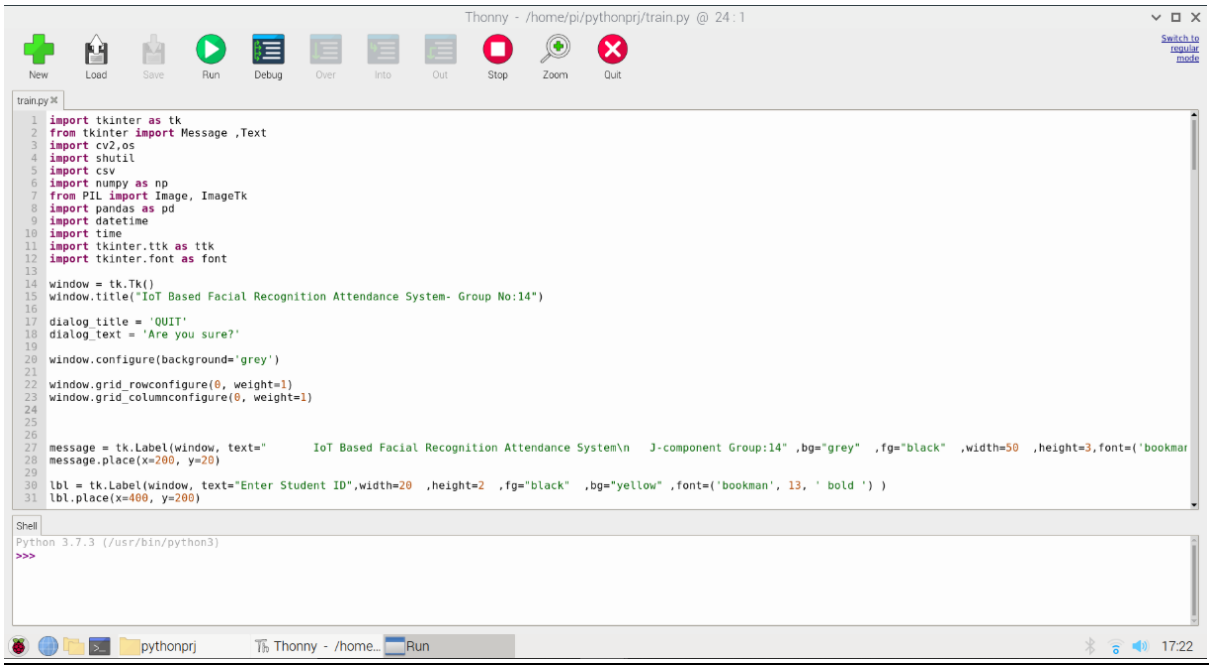
The stages are designed to reject negative samples as fast as possible. The assumption is that the vast majority of windows do not contain the object of interest. Conversely, true positives are rare and worth taking the time to verify.

A true positive occurs when a positive sample is correctly classified.
A false positive occurs when a negative sample is mistakenly classified as positive.
A false negative occurs when a positive sample is mistakenly classified as negative.
To work well, each stage in the cascade must have a low false negative rate. If a stage incorrectly labels an object as negative, the classification stops, and you cannot correct the mistake. However, each stage can have a high false positive rate. Even if the detector incorrectly labels a nonobject as positive, you can correct the mistake in subsequent stages. Adding more stages reduces the overall false positive rate, but it also reduces the overall true positive rate.

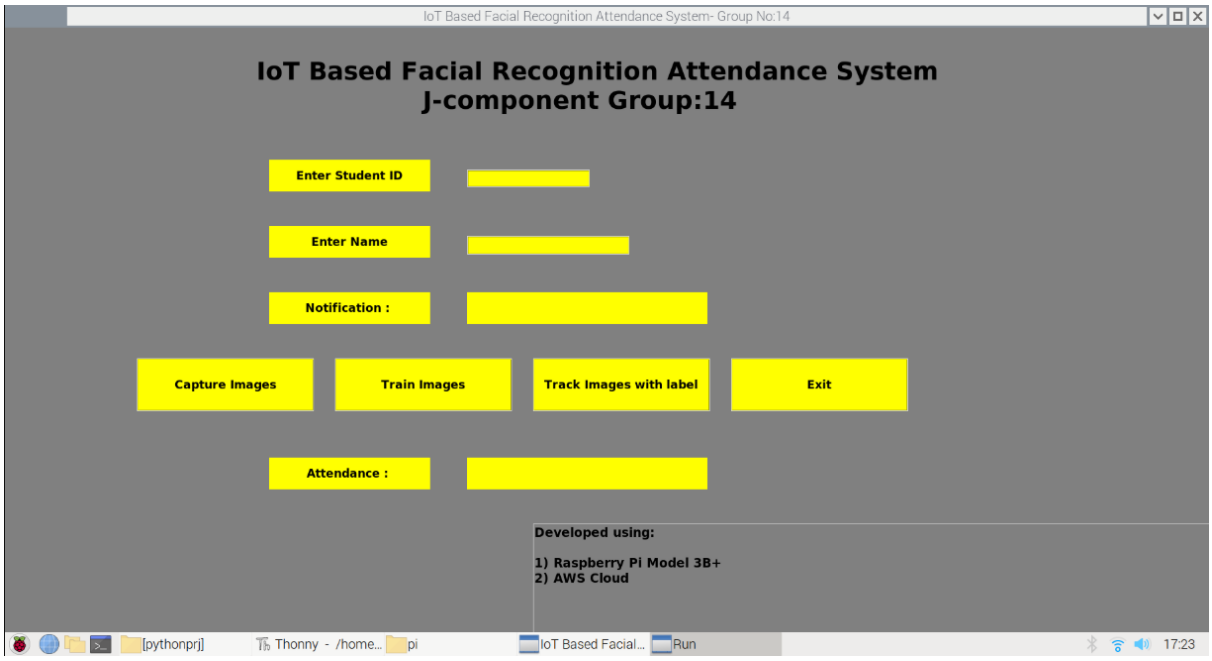


Cascade classifier training requires a set of positive samples and a set of negative images. You must provide a set of positive images with regions of interest specified to be used as positive samples. You can use the Image Labeller to label objects of interest with bounding boxes. The Image Labeller outputs a table to use for positive samples. You also must provide a set of negative images from which the function generates negative samples automatically. To achieve acceptable detector accuracy, set the number of stages, feature type, and other function parameters.

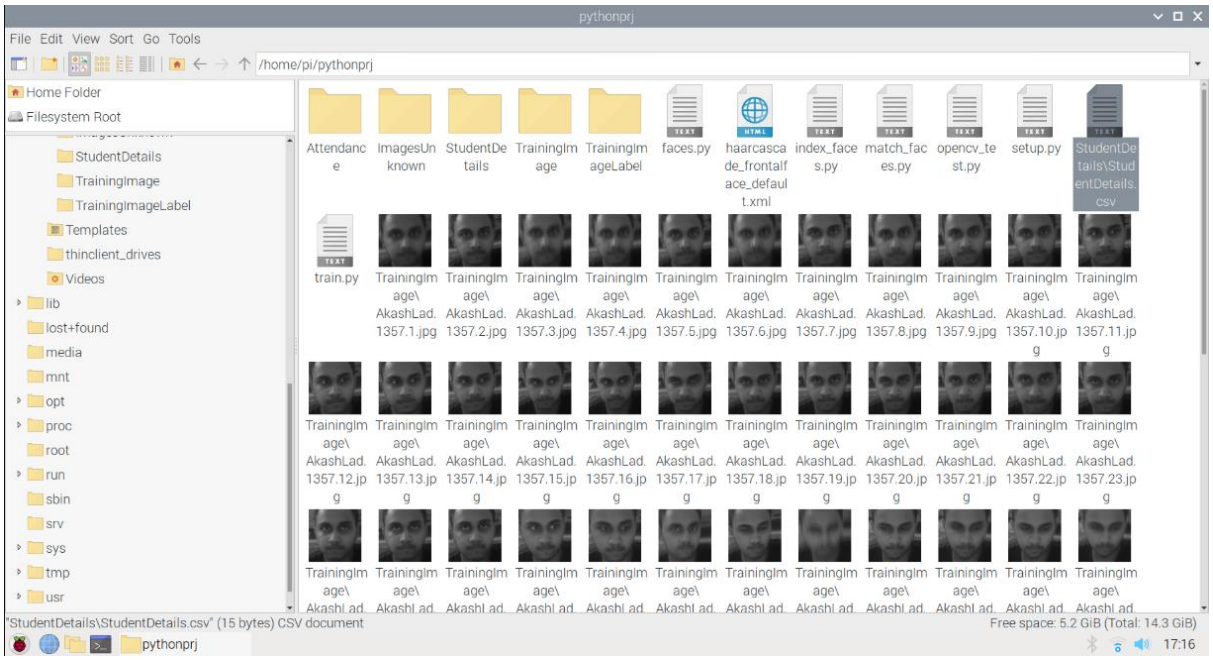
Screenshot:



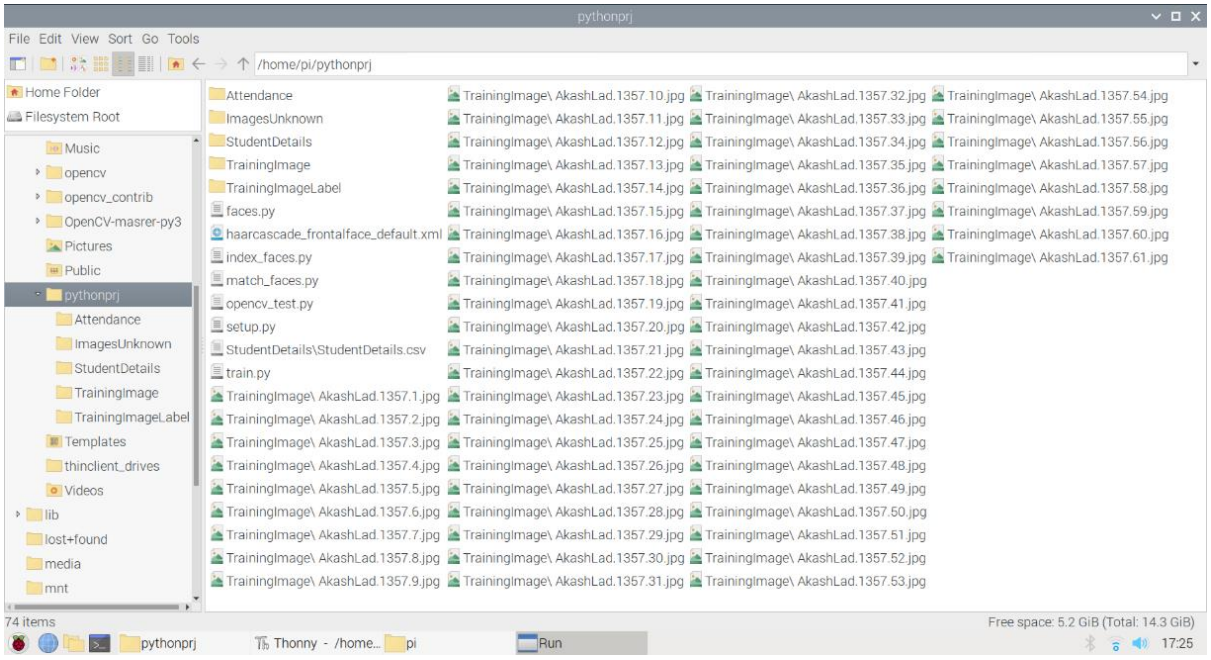
Python code (train.py) in raspberry pi editor



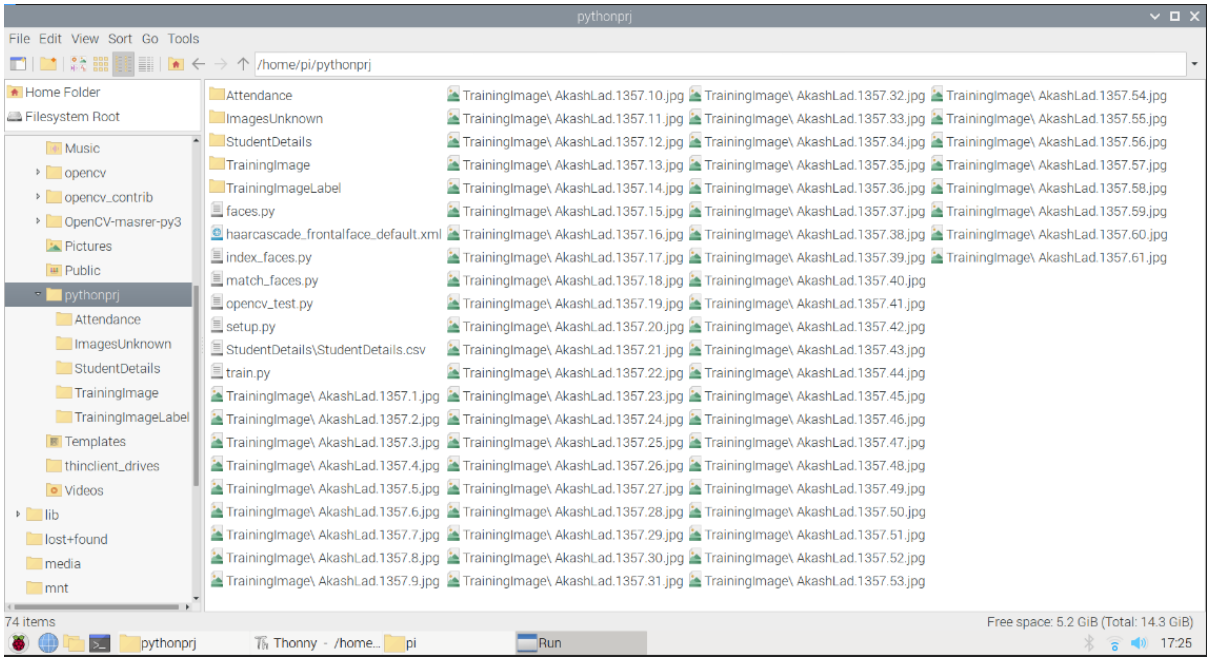
Interface developed using tkinter python library

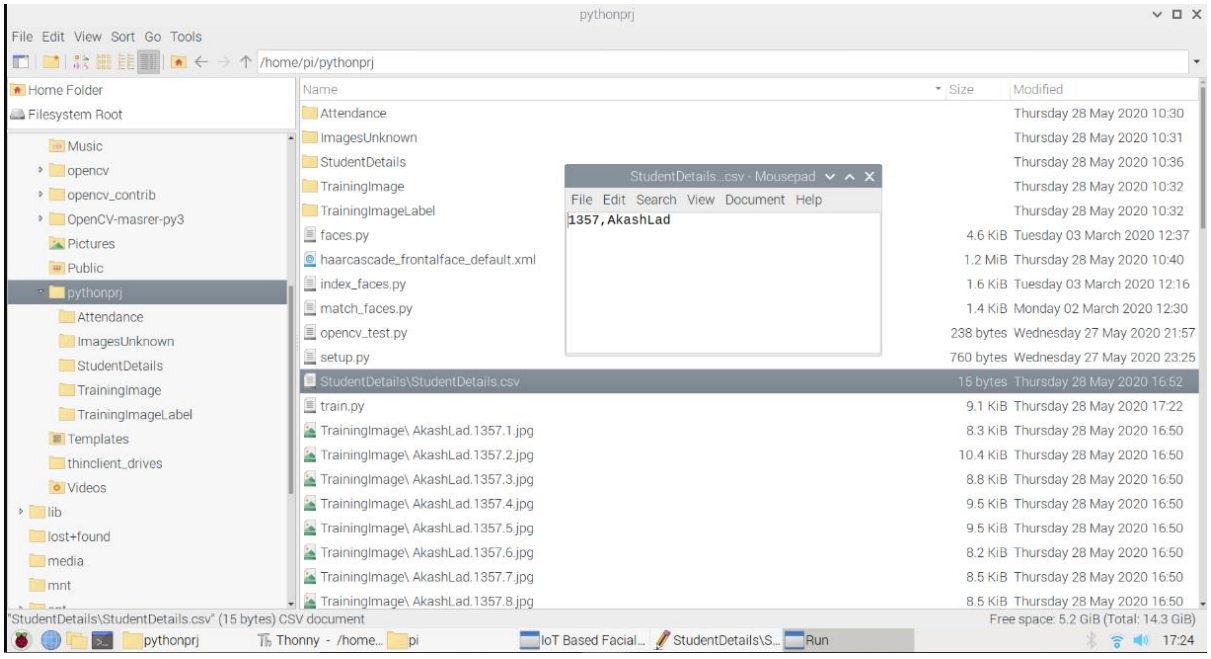


Training images stored in "TrainingImage" folder



Training dataset containing 60 images



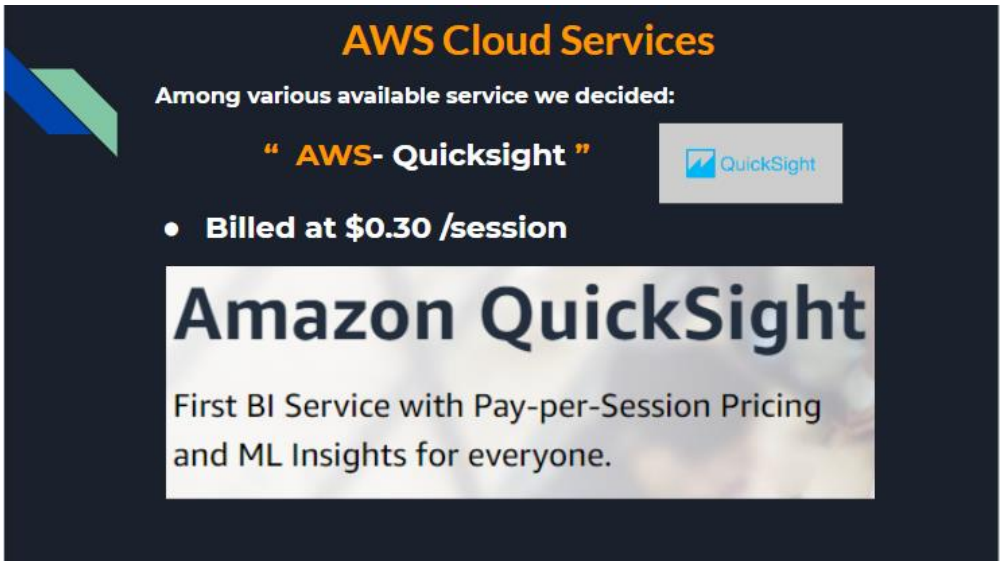


Attendance recorded in CSV format containing registration number and name

Part 3 – Analytics on AWS Cloud

After the Image recognition has been successfully completed and recorded, it becomes essential to get meaning out of the data generated. To get the insight of data generated by recording images from raspberry pi with help of pi-camera, we perform analytics on Cloud to generate information and inferences. Using these results, we can make certain predictions and generate reports and optimise our work.

We decided to use **AWS-Quicksight** to perform Analytics and report generation.



Amazon QuickSight is a fast, cloud-powered business intelligence service that makes it easy to deliver insights to everyone in your organization.

As a fully managed service, QuickSight lets you easily create and publish interactive dashboards that include ML Insights. Dashboards can then be accessed from any device, and embedded into your applications, portals, and websites. With Pay-per-Session pricing, QuickSight allows you to give everyone access to the data they need, while only paying for what you use.

Pay-per-session pricing

QuickSight offers a unique, industry first pay-per-session model for dashboard readers, users who consume dashboards others have created. Instead of paying a fixed license cost per month, readers are billed $0.30 for a 30-minute session up to a maximum charge of $5/reader/month for unlimited use. This pricing model allows all of your users to access secure, interactive dashboards and email reports on a pay-per-session basis with no upfront costs or complex capacity planning.

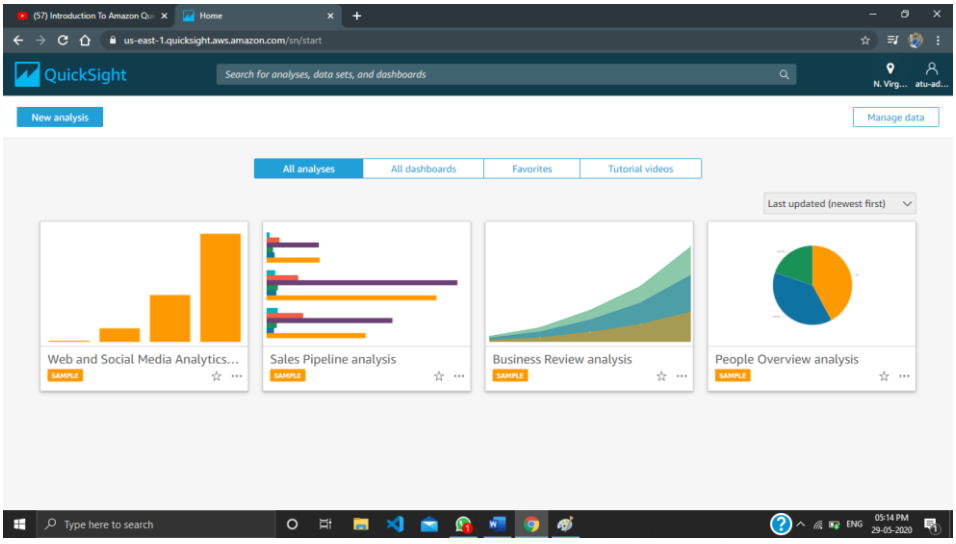
Features of **AWS-Quicksight:**

- Option to choose data of our own choice.
  It can be in any raw format like (.csv, .tsv,.xml,.xlsx etc…)

- SPICE (super-fast, parallel, in-memory, calculation engine)
  With SPICE, QuickSight's in-memory calculation engine we can achieve blazing fast performance at scale. SPICE automatically replicates data for high availability allowing thousands of users to simultaneously perform fast, interactive analysis while shielding your underlying data infrastructure, saving our time and resources.

- Explore, analyze, collaborate
  QuickSight gives users and analysts self-service business intelligence (BI), so they can answer their own questions, collaborate, and share insights. With QuickSight, the users can connect to data sources, create/edit datasets, create visual analyses, invite co-workers to collaborate on analyses, and publish dashboards and reports.

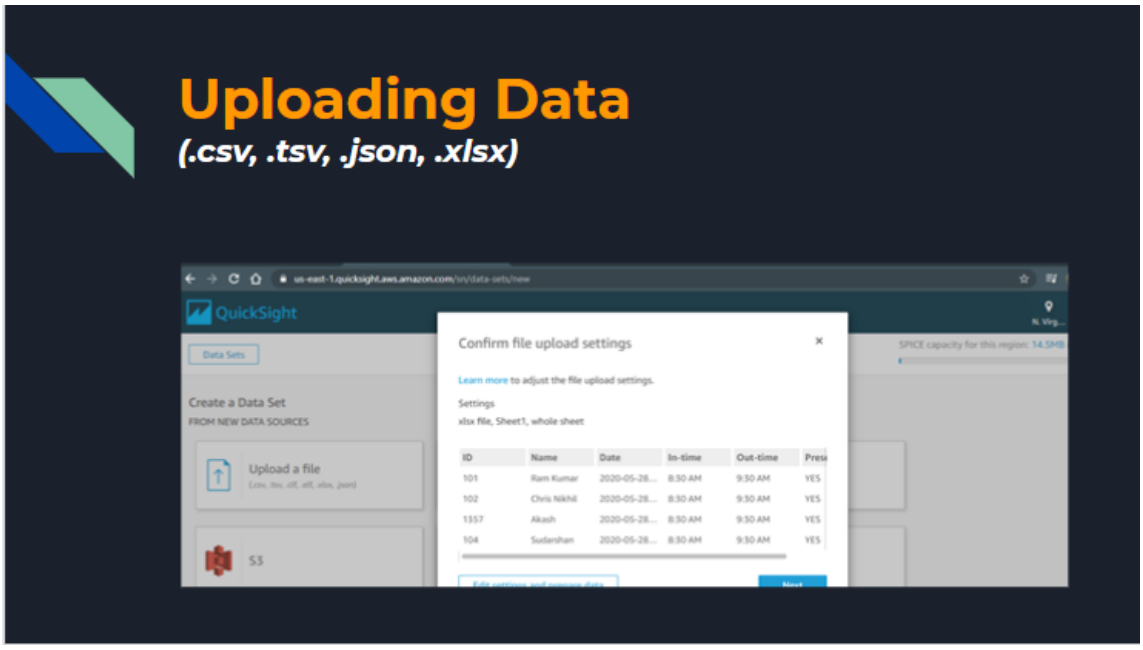
- Deliver rich, interactive dashboards for the readers
  QuickSight makes it easy and fast to create interactive dashboards and reports for viewers.



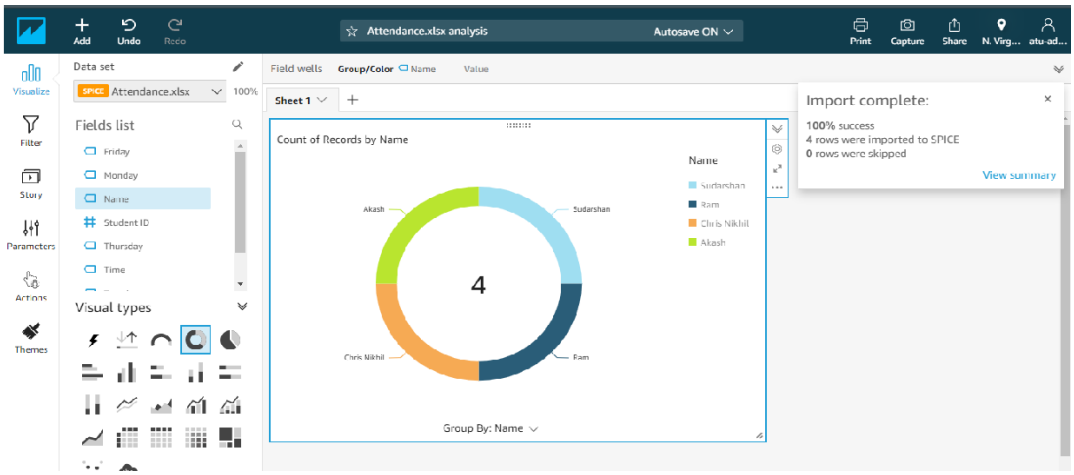
Quicksight Dashboard

Deploying DATA on Cloud

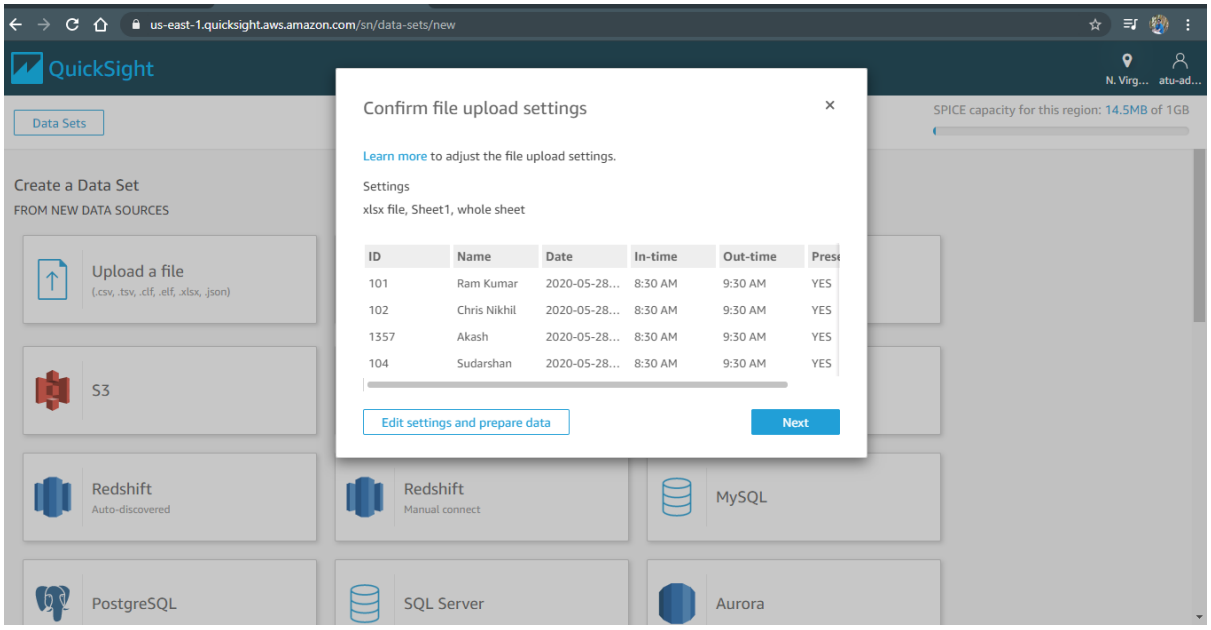
- The data is deployed on the Quicksight dashboard.
- Student's attendance is uploaded in CSV format and Quicksight provides option to alter the format of the data.
- Click on next if no changes are required to be done, upload the CSV file in the dashboard



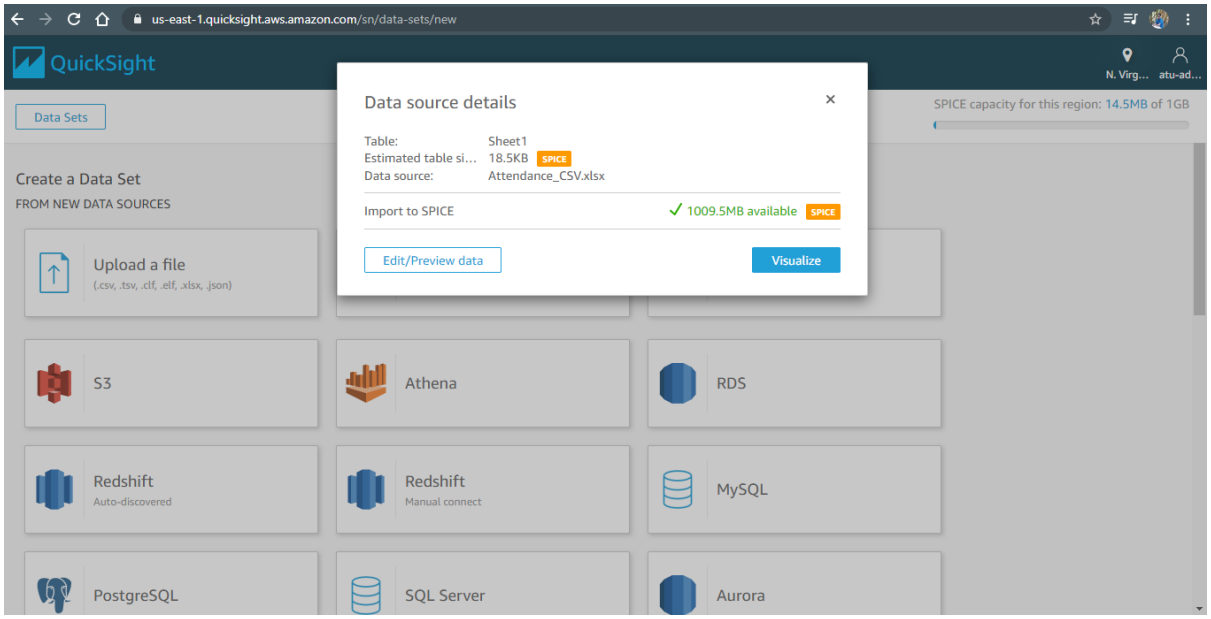
CSV file uploaded in Dashboard



Analytics Dashboard

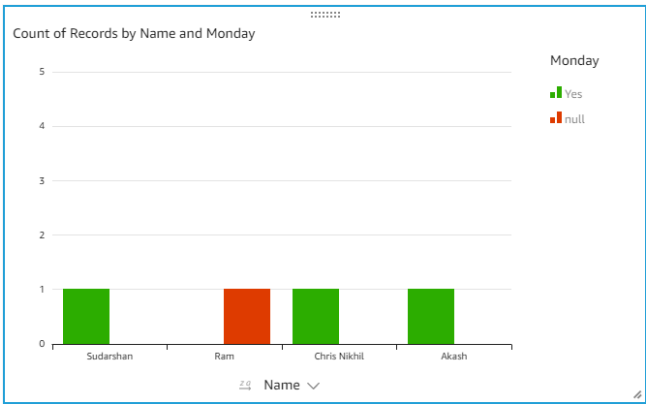


Tabular form of data of CSV file



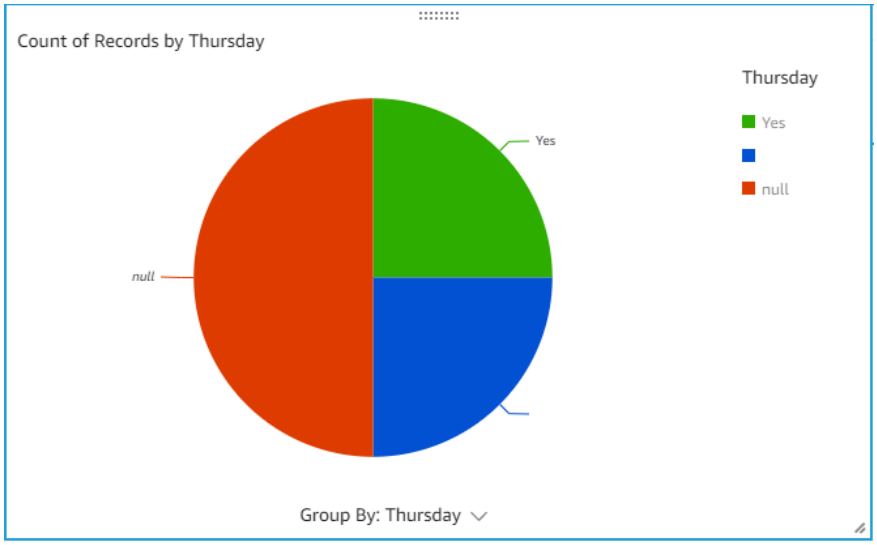
Uploading of data with details (memory occupied/available)

Analytics Dashboard & Plottings



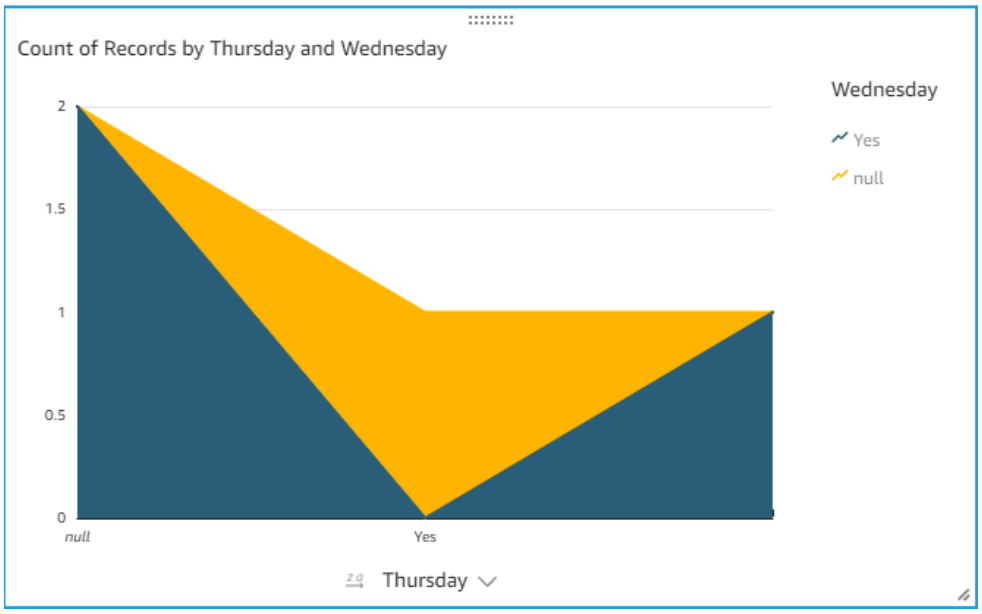
Bar-plot

Bar-plot showing record of students present marked in green and absent marked in red. Plotting is done in such a way that X-axis contains student names and Y-axis contains count.



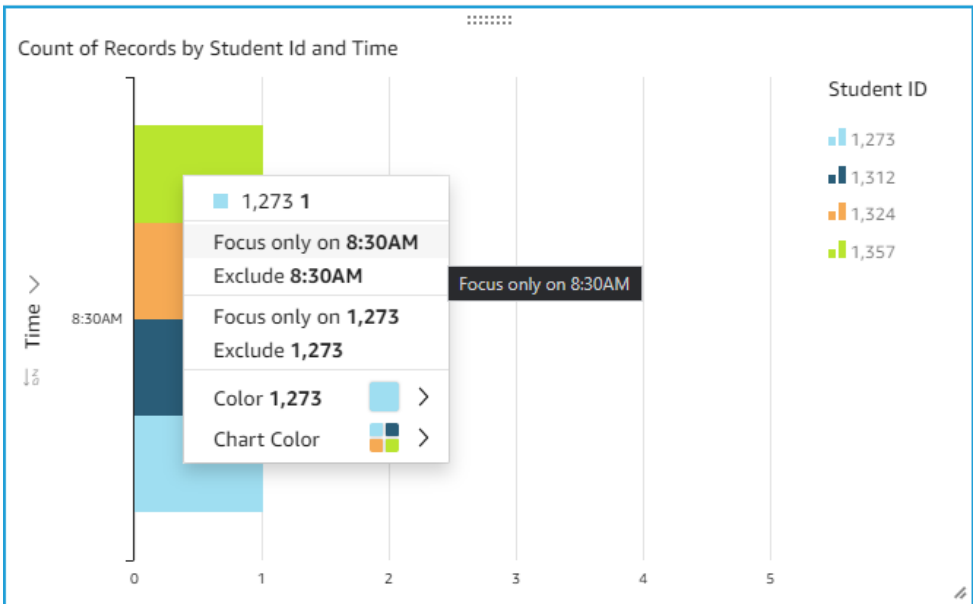
Pie-chart

Pie chart signifies the record of students present on Thursday. Red region indicates null values, blue indicates student available for particular time-slot and green indicating attendance marked successfully.



Area-plot

An area chart or area graph displays graphically quantitative data. It is based on the line chart. The area between axis and line are commonly emphasized with colours, textures and hatchings.



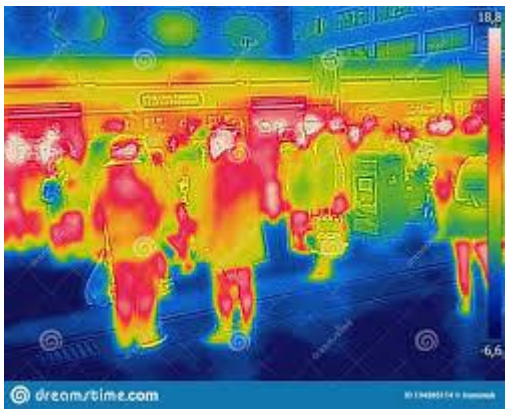
Data-Analysis in auto-mode

Unique feature that Quicksight provides whereby, the data is transformed and analysed automatically.

FUTURE WORK

In consideration with **Post COVID-19** Scenario

- Thermal Screening and Mask Detection for Attendance & Reporting



Heat-maps for contact tracing

Mask-Detection at public places

- Using Infrared Sensors and High Quality Camera for measuring temperature and image recognition
- Reporting nearby Medical centre in case of Emergency and health-care hospitality

CONCLUSION

- The IOT based face recognition system was built and implemented.
- The prototype developed can be implemented at any classroom.
- The system implements the machine learning concept in real life by training and matching images of students in a classroom with the help of a microprocessor, raspberry pi and its camera.
- The results are updated in the digital interface in cloud
- Thus, the data is conveniently available to view by students, staff and other higher officials

Number of hours spent on J-component

"IoT based Attendance System using Raspberry Pi and AWS Cloud *(Quicksight)*"

Project Time-Line:

End of Week-1 & Week-2 (3/Feb-8/Feb) & (10/Feb-15/Feb) (Project Planning & Analysis) **Research Paper:** IoT based automated attendance system with Face recognition By D.Narendar Singh, M Kusuma Sri,K Mouonika Analysis of Project components	Akash: Getting started with AWS Sudarshan: Installing dependencies for raspberry pi Chris Nikhil: website creation Ram: Designing the front-end

End of Week-3 & Week-4 (17/Feb-22/Feb) & (22/Feb-26/Feb) (Getting started with Project Implementation)	Akash: Data Analysis and recognition Sudarshan: Streaming images from camera and segmenting Chris Nikhil: Website hosting and api interface Ram: Creating image data sets

We have dedicated 2 hours per day during first 2 weeks and following next 2 weeks. We searched for reliable resources, documentation for python on working with numpy, pandas, opencv, tkinter, datetime, tutorials for setting up raspberry pi and working with the camera, research papers regarding haar-cascade classifier and articles related to it on Google Scholar. We also searched on working with AWS cloud and completed few MOOCs as a part of learning process. We spent around 2 x 28 = 50-56 hours during our college hours and free-time. During the vacation we spent around 10-12 hours to optimise the working model with various monitor and by tuning our neural network based recognition model to achieve accuracy about 87%.

Thus, considering overall scenario, we have spent 50 + 10 = 60 hours.

References

1. IoT attendance system
   https://www.ijitee.org/wpcontent/uploads/papers/v8i6s4/F10930486S419.pdf

2. Accurate & Optimised Face Recognition research paper
   https://ieeexplore.ieee.org/xpl/conhome/7889414/proceeding

3. LBPH Based Improved Face Recognition At Low Resolution
   https://www.researchgate.net/publication/327980768_LBPH_Based_Improved_Face_Recognition_At_Low_Resolution

4. Opencv Documentation: https://docs.opencv.org/

5. AWS-Quicksight-User-Guide:
   https://docs.aws.amazon.com/quicksight/latest/user/welcome.html

6. YouTube Tutorials for Raspberry pi Configuration:
   https://www.youtube.com/watch?v=RpseX2ylEuw&list=PLQVvvaa0QuDesV8WWHLLXW_avmTzHmJLv