CODINGS

```
# main.py

import os

import base64

from flask import Flask, render_template, Response, redirect, request, session,
abort, url_for

from camera import VideoCamera

from cryptography.hazmat.backends import default_backend

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

from cryptography.fernet import Fernet

import argparse

import cv2

import shutil

import random

from random import seed

from random import randint

import time

import PIL.Image

from PIL import Image, ImageChops

import numpy as np

import pandas as pd

import random

import seaborn as sns

import matplotlib.pyplot as plt

import math

import imagehash
```

```
import mysql.connector
import urllib.request
import urllib.parse
from werkzeug.utils import secure_filename
from urllib.request import urlopen
import webbrowser
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="",
  charset="utf8",
  database="animal_repellent"
)
UPLOAD_FOLDER = 'static/trained'
ALLOWED_EXTENSIONS = { 'png', 'jpg', 'jpeg', 'gif'}
app = Flask(__name__)
app.secret_key = 'abcdef'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
#@app.route('/')
#def index():
#    return render_template('index.html')
@app.route('/', methods=['GET', 'POST'])
def index():
    msg=""
```

```
    ff3=open("ulog.txt","w")
    ff3.write("")
    ff3.close()
        act=request.args.get("act")
    act2=request.args.get("act2")
    act3=request.args.get("act3")
   return render_template('index.html',msg=msg,act=act,act2=act2,act3=act3)
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    msg=""
    act=request.args.get("act")
    act2=request.args.get("act2")
    act3=request.args.get("act3")
    page=request.args.get("page")
    ff=open("msg.txt","w")
    ff.write('0')
    ff.close()
    fn=request.args.get("fn")
    fn2=""
    animal=""
    ss=""
    cname=[]
    afile=""
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM animal_info order by id")
    row = mycursor.fetchall()
    for row1 in row:
```

```
        cname.append(row1[1])
            if request.method=='POST':
        #print("d")
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
       file_type = file.content_type
        # if user does not select file, browser also
        # submit an empty part without filename
        tf=file.filename
        ff=open("log.txt","w")
        ff.write(tf)
        ff.close()
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file:
            fname = "m1.jpg"
            filename = secure_filename(fname)
            file.save(os.path.join("static/test", filename))
                cutoff=1
            for fname in os.listdir("static/dataset"):
                hash0 =
imagehash.average_hash(Image.open("static/dataset/"+fname))
                hash1 = imagehash.average_hash(Image.open("static/test/m1.jpg"))
                cc1=hash0 - hash1
                print("cc="+str(cc1))
```

```
            if cc1<=cutoff:
                fn=fname
                ss="ok"
                break
        if ss=="ok":
            act3="yes"
        else:
            act3="no"
         return redirect(url_for('upload', act3=act3,fn=fn,page=page))
       '''if request.method=='POST':
    uname=request.form['uname']
    pwd=request.form['pass']
    cursor = mydb.cursor()
    cursor.execute('SELECT * FROM admin WHERE username = %s AND
password = %s', (uname, pwd))
    account = cursor.fetchone()
    if account:
        session['username'] = uname
return redirect(url_for('admin'))
    else:
        # Account doesnt exist or username/password incorrect
        msg = 'Incorrect username/password!'
  '''
    if act3=="yes":
    g=1
    print(fn)
    #object_detect(fn)
    ##
```

```
        ff2=open("static/trained/tdata.txt","r")
        rd=ff2.read()
        ff2.close()
    num=[]
        r1=rd.split(',')
        s=len(r1)
        ss=s-1
        i=0
        while i<ss:
                num.append(int(r1[i]))
            i+=1
#print(num)
        dat=toString(num)
        dd2=[]
        d1=dat.split(',')

        ##

        for gff in d1:

            gf1=gff.split('-')

            if gf1[0]==fn:
                gid=int(gf1[1])-1
                fn2="c_"+fn
                animal=cname[gid]
                afile="a"+gf1[1]+".mp3"
```

```
 break
        print(fn2)
        print(animal)
        print(afile)

        ff3=open("ulog.txt","r")
        user=ff3.read()
        ff3.close()

        ff4=open("sms.txt","r")
        sms=ff4.read()
        ff4.close()

        if user=="":
            aa=1
        else:

            if sms=="":
                aa=1
            else:
                mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
                row1 = mycursor.fetchone()
                mobile=row1[2]
                name=row1[1]

                mess=animal+" detected"
```

```
url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess=
"+mess+"&mobile="+str(mobile)
            webbrowser.open_new(url)


            ff41=open("sms.txt","w")
            ff41.write("")
            ff41.close()


        mycursor = mydb.cursor()
        mycursor.execute("SELECT max(id)+1 FROM animal_detect")
        maxid = mycursor.fetchone()[0]
        if maxid is None:
            maxid=1
        sql = "INSERT INTO animal_detect(id,user,animal,image_name)
VALUES (%s, %s,%s,%s)"
        val = (maxid,user,animal,fn2)
        mycursor.execute(sql, val)
        mydb.commit()


    elif act3=="no":
        g=2
        msg="No Result"
    return
render_template('upload.html',msg=msg,act=act,act2=act2,act3=act3,fn=fn,ani
mal=animal,fn2=fn2,afile=afile,page=page)


@app.route('/process_upload', methods=['GET', 'POST'])
def process_upload():
```

```
msg=""
act=request.args.get("act")
act2=request.args.get("act2")
act3=request.args.get("act3")
page=request.args.get("page")
ff=open("msg.txt","w")
ff.write('0')
ff.close()
fn=request.args.get("fn")
fn2=""
animal=""
ss=""
cname=[]
afile=""
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM animal_info order by id")
row = mycursor.fetchall()
for row1 in row:
    cname.append(row1[1])

if request.method=='POST':
    #print("d")
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['file']
```

```
        file_type = file.content_type
        # if user does not select file, browser also
        # submit an empty part without filename
        tf=file.filename
        ff=open("log.txt","w")
        ff.write(tf)
        ff.close()
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file:
            fname = "m1.jpg"
            filename = secure_filename(fname)

            file.save(os.path.join("static/test", filename))



            cutoff=1
            for fname in os.listdir("static/dataset"):
                hash0 =
imagehash.average_hash(Image.open("static/dataset/"+fname))
                hash1 = imagehash.average_hash(Image.open("static/test/m1.jpg"))
                cc1=hash0 - hash1
                print("cc="+str(cc1))
                if cc1<=cutoff:
                    fn=fname
                    ss="ok"
                    break
```

```
            if ss=="ok":
                act3="yes"
            else:
                act3="no"
return redirect(url_for('process_upload', act3=act3,fn=fn,page=page))
    '''if request.method=='POST':
        uname=request.form['uname']
        pwd=request.form['pass']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM admin WHERE username = %s AND
password = %s', (uname, pwd))
        account = cursor.fetchone()
        if account:
            session['username'] = uname
            return redirect(url_for('admin'))
        else:
            # Account doesnt exist or username/password incorrect
            msg = 'Incorrect username/password!'
    '''


    if act3=="yes":
        g=1
        print(fn)
        #object_detect(fn)
        ##
        ff2=open("static/trained/tdata.txt","r")
        rd=ff2.read()
        ff2.close()
```

```
num=[]
r1=rd.split(',')
s=len(r1)
ss=s-1
i=0
while i<ss:

    num.append(int(r1[i]))
    i+=1

#print(num)
dat=toString(num)
dd2=[]
d1=dat.split(',')
##
        for gff in d1:
    gf1=gff.split('-')
    if gf1[0]==fn:
        gid=int(gf1[1])-1
        fn2="c_"+fn
        animal=cname[gid]
        afile="a"+gf1[1]+".mp3"
        break
print(fn2)
print(animal)
print(afile)
```

```
        ff3=open("ulog.txt","r")
        user=ff3.read()
        ff3.close()
        ff4=open("sms.txt","r")
        sms=ff4.read()
        ff4.close()
        if user=="":
            aa=1
        else:
            if sms=="":
                aa=1
            else:
                mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
                row1 = mycursor.fetchone()
                mobile=row1[2]
                name=row1[1]
                mess=animal+" detected"

url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess=
"+mess+"&mobile="+str(mobile)
                webbrowser.open_new(url)
                ff41=open("sms.txt","w")
                ff41.write("")
                ff41.close()
            mycursor = mydb.cursor()
            mycursor.execute("SELECT max(id)+1 FROM animal_detect")
            maxid = mycursor.fetchone()[0]
```

```
        if maxid is None:
            maxid=1
        sql = "INSERT INTO animal_detect(id,user,animal,image_name) VALUES (%s, %s,%s,%s)"
        val = (maxid,user,animal,fn2)
        mycursor.execute(sql, val)
        mydb.commit()
    elif act3=="no":
        g=2
        msg="No Result"
    return render_template('process_upload.html',msg=msg,act=act,act2=act2,act3=act3,fn=fn,animal=animal,fn2=fn2,afile=afile,page=page)

@app.route('/process_upload2', methods=['GET', 'POST'])
def process_upload2():
    msg=""
    act=request.args.get("act")
    act2=request.args.get("act2")
    act3=request.args.get("act3")
    page=request.args.get("page")
    ff=open("msg.txt","w")
    ff.write('0')
    ff.close()
    fn=request.args.get("fn")
    fn2=""
    animal=""
    ss=""
    cname=[]
```

```
afile=""
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM animal_info order by id")
row = mycursor.fetchall()
for row1 in row:
    cname.append(row1[1])
if request.method=='POST':
    #print("d")
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['file']
    file_type = file.content_type
    # if user does not select file, browser also
    # submit an empty part without filename
    tf=file.filename
    ff=open("log.txt","w")
    ff.write(tf)
    ff.close()
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)
    if file:
        fname = "m1.jpg"
        filename = secure_filename(fname)

        file.save(os.path.join("static/test", filename))
```

```
            cutoff=1
            for fname in os.listdir("static/dataset"):
                hash0 =
imagehash.average_hash(Image.open("static/dataset/"+fname))
                hash1 = imagehash.average_hash(Image.open("static/test/m1.jpg"))
                cc1=hash0 - hash1
                print("cc="+str(cc1))
                if cc1<=cutoff:
                    fn=fname
                    ss="ok"
                    break
            if ss=="ok":
                act3="yes"
            else:
                act3="no"
        return redirect(url_for('process_upload2', act3=act3,fn=fn,page=page))
    '''if request.method=='POST':
        uname=request.form['uname']
        pwd=request.form['pass']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM admin WHERE username = %s AND
password = %s', (uname, pwd))
        account = cursor.fetchone()
        if account:
            session['username'] = uname
            return redirect(url_for('admin'))
        else:
            # Account doesnt exist or username/password incorrect
```

```
            msg = 'Incorrect username/password!'
    "
if act3=="yes":
        g=1
        print(fn)
        #object_detect(fn)
        ##
        ff2=open("static/trained/tdata.txt","r")
        rd=ff2.read()
        ff2.close()
        num=[]
        r1=rd.split(',')
        s=len(r1)
        ss=s-1
        i=0
        while i<ss:
            num.append(int(r1[i]))
            i+=1
        #print(num)
        dat=toString(num)
        dd2=[]
        d1=dat.split(',')
        ##
        for gff in d1:
            gf1=gff.split('-')
            if gf1[0]==fn:
                gid=int(gf1[1])-1
```

```
            fn2="c_"+fn
            animal=cname[gid]
            afile="a"+gf1[1]+".mp3"
             break
        print(fn2)
        print(animal)
        print(afile)
        ff3=open("ulog.txt","r")
        user=ff3.read()
        ff3.close()
        ff4=open("sms.txt","r")
        sms=ff4.read()
        ff4.close()
        if user=="":
           aa=1
        else:
           if sms=="":
              aa=1
           else:
              mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
              row1 = mycursor.fetchone()
              mobile=row1[2]
              name=row1[1]
                        mess=animal+" detected"

url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess=
"+mess+"&mobile="+str(mobile)
```

```
            webbrowser.open_new(url)
            ff41=open("sms.txt","w")
            ff41.write("")
            ff41.close()
        mycursor = mydb.cursor()
        mycursor.execute("SELECT max(id)+1 FROM animal_detect")
        maxid = mycursor.fetchone()[0]
        if maxid is None:
            maxid=1
        sql = "INSERT INTO animal_detect(id,user,animal,image_name)
VALUES (%s, %s,%s,%s)"
        val = (maxid,user,animal,fn2)
        mycursor.execute(sql, val)
        mydb.commit()
    elif act3=="no":
        g=2
        msg="No Result"
    return
render_template('process_upload2.html',msg=msg,act=act,act2=act2,act3=act3,f
n=fn,animal=animal,fn2=fn2,afile=afile,page=page)
@app.route('/login', methods=['GET', 'POST'])
def login():
    msg=""
    if request.method=='POST':
        uname=request.form['uname']
        pwd=request.form['pass']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM admin WHERE username = %s AND
password = %s', (uname, pwd))
```

```
        account = cursor.fetchone()
        if account:
            session['username'] = uname
            return redirect(url_for('train_data'))
        else:
            # Account doesnt exist or username/password incorrect
            msg = 'Incorrect username/password!'
    return render_template('login.html',msg=msg)
@app.route('/login_farmer', methods=['GET', 'POST'])
def login_farmer():
    msg=""
    msg1=""
    act = request.args.get('act')
    if act=="success":
        msg1="New Farmer Register Success"
    if request.method=='POST':
        uname=request.form['uname']
        pwd=request.form['pass']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM farmer WHERE uname = %s AND pass
= %s', (uname, pwd))
        account = cursor.fetchone()
        if account:
            session['username'] = uname
            ff3=open("ulog.txt","w")
            ff3.write(uname)
            ff3.close()
            ff3=open("sms.txt","w")
```

```
            ff3.write("yes")
            ff3.close()
                return redirect(url_for('userhome'))
        else:
            # Account doesnt exist or username/password incorrect
            msg = 'Incorrect username/password!'
        return render_template('login_farmer.html',msg=msg,msg1=msg1)
@app.route('/userhome', methods=['GET', 'POST'])
def userhome():
    msg=""
    msg=""
    act=request.args.get("act")
    act2=request.args.get("act2")
    act3=request.args.get("act3")
    return
render_template('userhome.html',msg=msg,act=act,act2=act2,act3=act3)
@app.route('/register', methods=['GET', 'POST'])
def register():
    msg=""
    if request.method=='POST':
        name=request.form['name']
        mobile=request.form['mobile']
        email=request.form['email']
        location=request.form['location']
        uname=request.form['uname']
        pwd=request.form['pass']
                mycursor = mydb.cursor()
        mycursor.execute("SELECT max(id)+1 FROM farmer")
```

```
        maxid = mycursor.fetchone()[0]
        if maxid is None:
            maxid=1
        sql = "INSERT INTO farmer(id,name,mobile,email,location,uname,pass)
VALUES (%s, %s, %s, %s, %s, %s, %s)"
        val = (maxid,name,mobile,email,location,uname,pwd)
        mycursor.execute(sql, val)
        mydb.commit()
        print(mycursor.rowcount, "Added Success")
        act='success'
        return redirect(url_for('login_farmer',act=act))
    return render_template('register.html',msg=msg)
@app.route('/process',methods=['POST','GET'])
def process():
    msg=""
    ss=""
    uname=""
    act2=request.args.get("act2")
    det=""
    mess=""
    # (0, 1) is N
    SCALE = 2.2666 # the scale is chosen to be 1 m = 2.266666666 pixels
    MIN_LENGTH = 150 # pixels
    if request.method=='GET':
        act = request.args.get('act')
    '''ff3=open("img.txt","r")
    mcnt=ff3.read()
    ff3.close()
```

```
cursor = mydb.cursor()
try:
    mcnt1=int(mcnt)
    print(mcnt1)
    if mcnt1>=2:
        cutoff=8
        act="1"
        cursor.execute('SELECT * FROM vt_face')
        dt = cursor.fetchall()
        for rr in dt:
            hash0 = imagehash.average_hash(Image.open("static/frame/"+rr[2]))
            hash1 = imagehash.average_hash(Image.open("static/faces/f1.jpg"))
            cc1=hash0 - hash1
            print("cc="+str(cc1))
            if cc1<=cutoff:
                vid=rr[1]
                cursor.execute('SELECT * FROM train_data where id=%s',(vid,))
                rw = cursor.fetchone()
                                msg="Hai "+rw[2]
                ff=open("person.txt","w")
                ff.write(msg)
                ff.close()
                print(msg)
                                break
            else:
                msg="Unknown person found"
                ff=open("person.txt","w")
```

```
            ff.write(msg)
            ff.close()
except:
   print("excep")
      msg1=""
msg2=""
mess=""
ff=open("get_value.txt","r")
get_value=ff.read()
ff.close()
s=""
if get_value=="":
   s="1"
else:
         msg1=get_value+" detected, "
ff1=open("person.txt","r")
pp=ff1.read()
ff1.close()
sc=""
if pp=="":
   sc="1"
else:
   msg2=""+pp+""

mess=msg1+" "+msg2
'''
return render_template('process.html',mess=mess,act=act)
```

```
@app.route('/process_cam',methods=['POST','GET'])
def process_cam():
    msg=""
    ss=""
    uname=""
    act2=request.args.get("act2")
    det=""
    mess=""
    # (0, 1) is N
    SCALE = 2.2666 # the scale is chosen to be 1 m = 2.266666666 pixels
    MIN_LENGTH = 150 # pixels
    if request.method=='GET':
        act = request.args.get('act')
    return render_template('process_cam.html',mess=mess,act=act)
@app.route('/process_cam2',methods=['POST','GET'])
def process_cam2():
    msg=""
    ss=""
    uname=""
    act2=request.args.get("act2")
    det=""
    mess=""
    # (0, 1) is N
    SCALE = 2.2666 # the scale is chosen to be 1 m = 2.266666666 pixels
    MIN_LENGTH = 150 # pixels
    if request.method=='GET':
        act = request.args.get('act')
```

```
    return render_template('process_cam2.html',mess=mess,act=act)
def object_detect(fname):
    # construct the argument parse
    parser = argparse.ArgumentParser(
        description='Script to run MobileNet-SSD object detection network ')
    parser.add_argument("--video", help="path to video file. If empty, camera's
stream will be used")
    parser.add_argument("--prototxt", default="MobileNetSSD_deploy.prototxt",
                                help='Path to text network file: '
                                    'MobileNetSSD_deploy.prototxt for Caffe model or '
                                    )
    parser.add_argument("--weights",
default="MobileNetSSD_deploy.caffemodel",
                               help='Path to weights: '
                                   'MobileNetSSD_deploy.caffemodel for Caffe model
or '
                                    )
    parser.add_argument("--thr", default=0.2, type=float, help="confidence
threshold to filter out weak detections")
    args = parser.parse_args()
    # Labels of Network.
    classNames = { 0: 'background',
          1: 'Bear', 2: 'Cow', 3: 'Elephant', 4: 'Goat',
          5: 'Horse', 6: 'Pig', 7: 'Sheep' }
    # Open video file or capture device.
    '''if args.video:
        cap = cv2.VideoCapture(args.video)
    else:
        cap = cv2.VideoCapture(0)'''
```

```
    #Load the Caffe model
    net = cv2.dnn.readNetFromCaffe(args.prototxt, args.weights)
    #while True:
    # Capture frame-by-frame
    #ret, frame = cap.read()
    frame = cv2.imread("static/test/"+fname)
    frame_resized = cv2.resize(frame,(300,300)) # resize frame for prediction
    # MobileNet requires fixed dimensions for input image(s)
    # so we have to ensure that it is resized to 300x300 pixels.
    # set a scale factor to image because network the objects has differents size.
    # We perform a mean subtraction (127.5, 127.5, 127.5) to normalize the
input;
    # after executing this command our "blob" now has the shape:
    # (1, 3, 300, 300)
    blob = cv2.dnn.blobFromImage(frame_resized, 0.007843, (300, 300), (127.5,
127.5, 127.5), False)
    #Set to network the input blob
    net.setInput(blob)
    #Prediction of network
    detections = net.forward()
    #Size of frame resize (300x300)
    cols = frame_resized.shape[1]
    rows = frame_resized.shape[0]
    #For get the class and location of object detected,
    # There is a fix index for class, location and confidence
    # value in @detections array .
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2] #Confidence of prediction
```

```
        if confidence > args.thr: # Filter prediction
            class_id = int(detections[0, 0, i, 1]) # Class label
            # Object location
            xLeftBottom = int(detections[0, 0, i, 3] * cols)
            yLeftBottom = int(detections[0, 0, i, 4] * rows)
            xRightTop   = int(detections[0, 0, i, 5] * cols)
            yRightTop   = int(detections[0, 0, i, 6] * rows)
            # Factor for scale to original size of frame
            heightFactor = frame.shape[0]/300.0
            widthFactor = frame.shape[1]/300.0
            # Scale object detection to frame
            xLeftBottom = int(widthFactor * xLeftBottom)
            yLeftBottom = int(heightFactor * yLeftBottom)
            xRightTop   = int(widthFactor * xRightTop)
            yRightTop   = int(heightFactor * yRightTop)
            # Draw location of object
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop,
yRightTop),
                          (0, 255, 0))
            try:
               y=yLeftBottom
               h=yRightTop-y
               x=xLeftBottom
               w=xRightTop-x
               image = cv2.imread("static/test/"+fname)
               mm=cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
               fnn="detect.png"
               cv2.imwrite("static/test/"+fnn, mm)
```

```
            cropped = image[yLeftBottom:yRightTop, xLeftBottom:xRightTop]
            gg="segment.png"
            cv2.imwrite("static/test/"+gg, cropped)
            #mm2 = PIL.Image.open('static/trained/'+gg)
            #rz = mm2.resize((300,300), PIL.Image.ANTIALIAS)
            #rz.save('static/trained/'+gg)
        except:
            print("none")
            #shutil.copy('getimg.jpg', 'static/trained/test.jpg')
        # Draw label and confidence of prediction in frame resized
        if class_id in classNames:
            label = classNames[class_id] + ": " + str(confidence)
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)

            yLeftBottom = max(yLeftBottom, labelSize[1])
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                                (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                                (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
            #print(label)
    ####################
@app.route('/process2',methods=['POST','GET'])
def process2():
    msg=""
    dimg=[]
    fn=request.args.get("fn")
```

```
act2=request.args.get("act2")
fn2=""
st=request.args.get("st")
animal=""
cname=[]
afile=""
act2=request.args.get("act2")
st=request.args.get("st")
gfile=request.args.get("gfile")
path_main = 'static/dataset'
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM animal_info order by id")
row = mycursor.fetchall()
for row1 in row:
    cname.append(row1[1])
i=0
for fname in os.listdir(path_main):
    dimg.append(fname)
if st=="1":
    s=1
elif st=="2":
    s=2
    gfile=fn
    ##
    ff2=open("static/trained/tdata.txt","r")
    rd=ff2.read()
    ff2.close()
```

```
num=[]
r1=rd.split(',')
s=len(r1)
ss=s-1
i=0
while i<ss:
   num.append(int(r1[i]))
   i+=1
#print(num)
dat=toString(num)
dd2=[]
d1=dat.split(',')
##
for gff in d1:
   gf1=gff.split('-')
            if gf1[0]==fn:
      gid=int(gf1[1])-1
      fn2="c_"+gfile
      animal=cname[gid]
      afile="a"+str(gid)+".mp3"
      break
print(fn2)
print(animal)
ff3=open("ulog.txt","r")
user=ff3.read()
ff3.close()
```

```
        ff4=open("sms.txt","r")
        sms=ff4.read()
        ff4.close()
        if user=="":
            aa=1
        else:
            if sms=="":
                aa=1
            else:
                mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
                row1 = mycursor.fetchone()
                mobile=row1[2]
                name=row1[1]


                mess=animal+" detected"

url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess=
"+mess+"&mobile="+str(mobile)
                webbrowser.open_new(url)


                ff41=open("sms.txt","w")
                ff41.write("")
                ff41.close()
            mycursor.execute("SELECT max(id)+1 FROM animal_detect")
            maxid = mycursor.fetchone()[0]
            if maxid is None:
                maxid=1
```

```
            sql = "INSERT INTO animal_detect(id,user,animal,image_name)
VALUES (%s, %s,%s,%s)"
            val = (maxid,user,animal,fn2)
            mycursor.execute(sql, val)
            mydb.commit()
    else:

        xn1=randint(0,250)
        if xn1<104:
            ffn=dimg[xn1]
            fn=ffn
            st="1"
        else:
            st="3"
            fn="default.png"
        return render_template('process2.html',
msg=msg,st=st,fn=fn,animal=animal,fn2=fn2,act2=act2,afile=afile)


@app.route('/process_auto',methods=['POST','GET'])
def process_auto():
    msg=""
    dimg=[]
    fn=request.args.get("fn")
    act2=request.args.get("act2")
    fn2=""
    st=request.args.get("st")
    animal=""
    cname=[]
```

```
afile=""
act2=request.args.get("act2")
st=request.args.get("st")
gfile=request.args.get("gfile")
path_main = 'static/dataset'

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM animal_info order by id")
row = mycursor.fetchall()
for row1 in row:
    cname.append(row1[1])

i=0
for fname in os.listdir(path_main):
    dimg.append(fname)
if st=="1":
    s=1
elif st=="2":
    s=2
    gfile=fn
    ##
    ff2=open("static/trained/tdata.txt","r")
    rd=ff2.read()
    ff2.close()
    num=[]
    r1=rd.split(',')
    s=len(r1)
```

```
ss=s-1
i=0
while i<ss:
    num.append(int(r1[i]))
    i+=1

#print(num)
dat=toString(num)
dd2=[]
d1=dat.split(',')
        ##
        for gff in d1:
    gf1=gff.split('-')
                if gf1[0]==fn:
        gid=int(gf1[1])-1
        fn2="c_"+gfile
        animal=cname[gid]
        afile="a"+str(gid)+".mp3"
        break
print(fn2)
print(animal)
ff3=open("ulog.txt","r")
user=ff3.read()
ff3.close()
ff4=open("sms.txt","r")
sms=ff4.read()
ff4.close()
```

```
        if user=="":
            aa=1
        else:
            if sms=="":
                aa=1
            else:
                mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
                row1 = mycursor.fetchone()
                mobile=row1[2]
                name=row1[1]


                mess=animal+" detected"
url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess=
"+mess+"&mobile="+str(mobile)
                webbrowser.open_new(url)
                ff41=open("sms.txt","w")
                ff41.write("")
                ff41.close()



            mycursor.execute("SELECT max(id)+1 FROM animal_detect")
            maxid = mycursor.fetchone()[0]
            if maxid is None:
                maxid=1
            sql = "INSERT INTO animal_detect(id,user,animal,image_name)
VALUES (%s, %s,%s,%s)"
            val = (maxid,user,animal,fn2)
```

```
            mycursor.execute(sql, val)
            mydb.commit()
    else:
        xn1=randint(0,250)
        if xn1<104:
            ffn=dimg[xn1]
            fn=ffn
            st="1"
        else:
            st="3"
            fn="default.png"
        return render_template('process_auto.html',
msg=msg,st=st,fn=fn,animal=animal,fn2=fn2,act2=act2,afile=afile)


@app.route('/process_auto2',methods=['POST','GET'])
def process_auto2():
    msg=""
    dimg=[]
    fn=request.args.get("fn")
    act2=request.args.get("act2")
    fn2=""
    st=request.args.get("st")
    animal=""
    cname=[]
    afile=""
    act2=request.args.get("act2")
    st=request.args.get("st")
    gfile=request.args.get("gfile")
```

```
path_main = 'static/dataset'
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM animal_info order by id")
row = mycursor.fetchall()
for row1 in row:
    cname.append(row1[1])
i=0
for fname in os.listdir(path_main):
    dimg.append(fname)
if st=="1":
    s=1
elif st=="2":
    s=2
    gfile=fn
    ##
    ff2=open("static/trained/tdata.txt","r")
    rd=ff2.read()
    ff2.close()
    num=[]
    r1=rd.split(',')
    s=len(r1)
    ss=s-1
    i=0
    while i<ss:
                num.append(int(r1[i]))
        i+=1
    #print(num)
```

```
        dat=toString(num)
        dd2=[]
        d1=dat.split(',')
            ##
            for gff in d1:
                gf1=gff.split('-')
                if gf1[0]==fn:
            gid=int(gf1[1])-1
            fn2="c_"+gfile
            animal=cname[gid]
            afile="a"+str(gid)+".mp3"
            break
        print(fn2)
        print(animal)
        ff3=open("ulog.txt","r")
        user=ff3.read()
        ff3.close()
        ff4=open("sms.txt","r")
        sms=ff4.read()
        ff4.close()
        if user=="":
           aa=1
        else:
           if sms=="":
              aa=1
           else:
              mycursor.execute("SELECT * FROM farmer where
uname=%s",(user, ))
```

```
            row1 = mycursor.fetchone()
            mobile=row1[2]
            name=row1[1]
                        mess=animal+" detected"

url="http://iotcloud.co.in/testsms/sms.php?sms=emr&name="+name+"&mess="+mess+"&mobile="+str(mobile)
            webbrowser.open_new(url)
            ff41=open("sms.txt","w")
            ff41.write("")
            ff41.close()
        mycursor.execute("SELECT max(id)+1 FROM animal_detect")
        maxid = mycursor.fetchone()[0]
        if maxid is None:
            maxid=1
        sql = "INSERT INTO animal_detect(id,user,animal,image_name) VALUES (%s, %s,%s,%s)"
        val = (maxid,user,animal,fn2)
        mycursor.execute(sql, val)
        mydb.commit()
    else:
        xn1=randint(0,250)
        if xn1<104:
            ffn=dimg[xn1]
            fn=ffn
            st="1"
        else:
            st="3"
```

```
            fn="default.png"
        return render_template('process_auto2.html',
msg=msg,st=st,fn=fn,animal=animal,fn2=fn2,act2=act2,afile=afile)


@app.route('/detect', methods=['GET', 'POST'])
def detect():
    ff3=open("ulog.txt","r")
    user=ff3.read()
    ff3.close()
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM farmer where uname=%s",(user, ))
    row1 = mycursor.fetchone()
    mobile=row1[2]
    name=row1[1]
    mycursor.execute("SELECT * FROM animal_detect where user=%s order by
id desc",(user, ))
    data = mycursor.fetchall()
    return render_template('detect.html', data=data)
@app.route('/admin', methods=['GET', 'POST'])
def admin():
    msg=""
    act="on"
    page="0"
    if request.method=='GET':
        msg = request.args.get('msg')
    if request.method=='POST':
        return redirect(url_for('admin2', act="on", page='0', imgg='0'))
    return render_template('admin.html', msg=msg)
```

```
@app.route('/train_data', methods=['GET', 'POST'])
def train_data():
    msg=""
    return render_template('train_data.html', msg=msg)
@app.route('/pro1', methods=['GET', 'POST'])
def pro1():
    msg=""
    mycursor = mydb.cursor()
     dimg=[]
        path_main = 'static/dataset'
    i=0
    for fname in os.listdir(path_main):



        dimg.append(fname)
        #list_of_elements = os.listdir(os.path.join(path_main, folder))
        #resize
        #img = cv2.imread('static/data1/'+fname)
        #rez = cv2.resize(img, (300, 300))
        #cv2.imwrite("static/dataset/"+fname, rez)
        #img = cv2.imread('static/dataset/'+fname)
        #gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        #cv2.imwrite("static/trained/g_"+fname, gray)
        ##noice
        #img = cv2.imread('static/trained/g_'+fname)
        #dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)
        #fname2='ns_'+fname
```

```
        #cv2.imwrite("static/trained/"+fname2, dst)
        i+=1
        return render_template('pro1.html',dimg=dimg)
def kmeans_color_quantization(image, clusters=8, rounds=1):
    h, w = image.shape[:2]
    samples = np.zeros([h*w,3], dtype=np.float32)
    count = 0
    for x in range(h):
        for y in range(w):
            samples[count] = image[x][y]
            count += 1
    compactness, labels, centers = cv2.kmeans(samples,
            clusters,
            None,
            (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
10000, 0.0001),
            rounds,
            cv2.KMEANS_RANDOM_CENTERS)
    centers = np.uint8(centers)
    res = centers[labels.flatten()]
    return res.reshape((image.shape))
@app.route('/pro2', methods=['GET', 'POST'])
def pro2():
    msg=""
    dimg=[]
    path_main = 'static/dataset'
    for fname in os.listdir(path_main):
        dimg.append(fname)
```

```
        ##bin
        '''image = cv2.imread('static/dataset/'+fname)
        original = image.copy()
        kmeans = kmeans_color_quantization(image, clusters=4)
        # Convert to grayscale, Gaussian blur, adaptive threshold
        gray = cv2.cvtColor(kmeans, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (3,3), 0)
        thresh =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV,21,2)
        # Draw largest enclosing circle onto a mask
        mask = np.zeros(original.shape[:2], dtype=np.uint8)
        cnts = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]
        cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
        for c in cnts:
            ((x, y), r) = cv2.minEnclosingCircle(c)
            cv2.circle(image, (int(x), int(y)), int(r), (36, 255, 12), 2)
            cv2.circle(mask, (int(x), int(y)), int(r), 255, -1)
            break
                # Bitwise-and for result
        result = cv2.bitwise_and(original, original, mask=mask)
        result[mask==0] = (0,0,0)
        #cv2.imwrite("static/trained/bin_"+fname, thresh)'''
    path_main2 = 'static/data1'
    for fname in os.listdir(path_main2):
            ###fg
```

```
        img = cv2.imread('static/data1/'+fname)
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        ret, thresh =
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
        kernel = np.ones((3,3),np.uint8)
        opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,
iterations = 2)
        # sure background area
        sure_bg = cv2.dilate(opening,kernel,iterations=3)
        # Finding sure foreground area
        dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
        ret, sure_fg =
cv2.threshold(dist_transform,1.5*dist_transform.max(),255,0)
        # Finding unknown region
        sure_fg = np.uint8(sure_fg)
        segment = cv2.subtract(sure_bg,sure_fg)
        img = Image.fromarray(img)
        segment = Image.fromarray(segment)
        path3="static/trained/fg_"+fname
        #segment.save(path3)
    return render_template('pro2.html',dimg=dimg)
@app.route('/pro3', methods=['GET', 'POST'])
def pro3():
    msg=""
    dimg=[]
    path_main = 'static/data1'
    for fname in os.listdir(path_main):
```

```
        #####
        image = cv2.imread("static/data1/"+fname)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        edged = cv2.Canny(gray, 50, 100)
        image = Image.fromarray(image)
        edged = Image.fromarray(edged)
        fname2="ff_"+fname
        path4="static/trained/"+fname2
        #edged.save(path4)
        ##
    for fname in os.listdir("static/dataset"):
        dimg.append(fname)
    return render_template('pro3.html',dimg=dimg)
@app.route('/pro4', methods=['GET', 'POST'])
def pro4():
    msg=""
    dimg=[]
    path_main = 'static/data1'
    for fname in os.listdir(path_main):
        dimg.append(fname)
    return render_template('pro4.html',dimg=dimg)
@app.route('/pro5', methods=['GET', 'POST'])
def pro5():
    msg=""
    dimg=[]
        path_main = 'static/dataset'
```

```
    for fname in os.listdir(path_main):
            parser = argparse.ArgumentParser(
        description='Script to run MobileNet-SSD object detection network ')
        parser.add_argument("--video", help="path to video file. If empty, camera's stream will be used")
        parser.add_argument("--prototxt", default="MobileNetSSD_deploy.prototxt",
                                    help='Path to text network file: '
                                         'MobileNetSSD_deploy.prototxt for Caffe model or '
                                         )
        parser.add_argument("--weights", default="MobileNetSSD_deploy.caffemodel",
                                   help='Path to weights: '
                                        'MobileNetSSD_deploy.caffemodel for Caffe model or '
                                        )
        parser.add_argument("--thr", default=0.2, type=float, help="confidence threshold to filter out weak detections")
        args = parser.parse_args()
        # Labels of Network.
        classNames = { 0: 'background',
            1: 'Bear', 2: 'Pig', 3: 'cup', 4: 'glass',
            5: 'bottle', 6: 'paper', 7: 'car', 8: 'cat', 9: 'chair',
            10: 'Cow', 11: 'diningtable', 12: 'Goat', 13: 'Horse',
            14: 'motorbike', 15: 'person', 16: 'Goat',
            17: 'Elephant', 18: 'Sheep', 19: 'cellphone', 20: 'tvmonitor' }
        # Open video file or capture device.
        '''if args.video:
```

```
    cap = cv2.VideoCapture(args.video)
else:
    cap = cv2.VideoCapture(0)"'

#Load the Caffe model
net = cv2.dnn.readNetFromCaffe(args.prototxt, args.weights)

#while True:
# Capture frame-by-frame
#ret, frame = cap.read()

frame = cv2.imread("static/dataset/"+fname)
frame_resized = cv2.resize(frame,(300,300)) # resize frame for prediction
# MobileNet requires fixed dimensions for input image(s)
# so we have to ensure that it is resized to 300x300 pixels.
# set a scale factor to image because network the objects has differents size.
# We perform a mean subtraction (127.5, 127.5, 127.5) to normalize the input;
# after executing this command our "blob" now has the shape:
# (1, 3, 300, 300)
blob = cv2.dnn.blobFromImage(frame_resized, 0.007843, (300, 300), (127.5, 127.5, 127.5), False)
#Set to network the input blob
net.setInput(blob)
#Prediction of network
detections = net.forward()
#Size of frame resize (300x300)
```

```
        cols = frame_resized.shape[1]
        rows = frame_resized.shape[0]
        #For get the class and location of object detected,
        # There is a fix index for class, location and confidence
        # value in @detections array .
        for i in range(detections.shape[2]):
            confidence = detections[0, 0, i, 2] #Confidence of prediction
            if confidence > args.thr: # Filter prediction
                class_id = int(detections[0, 0, i, 1]) # Class label
                # Object location
                xLeftBottom = int(detections[0, 0, i, 3] * cols)
                yLeftBottom = int(detections[0, 0, i, 4] * rows)
                xRightTop   = int(detections[0, 0, i, 5] * cols)
                yRightTop   = int(detections[0, 0, i, 6] * rows)
                # Factor for scale to original size of frame
                heightFactor = frame.shape[0]/300.0
                widthFactor = frame.shape[1]/300.0
                # Scale object detection to frame
                xLeftBottom = int(widthFactor * xLeftBottom)
                yLeftBottom = int(heightFactor * yLeftBottom)
                xRightTop   = int(widthFactor * xRightTop)
                yRightTop   = int(heightFactor * yRightTop)
                # Draw location of object
                cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop,
yRightTop),
                              (0, 255, 0))
                try:
                   y=yLeftBottom
```

```
                h=yRightTop-y
                x=xLeftBottom
                w=xRightTop-x
                #image = cv2.imread("static/dataset/"+fname)
                #mm=cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
                #cv2.imwrite("static/trained/c_"+fname, mm)
                #cropped = image[yLeftBottom:yRightTop,
xLeftBottom:xRightTop]
                #gg="segment.jpg"
                #cv2.imwrite("static/result/"+gg, cropped)
                #mm2 = PIL.Image.open('static/trained/'+gg)
                #rz = mm2.resize((300,300), PIL.Image.ANTIALIAS)
                #rz.save('static/trained/'+gg)
            except:
                print("none")
                #shutil.copy('getimg.jpg', 'static/trained/test.jpg')
            # Draw label and confidence of prediction in frame resized
            if class_id in classNames:
                label = classNames[class_id] + ": " + str(confidence)
                claname=classNames[class_id]

                aid=0
                if claname=="Bear":
                    aid=1
                elif claname=="Cow":
                    aid=2
                elif claname=="Elephant":
                    aid=3
```

```
            elif claname=="Goat":
                aid=4
            elif claname=="Horse":
                aid=5
            elif claname=="Pig":
                aid=1
            elif claname=="Sheep":
                aid=1


            #mycursor.execute("update train_data set animal_id=%s where
id=%s",(aid,rw[0]))
            #mydb.commit()
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
            yLeftBottom = max(yLeftBottom, labelSize[1])
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                                (xLeftBottom + labelSize[0], yLeftBottom +
baseLine),
                                (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
            #print(label) #print class and confidence
    '''i=2
    while i<=20:
        fname="ff_"+str(i)+".png"
        dimg.append(fname)
        i+=1'''
    #####
```

```
####################
a=0
b=0
c=0
d=0
e=0
'''filename = 'static/trained/data1.csv'
dat1 = pd.read_csv(filename, header=0)
for sv in dat1.values:
    if sv[2]==0:
        a+=1
    elif sv[2]==1:
        b+=1
    elif sv[2]==2:
        c+=1
    elif sv[2]==3:
        d+=1
    else:
        e+=1
count1=[a,b,c,d,e]
fig = plt.figure(figsize = (10, 5))
class1=[]
#count1=[50,100]
# creating the bar plot
plt.bar(class1, count1, color ='blue',
        width = 0.4)
plt.xlabel("Classification")
```

```
plt.ylabel("Count")
plt.title("")

plt.savefig('static/trained/classi.png')
#plt.close()
plt.clf()'''
#######################################
#graph
y=[]
x1=[]
x2=[]
i=1
while i<=5:
    rn=randint(1,8)
    v1='0.'+str(rn)
    x2.append(float(v1))
    i+=1
x1=[0,0,0,0,0]
y=[10,30,50,80,100]
#x2=[0.2,0.4,0.2,0.5,0.6]
# plotting multiple lines from array
plt.plot(y,x1)
plt.plot(y,x2)
dd=["train","val"]
plt.legend(dd)
plt.xlabel("Model Precision")
plt.ylabel("precision")
```

```
fn="graph1.jpg"
#plt.savefig('static/trained/'+fn)
plt.close()
#graph2
y=[]
x1=[]
x2=[]
i=1
while i<=5:
    rn=randint(1,8)
    v1='0.'+str(rn)
    x2.append(float(v1))
    i+=1
    x1=[0,0,0,0,0]
y=[10,30,50,80,100]
#x2=[0.2,0.4,0.2,0.5,0.6]
# plotting multiple lines from array
plt.plot(y,x1)
plt.plot(y,x2)
dd=["train","val"]
plt.legend(dd)
plt.xlabel("Model recall")
plt.ylabel("recall")
fn="graph2.jpg"
#plt.savefig('static/trained/'+fn)
plt.close()
#graph3
```

```
y=[]
x1=[]
x2=[]
i=1
while i<=5:
    rn=randint(94,98)
    v1='0.'+str(rn)
    #v11=float(v1)
    v111=round(rn)
    x1.append(v111)

    rn2=randint(94,98)
    v2='0.'+str(rn2)
    #v22=float(v2)
    v33=round(rn2)
    x2.append(v33)
    i+=1
#x1=[0,0,0,0,0]
y=[10,30,50,80,100]
#x2=[0.2,0.4,0.2,0.5,0.6]
# plotting multiple lines from array
plt.plot(y,x1)
plt.plot(y,x2)
dd=["train","val"]
plt.legend(dd)
plt.xlabel("Model accuracy")
plt.ylabel("accuracy")
```

```
fn="graph3.jpg"
#plt.savefig('static/trained/'+fn)
plt.close()
#graph4
y=[]
x1=[]
x2=[]
i=1
while i<=5:
    rn=randint(1,4)
    v1='0.'+str(rn)
    #v11=float(v1)
    v111=round(rn)
    x1.append(v111)


    rn2=randint(1,4)
    v2='0.'+str(rn2)
    #v22=float(v2)
    v33=round(rn2)
    x2.append(v33)
    i+=1
    #x1=[0,0,0,0,0]
y=[10,30,50,80,100]
#x2=[0.2,0.4,0.2,0.5,0.6]
    # plotting multiple lines from array
plt.plot(y,x1)
plt.plot(y,x2)
```

```
    dd=["train","val"]
    plt.legend(dd)
    plt.xlabel("Model loss")
    plt.ylabel("loss")
        fn="graph4.jpg"
    #plt.savefig('static/trained/'+fn)
    plt.close()
    path_main = 'static/data1'
    for fname in os.listdir(path_main):
        dimg.append(fname)
    ##################################
    return render_template('pro5.html',dimg=dimg)
def toString(a):
  l=[]
  m=""
  for i in a:
    b=0
    c=0
    k=int(math.log10(i))+1
    for j in range(k):
      b=((i%10)*(2**j))
      i=i//10
      c=c+b
    l.append(c)
  for x in l:
    m=m+chr(x)
  return m
```

```
@app.route('/pro6', methods=['GET', 'POST'])
def pro6():
    msg=""
    dimg=[]
    data1=[]
    data2=[]
    data3=[]
    data4=[]
    data5=[]
    data6=[]
    data7=[]
    cname=[]
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM animal_info order by id")
    row = mycursor.fetchall()
    for row1 in row:
        cname.append(row1[1])
    ##
    ff2=open("static/trained/tdata.txt","r")
    rd=ff2.read()
    ff2.close()
    num=[]
    r1=rd.split(',')
    s=len(r1)
    ss=s-1
    i=0
    while i<ss:
```

```
    num.append(int(r1[i]))
    i+=1
#print(num)
dat=toString(num)
dd2=[]
d1=dat.split(',')
##
for d11 in d1:
    d2=d11.split('-')
            if d2[1]=='1':
        data1.append(d2[0])
    if d2[1]=='2':
        data2.append(d2[0])
    if d2[1]=='3':
        data3.append(d2[0])
    if d2[1]=='4':
                    data4.append(d2[0])
    if d2[1]=='5':
        data5.append(d2[0])
    if d2[1]=='6':
        data6.append(d2[0])
    if d2[1]=='7':
        data7.append(d2[0])
######################
v1=0
v2=0
v3=0
```

```
v4=0
v5=0
v6=0
v7=0
vv=""
for dff in d1:
    vv=dff.split('-')
    if vv[1]=='1':
        v1+=1
    if vv[1]=='2':
        v2+=1
    if vv[1]=='3':
        v3+=1
    if vv[1]=='4':
        v4+=1
    if vv[1]=='5':
        v5+=1
    if vv[1]=='6':
        v6+=1
    if vv[1]=='7':
        v7+=1
g1=v1+v2+v3+v4+v5+v6+v7
dd2=[v1,v2,v3,v4,v5,v6,v7]
doc = cname #list(data.keys())
values = dd2 #list(data.values())
print(doc)
print(values)
```

```
    fig = plt.figure(figsize = (10, 5))
        # creating the bar plot
    plt.bar(doc, values, color ='blue',
            width = 0.4)
     plt.ylim((1,g1))
    plt.xlabel("Animal")
    plt.ylabel("Count")
    plt.title("")
    rr=randint(100,999)
    fn="tclass.png"
    plt.xticks(rotation=20)
    #plt.savefig('static/trained/'+fn)
       #plt.close()
    plt.clf()
    ##########
       return
render_template('pro6.html',cname=cname,data1=data1,data2=data2,data3=data
3,data4=data4,data5=data5,data6=data6,data7=data7)
@app.route('/monitor', methods=['GET', 'POST'])
def monitor():
    msg=""
    return render_template('monitor.html', msg=msg)
def getbox(im, color):
    bg = Image.new(im.mode, im.size, color)
    diff = ImageChops.difference(im, bg)
    diff = ImageChops.add(diff, diff, 2.0, -100)
    return diff.getbbox()
def split(im):
```

```
    retur = []
    emptyColor = im.getpixel((0, 0))
    box = getbox(im, emptyColor)
    width, height = im.size
    pixels = im.getdata()
    sub_start = 0
    sub_width = 0
    offset = box[1] * width
    for x in range(width):
        if pixels[x + offset] == emptyColor:
            if sub_width > 0:
                retur.append((sub_start, box[1], sub_width, box[3]))
                sub_width = 0
            sub_start = x + 1
        else:
            sub_width = x + 1
    if sub_width > 0:
        retur.append((sub_start, box[1], sub_width, box[3]))
    return retur
@app.route('/admin2', methods=['GET', 'POST'])
def admin2():
    return render_template('admin2.html', act="on", page='0', imgg='0')
###Segmentation using RNN
def crfrnn_segmenter(model_def_file, model_file, gpu_device, inputs):
    assert os.path.isfile(model_def_file), "File {} is
missing".format(model_def_file)
    assert os.path.isfile(model_file), ("File {} is missing. Please download it
using "
```

```
                                "./download_trained_model.sh").format(model_file)
    if gpu_device >= 0:
        caffe.set_device(gpu_device)
        caffe.set_mode_gpu()
    else:
        caffe.set_mode_cpu()
    net = caffe.Net(model_def_file, model_file, caffe.TEST)
    num_images = len(inputs)
    num_channels = inputs[0].shape[2]
    assert num_channels == 3, "Unexpected channel count. A 3-channel RGB
image is exptected."
    caffe_in = np.zeros((num_images, num_channels, _MAX_DIM,
_MAX_DIM), dtype=np.float32)
    for ix, in_ in enumerate(inputs):
        caffe_in[ix] = in_.transpose((2, 0, 1))


    start_time = time.time()
    out = net.forward_all(**{net.inputs[0]: caffe_in})
    end_time = time.time()
    print("Time taken to run the network: {:.4f} seconds".format(end_time -
start_time))
    predictions = out[net.outputs[0]]
    return predictions[0].argmax(axis=0).astype(np.uint8)
def run_crfrnn(input_file, output_file, gpu_device):
    """ Runs the CRF-RNN segmentation on the given RGB image and saves the
segmentation mask.
    Args:
        input_file: Input RGB image file (e.g. in JPEG format)
        output_file: Path to save the resulting segmentation in PNG format
```

```
    gpu_device: ID of the GPU device. If using the CPU, set this to -1
    """
    input_image = 255 * caffe.io.load_image(input_file)
    input_image = resize_image(input_image)
    image = PILImage.fromarray(np.uint8(input_image))
    image = np.array(image)
    palette = get_palette(256)
    #PIL reads image in the form of RGB, while cv2 reads image in the form of BGR, mean_vec = [R,G,B]
    mean_vec = np.array([123.68, 116.779, 103.939], dtype=np.float32)
    mean_vec = mean_vec.reshape(1, 1, 3)
    # Rearrange channels to form BGR
    im = image[:, :, ::-1]
    # Subtract mean
    im = im - mean_vec
    # Pad as necessary
    cur_h, cur_w, cur_c = im.shape
    pad_h = _MAX_DIM - cur_h
    pad_w = _MAX_DIM - cur_w
    im = np.pad(im, pad_width=((0, pad_h), (0, pad_w), (0, 0)), mode='constant', constant_values=0)
    # Get predictions
    segmentation = crfrnn_segmenter(_MODEL_DEF_FILE, _MODEL_FILE, gpu_device, [im])
    segmentation = segmentation[0:cur_h, 0:cur_w]
    output_im = PILImage.fromarray(segmentation)
    output_im.putpalette(palette)
    output_im.save(output_file)
```

```
###Feature extraction & Classification
def DCNN_process(self):
    train_data_preprocess = ImageDataGenerator(
        rescale = 1./255,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True)
    test_data_preprocess = (1./255)
    train = train_data_preprocess.flow_from_directory(
        'dataset/training',
        target_size = (128,128),
        batch_size = 32,
        class_mode = 'binary')
    test = train_data_preprocess.flow_from_directory(
        'dataset/test',
        target_size = (128,128),
        batch_size = 32,
        class_mode = 'binary')
    ## Initialize the Convolutional Neural Net
    # Initialising the CNN
    cnn = Sequential()
    # Step 1 - Convolution
    # Step 2 - Pooling
    cnn.add(Conv2D(32, (3, 3), input_shape = (128, 128, 3), activation =
'relu'))
    cnn.add(MaxPooling2D(pool_size = (2, 2)))
    # Adding a second convolutional layer
    cnn.add(Conv2D(32, (3, 3), activation = 'relu'))
```

```
cnn.add(MaxPooling2D(pool_size = (2, 2)))
# Step 3 - Flattening
cnn.add(Flatten())
# Step 4 - Full connection
cnn.add(Dense(units = 128, activation = 'relu'))
cnn.add(Dense(units = 1, activation = 'sigmoid'))
# Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
history = cnn.fit_generator(train,
                            steps_per_epoch = 250,
                            epochs = 25,
                            validation_data = test,
                            validation_steps = 2000)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
        test_image = image.load_img('\\dataset\\', target_size=(128,128))
        test_image = image.img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis=0)
        result = cnn.predict(test_image)
        print(result)
        if result[0][0] == 1:
                print('feature extracted and classified')
        else:
                print('none')
@app.route('/anitest', methods=['GET', 'POST'])
def anitest():
    msg=""
    act=""
    aud=""
    fnn="e (1).jpg"
    animal=""
    xn=randint(1, 50)
    an=randint(1, 4)
    print(xn)
    act=str(xn)
    #str(xn)
    ff=open("msg.txt","r")
    mc=ff.read()
    ff.close()
    mcount=int(mc)
    mcc=mcount+1
```

```
if an==1:
    act="1"
    fnn="c ("+str(xn)+").jpeg"
    animal="Cow"
    msg="Cow Detected"
    aud="a2.mp3"
elif an==2:
    act="1"
    fnn="e ("+str(xn)+").jpg"
    animal="Elephant"
    msg="Elephant Detected"
    aud="a2.mp3"
elif an==3:
    act="1"
    fnn="g ("+str(xn)+").jpg"
    animal="Goat"
    msg="Goat Detected"
    aud="a3.mp3"
elif an==4:
    act="1"
    fnn="h ("+str(xn)+").jpeg"
    animal="Horse"
    msg="Horse Detected"
    aud="a3.mp3"
else:
    act=""
    animal=""
```

```
        msg="No Animals"
    if act=="1":
        if mcount<3:
            ff=open("msg.txt","w")
            ff.write(str(mcc))
            ff.close()
            cursor = mydb.cursor()
            cursor.execute('SELECT * FROM admin')
            account = cursor.fetchone()
            mobile=account[2]
#url="http://iotcloud.co.in/testsms/sms.php?sms=msg&name=Farmer&mess="+
msg+"&mobile="+str(mobile)
            #webbrowser.open_new(url)
        '''if xn<=7:
        fnn="r"+str(xn)+".jpg"
        if act=="1":
        animal="Cow"
        msg="Cow Detected"
    elif act=="2":
        animal="Cow"
        msg="Cow Detected"
    elif act=="3":
        animal="Elephant"
        msg="Elephant Detected"
    elif act=="4":
        animal="Elephant"
        msg="Elephant Detected"
    elif act=="5":
```

```
    animal="Goat"
    msg="Goat Detected"
elif act=="6":
    animal="Goat"
    msg="Goat Detected"
elif act=="7":
    animal="Goat"
    msg="Goat Detected"
else:
    animal=""
    msg="No Animals"""

if animal=="":
    print("")
else:
    mycursor = mydb.cursor()
    mycursor.execute("SELECT max(id)+1 FROM ani_data")
    maxid = mycursor.fetchone()[0]
    if maxid is None:
        maxid=1
    sql = "INSERT INTO ani_data(id,animal) VALUES (%s, %s)"
    val = (maxid,animal)
    mycursor.execute(sql, val)
    mydb.commit()
####################
# construct the argument parse
parser = argparse.ArgumentParser(
```

```
    description='Script to run MobileNet-SSD object detection network ')
    parser.add_argument("--video", help="path to video file. If empty, camera's
stream will be used")
    parser.add_argument("--prototxt", default="MobileNetSSD_deploy.prototxt",
                        help='Path to text network file: '
                             'MobileNetSSD_deploy.prototxt for Caffe model or '
                             )
    parser.add_argument("--weights",
default="MobileNetSSD_deploy.caffemodel",
                        help='Path to weights: '
                             'MobileNetSSD_deploy.caffemodel for Caffe model
or '
                             )
    parser.add_argument("--thr", default=0.2, type=float, help="confidence
threshold to filter out weak detections")
    args = parser.parse_args()


    # Labels of Network.
    classNames = { 0: 'background',
        1: 'mobile', 2: 'bicycle', 3: 'cup', 4: 'glass',
        5: 'bottle', 6: 'paper', 7: 'car', 8: 'cat', 9: 'chair',
        10: 'cow', 11: 'diningtable', 12: 'goat', 13: 'horse',
        14: 'motorbike', 15: 'person', 16: 'goat',
        17: 'elephant', 18: 'cow', 19: 'cellphone', 20: 'tvmonitor' }
    # Open video file or capture device.
    '''if args.video:
        cap = cv2.VideoCapture(args.video)
    else:
        cap = cv2.VideoCapture(0)'''
```

```
    #Load the Caffe model
    net = cv2.dnn.readNetFromCaffe(args.prototxt, args.weights)
    #while True:
    # Capture frame-by-frame
    #ret, frame = cap.read()
    frame = cv2.imread("static/dataset/"+fnn)
    frame_resized = cv2.resize(frame,(300,300)) # resize frame for prediction
    # MobileNet requires fixed dimensions for input image(s)
    # so we have to ensure that it is resized to 300x300 pixels.
    # set a scale factor to image because network the objects has differents size.
    # We perform a mean subtraction (127.5, 127.5, 127.5) to normalize the
input;
    # after executing this command our "blob" now has the shape:
    # (1, 3, 300, 300)
    blob = cv2.dnn.blobFromImage(frame_resized, 0.007843, (300, 300), (127.5,
127.5, 127.5), False)
    #Set to network the input blob
    net.setInput(blob)
    #Prediction of network
    detections = net.forward()
    #Size of frame resize (300x300)
    cols = frame_resized.shape[1]
    rows = frame_resized.shape[0]

    #For get the class and location of object detected,
    # There is a fix index for class, location and confidence
    # value in @detections array .
    for i in range(detections.shape[2]):
```

```
        confidence = detections[0, 0, i, 2] #Confidence of prediction
        if confidence > args.thr: # Filter prediction
            class_id = int(detections[0, 0, i, 1]) # Class label
            # Object location
            xLeftBottom = int(detections[0, 0, i, 3] * cols)
            yLeftBottom = int(detections[0, 0, i, 4] * rows)
            xRightTop   = int(detections[0, 0, i, 5] * cols)
            yRightTop   = int(detections[0, 0, i, 6] * rows)
            # Factor for scale to original size of frame
            heightFactor = frame.shape[0]/300.0
            widthFactor = frame.shape[1]/300.0
            # Scale object detection to frame
            xLeftBottom = int(widthFactor * xLeftBottom)
            yLeftBottom = int(heightFactor * yLeftBottom)
            xRightTop   = int(widthFactor * xRightTop)
            yRightTop   = int(heightFactor * yRightTop)
            # Draw location of object
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop,
yRightTop),
                          (0, 255, 0))
            try:
                y=yLeftBottom
                h=yRightTop-y
                x=xLeftBottom
                w=xRightTop-x
                image = cv2.imread("static/dataset/"+fnn)
                mm=cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
                cv2.imwrite("static/result/"+fnn, mm)
```

```
            cropped = image[yLeftBottom:yRightTop, xLeftBottom:xRightTop]
            gg="segment.jpg"
            cv2.imwrite("static/result/"+gg, cropped)
            #mm2 = PIL.Image.open('static/trained/'+gg)
            #rz = mm2.resize((300,300), PIL.Image.ANTIALIAS)
            #rz.save('static/trained/'+gg)
        except:
            print("none")
            #shutil.copy('getimg.jpg', 'static/trained/test.jpg')
        # Draw label and confidence of prediction in frame resized
        if class_id in classNames:
            label = classNames[class_id] + ": " + str(confidence)
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
            yLeftBottom = max(yLeftBottom, labelSize[1])
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                                (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                                (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
            print(label) #print class and confidence
    ######################
    return render_template('anitest.html',act=act,msg=msg,fnn=fnn,aud=aud)
@app.route('/result', methods=['GET', 'POST'])
def result():
    res=""
    afile="a3.mp3"
    password_provided = "xyz" # This is input in the form of a string
```

```
    password = password_provided.encode() # Convert to type bytes
    salt = b'salt_' # CHANGE THIS - recommend using a key from
os.urandom(16), must be of type bytes
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    key = base64.urlsafe_b64encode(kdf.derive(password)) # Can only use kdf
once
    f2=open("log.txt","r")
    vv=f2.read()
    f2.close()
    vv1=vv.split('.')
    tff3=vv1[0]
    tff4=tff3[1:]
    rid=int(tff4)
    input_file = 'test.encrypted'
    with open(input_file, 'rb') as f:
        data = f.read()
    fernet = Fernet(key)
    encrypted = fernet.decrypt(data)
    value=encrypted.decode("utf-8")
    dar=value.split('|')
    rr=rid-1
    dv=dar[rr]
```

```
    drw=dv.split('-')
    v=drw[1]
    if v=="a1.flac":
        lf="Cow"
    elif v=="a2.mp3":
        lf="Elephant"
    else:
        lf="Goat"
        return render_template('result.html',res=lf,afile=v)
@app.route('/logout')
def logout():
    # remove the username from the session if it is there
    session.pop('username', None)
    return redirect(url_for('index'))
def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
    @app.route('/video_feed')
def video_feed():
    return Response(gen(VideoCamera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
if __name__ == "__main__":
    app.secret_key = os.urandom(12)
    app.run(debug=True,host='0.0.0.0', port=5000)
```