from flask import Flask, render\_template, flash, request, session

from flask import render\_template, redirect, url\_for, request

import mysql.connector

import datetime

import time

app = Flask(\_\_name\_\_)

app.config['DEBUG']

app.config['SECRET\_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")

def homepage():

return render\_template('index.html')

@app.route("/Home")

def Home():

return render\_template('index.html')

@app.route("/AdminLogin")

def AdminLogin():

return render\_template('AdminLogin.html')

@app.route("/UserLogin")

def UserLogin():

return render\_template('UserLogin.html')

@app.route("/NewUser")

def NewUser():

returnrender\_template('NewUser.html')

@app.route("/adminlogin", methods=['GET', 'POST'])

def adminlogin():

error = None

if request.method == 'POST':

if request.form['uname'] == 'admin' or request.form['password'] == 'admin':

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb")

data = cur.fetchall()

return render\_template('AdminHome.html', data=data)

else:

return render\_template('index.html', error=error)

@app.route("/AdminHome")

def AdminHome():

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb")

data = cur.fetchall()

return render\_template('AdminHome.html', data=data)

@app.route("/remove")

def remove():

did = request.args.get('did')

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("delete from regtb where Id='"+ did +"' ")

conn.commit()

conn.close()

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb ")

data = cur.fetchall()

return render\_template('AdminHome.html', data=data )

@app.route("/Report")

def Report():

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb")

data = cur.fetchall()

return render\_template('AdminReport.html', data=data)

import cv2

import numpy as np

from skimage.filters import threshold\_local

import tensorflow as tf

from skimage import measure

import imutils

import pytesseract

import re

import mysql.connector

def sort\_cont(character\_contours):

"""

To sort contours from left to right

"""

i = 0

boundingBoxes = [cv2.boundingRect(c) for c in character\_contours]

(character\_contours, boundingBoxes) = zip(\*sorted(zip(character\_contours, boundingBoxes),

key=lambda b: b[1][i], reverse=False))

return character\_contours

def segment\_chars(plate\_img, fixed\_width):

"""

extract Value channel from the HSV format of image and apply adaptive thresholding

to reveal the characters on the license plate

"""

V = cv2.split(cv2.cvtColor(plate\_img, cv2.COLOR\_BGR2HSV))[2]

T = threshold\_local(V, 29, offset=15, method='gaussian')

thresh= (V > T).astype('uint8') \* 255

thresh = cv2.bitwise\_not(thresh)

# resize the license plate region to a canoncial size

plate\_img = imutils.resize(plate\_img, width=fixed\_width)

thresh = imutils.resize(thresh, width=fixed\_width)

bgr\_thresh = cv2.cvtColor(thresh, cv2.COLOR\_GRAY2BGR)

# perform a connected components analysis and initialize the mask to store the locations

# of the character candidates

labels = measure.label(thresh, neighbors=8, background=0)

charCandidates = np.zeros(thresh.shape, dtype='uint8')

# loop over the unique components

characters = []

for label in np.unique(labels):

# if this is the background label, ignore it

if label == 0:

continue

# otherwise, construct the label mask to display only connected components for the

# current label, then find contours in the label mask

labelMask = np.zeros(thresh.shape, dtype='uint8')

labelMask[labels == label] = 255

cnts = cv2.findContours(labelMask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if imutils.is\_cv2() else cnts[1]

# ensure at least one contour was found in the mask

if len(cnts) > 0:

# grab the largest contour which corresponds to the component in the mask, then

# grab the bounding box for the contour

c = max(cnts, key=cv2.contourArea)

(boxX, boxY, boxW, boxH) = cv2.boundingRect(c)

# compute the aspect ratio, solodity, and height ration for the component

aspectRatio = boxW / float(boxH)

solidity = cv2.contourArea(c) / float(boxW \* boxH)

heightRatio = boxH / float(plate\_img.shape[0])

# determine if the aspect ratio, solidity, and height of the contour pass

# the rules tests

keepAspectRatio = aspectRatio < 1.0

keepSolidity = solidity > 0.15

keepHeight = heightRatio > 0.5 and heightRatio < 0.95

# check to see if the component passes all the tests

if keepAspectRatio and keepSolidity and keepHeight and boxW > 14:

# compute the convex hull of the contour and draw it on the character

# candidates mask

hull = cv2.convexHull(c)

cv2.drawContours(charCandidates, [hull], -1, 255, -1)

\_, contours, hier = cv2.findContours(charCandidates, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

contours = sort\_cont(contours)

addPixel = 4 # value to be added to each dimension of the character

for c in contours:

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

if y > addPixel:

y = y - addPixel

else:

y = 0

if x > addPixel:

x = x - addPixel

else:

x = 0

temp = bgr\_thresh[y:y + h + (addPixel \* 2), x:x + w + (addPixel \* 2)]

characters.append(temp)

return characters

else:

return None

class PlateFinder:

def \_\_init\_\_(self):

self.min\_area = 4500 # minimum area of the plate

self.max\_area = 30000 # maximum area of the plate

self.element\_structure = cv2.getStructuringElement(shape=cv2.MORPH\_RECT, ksize=(22, 3))

def preprocess(self, input\_img):

imgBlurred = cv2.GaussianBlur(input\_img, (7, 7), 0) # old window was (5,5)

gray = cv2.cvtColor(imgBlurred, cv2.COLOR\_BGR2GRAY) # convert to gray

sobelx = cv2.Sobel(gray, cv2.CV\_8U, 1, 0, ksize=3) # sobelX to get the vertical edges

ret2, threshold\_img = cv2.threshold(sobelx, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

element = self.element\_structure

morph\_n\_thresholded\_img = threshold\_img.copy()

cv2.morphologyEx(src=threshold\_img, op=cv2.MORPH\_CLOSE, kernel=element, dst=morph\_n\_thresholded\_img)

return morph\_n\_thresholded\_img

def extract\_contours(self, after\_preprocess):

\_, contours, \_ = cv2.findContours(after\_preprocess, mode=cv2.RETR\_EXTERNAL,

method=cv2.CHAIN\_APPROX\_NONE)

return contours

def clean\_plate(self, plate):

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

thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

\_, contours, \_ = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

if contours:

areas = [cv2.contourArea(c) for c in contours]

max\_index = np.argmax(areas) # index of the largest contour in the area array

max\_cnt = contours[max\_index]

max\_cntArea = areas[max\_index]

x, y, w, h = cv2.boundingRect(max\_cnt)

rect = cv2.minAreaRect(max\_cnt)

rotatedPlate = plate

if not self.ratioCheck(max\_cntArea, rotatedPlate.shape[1], rotatedPlate.shape[0]):

return plate, False, None

return rotatedPlate, True, [x, y, w, h]

else:

return plate, False, None

def check\_plate(self, input\_img, contour):

min\_rect = cv2.minAreaRect(contour)

if self.validateRatio(min\_rect):

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

after\_validation\_img = input\_img[y:y + h, x:x + w]

after\_clean\_plate\_img, plateFound, coordinates = self.clean\_plate(after\_validation\_img)

if plateFound:

characters\_on\_plate = self.find\_characters\_on\_plate(after\_clean\_plate\_img)

if (characters\_on\_plate is not None and len(characters\_on\_plate) == 10):

x1, y1, w1, h1 = coordinates

coordinates = x1 + x, y1 + y

after\_check\_plate\_img = after\_clean\_plate\_img

return after\_check\_plate\_img, characters\_on\_plate, coordinates

return None, None, None

def find\_possible\_plates(self, input\_img):

"""

Finding all possible contours that can be plates

"""

plates = []

self.char\_on\_plate = []

self.corresponding\_area = []

self.after\_preprocess = self.preprocess(input\_img)

possible\_plate\_contours = self.extract\_contours(self.after\_preprocess)

for cnts in possible\_plate\_contours:

plate, characters\_on\_plate, coordinates = self.check\_plate(input\_img, cnts)

if plate is not None:

plates.append(plate)

self.char\_on\_plate.append(characters\_on\_plate)

self.corresponding\_area.append(coordinates)

if (len(plates) > 0):

return plates

else:

return None

def find\_characters\_on\_plate(self, plate):

charactersFound = segment\_chars(plate, 400)

if charactersFound:

return charactersFound

# PLATE FEATURES

def ratioCheck(self, area, width, height):

min = self.min\_area

max = self.max\_area

ratioMin = 3

ratioMax = 6

ratio = float(width) / float(height)

if ratio < 1:

ratio = 1 / ratio

if (area < min or area > max) or (ratio < ratioMin or ratio > ratioMax):

return False

return True

def preRatioCheck(self, area, width, height):

min = self.min\_area

max = self.max\_area

ratioMin = 2.5

ratioMax = 7

ratio = float(width) / float(height)

if ratio < 1:

ratio = 1 / ratio

if (area < min or area > max) or (ratio < ratioMin or ratio > ratioMax):

return False

return True

def validateRatio(self, rect):

(x, y), (width, height), rect\_angle = rect

if (width > height):

angle = -rect\_angle

else:

angle = 90 + rect\_angle

if angle > 15:

return False

if (height == 0 or width == 0):

return False

area = width \* height

if not self.preRatioCheck(area, width, height):

return False

else:

return True

class NeuralNetwork:

def \_\_init\_\_(self):

self.model\_file = "./model/binary\_128\_0.50\_ver3.pb"

self.label\_file = "./model/binary\_128\_0.50\_labels\_ver2.txt"

self.label = self.load\_label(self.label\_file)

self.graph = self.load\_graph(self.model\_file)

self.sess = tf.Session(graph=self.graph)

def load\_graph(self, modelFile):

graph = tf.Graph()

graph\_def = tf.GraphDef()

with open(modelFile, "rb") as f:

graph\_def.ParseFromString(f.read())

with graph.as\_default():

tf.import\_graph\_def(graph\_def)

return graph

def load\_label(self, labelFile):

label = []

proto\_as\_ascii\_lines = tf.gfile.GFile(labelFile).readlines()

for l in proto\_as\_ascii\_lines:

label.append(l.rstrip())

return label

def convert\_tensor(self, image, imageSizeOuput):

"""

takes an image and tranform it in tensor

"""

image = cv2.resize(image, dsize=(imageSizeOuput, imageSizeOuput), interpolation=cv2.INTER\_CUBIC)

np\_image\_data = np.asarray(image)

np\_image\_data = cv2.normalize(np\_image\_data.astype('float'), None, -0.5, .5, cv2.NORM\_MINMAX)

np\_final = np.expand\_dims(np\_image\_data, axis=0)

return np\_final

def label\_image(self, tensor):

input\_name = "import/input"

output\_name = "import/final\_result"

input\_operation = self.graph.get\_operation\_by\_name(input\_name)

output\_operation = self.graph.get\_operation\_by\_name(output\_name)

results = self.sess.run(output\_operation.outputs[0],

{input\_operation.outputs[0]: tensor})

results = np.squeeze(results)

labels = self.label

top = results.argsort()[-1:][::-1]

return labels[top[0]]

def label\_image\_list(self, listImages, imageSizeOuput):

plate = ""

for img in listImages:

if cv2.waitKey(25) & 0xFF == ord('q'):

break

plate = plate + self.label\_image(self.convert\_tensor(img, imageSizeOuput))

return plate, len(plate)

@app.route("/Verify")

def Verify():

import numpy as np

import time

import cv2

import os

import numpy as np

args = {"confidence": 0.8, "threshold": 0.5}

flag = False

labelsPath = "./helmet.names"

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

final\_classes = ['Helmet']

np.random.seed(42)

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

dtype="uint8")

weightsPath = os.path.abspath("./yolov3-helmet.weights")

configPath = os.path.abspath("./yolov3-helmet.cfg")

# print(configPath, "\n", weightsPath)

flagg1 = 0

flagg2 = 0

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

ln = net.getLayerNames()

ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

vs = cv2.VideoCapture(0)

writer = None

(W, H) = (None, None)

flag = True

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

while True:

# read the next frame from the file

(grabbed, frame) = vs.read()

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

# of the stream

if not grabbed:

break

# if the frame dimensions are empty, grab them

if W is None or H is None:

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

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

swapRB=True, crop=False)

net.setInput(blob)

start = time.time()

layerOutputs = net.forward(ln)

end = time.time()

# initialize our lists of detected bounding boxes, confidences,

# and class IDs, respectively

boxes = []

confidences = []

classIDs = []

# loop over each of the layer outputs

for output in layerOutputs:

# loop over each of the detections

for detection in output:

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

# of the current object detection

scores = detection[5:]

classID = np.argmax(scores)

confidence = scores[classID]

# filter out weak predictions by ensuring the detected

# probability is greater than the minimum probability

if confidence > args["confidence"]:

# scale the bounding box coordinates back relative to

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

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

# the bounding box followed by the boxes' width and

# height

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

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

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

# and and left corner of the bounding box

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

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

# update our list of bounding box coordinates,

# confidences, and class IDs

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

confidences.append(float(confidence))

classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping

# bounding boxes

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

args["threshold"])

# ensure at least one detection exists

if len(idxs) > 0:

# loop over the indexes we are keeping

for i in idxs.flatten():

# extract the bounding box coordinates

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

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

if (LABELS[classIDs[i]] in final\_classes):

# playsound('alarm.wav')

if (flag):

# alert()

flag = False

# async\_email(LABELS[classIDs[i]])

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

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

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

confidences[i])

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

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

cv2.imshow("Color", frame)

flagg1 += 1

# print(flag)

if (flagg1 == 30):

print("Helmet Found!")

vs.release()

cv2.destroyAllWindows()

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb ")

data = cur.fetchall()

return render\_template('AdminReport.html', data=data)

else:

flag = True

print(0)

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

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

for x, y, w, h in faces:

frame = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

cv2.imshow('Color', frame)

flagg2 += 1

# print(flag)

if (flagg2 == 30):

flagg2 = 0

print("Helmet Not Found!")

vs.release()

cv2.destroyAllWindows()

return vv()

if cv2.waitKey(1) == ord('q'):

break

# release the webcam and destroy all active windows

vs.release()

cv2.destroyAllWindows()

def vv():

session['vno'] = ''

findPlate = PlateFinder()

# Initialize the Neural Network

model = NeuralNetwork()

cap = cv2.VideoCapture(0)

while (True):

ret, img = cap.read()

if ret == True:

cv2.imshow('original video', img)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

# cv2.waitKey(0)

possible\_plates = findPlate.find\_possible\_plates(img)

pytesseract.pytesseract.tesseract\_cmd = 'C:\\Program Files (x86)\\Tesseract-OCR\\tesseract.exe'

if possible\_plates is not None:

for i, p in enumerate(possible\_plates):

chars\_on\_plate = findPlate.char\_on\_plate[i]

recognized\_plate, \_ = model.label\_image\_list(chars\_on\_plate, imageSizeOuput=128)

print(recognized\_plate)

cv2.imshow('plate', p)

predicted\_result = pytesseract.image\_to\_string(p, lang='eng',

config='--oem 3 --psm 6 -c tessedit\_char\_whitelist = ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')

print(predicted\_result)

vno = re.sub(r"[^a-zA-Z0-9]", "", predicted\_result)

print(vno)

ts = time.time()

date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')

timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("select \* from regtb where VehicleNo='" + str(vno) + "' ")

data = cursor.fetchone()

if data is None:

print("VehilceNo Not Found")

else:

unm = data[8]

mob = data[6]

conn = mysql.connector.connect(user='root', password='', host='localhost',

database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute(

"select \* from entrytb where Date='" + str(date) + "' and VehicleNo='" + str(vno) + "'")

data = cursor.fetchone()

if data is None:

conn = mysql.connector.connect(user='root', password='', host='localhost',

database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute(

"insert into entrytb values('','" + str(vno) + "','"+ unm +"','" + str(date) + "','" + str(

timeStamp) + "','500','NotPaid')")

conn.commit()

conn.close()

print("Fine Amount Info Saved")

sendmsg(mob," Fine Amount For Helmet Not Wearing RS.500")

vnoo = vno

cap.release()

cv2.destroyAllWindows()

conn = mysql.connector.connect(user='root', password='', host='localhost',

database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb ")

data = cur.fetchall()

return render\_template('AdminReport.html',data=data)

else:

cap.release()

cv2.destroyAllWindows()

return "Already Fine Amount Info Saved"

if cv2.waitKey(25) & 0xFF == ord('q'):

break

else:

break

cap.release()

cv2.destroyAllWindows()

return render\_template('AdminHome.html')

# if(session['vno'] !=''):

@app.route("/newsuer", methods=['GET', 'POST'])

def newsuer():

if request.method == 'POST':

vno = request.form['vno']

name = request.form['name']

gender = request.form['gender']

Age = request.form['Age']

email = request.form['email']

pnumber = request.form['pnumber']

address = request.form['address']

uname = request.form['uname']

password = request.form['password']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("insert into regtb values('','"+vno+"','"+name+"','"+gender+"','"+Age+"','"+email+"','"+pnumber+"','"+address +"','"+uname+"','"+password+"')")

conn.commit()

conn.close()

return render\_template("UserLogin.html")

@app.route("/AUserSearch", methods=['GET', 'POST'])

def AUserSearch():

if request.method == 'POST':

date = request.form['date']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb where date='"+date+"'")

data = cur.fetchall()

return render\_template('AdminReport.html', data=data)

global vno

global Email

def examvales1():

vvno= session['vno']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM regtb where VehicleNo ='" + vvno + "'")

data = cursor.fetchone()

if data:

vno = data[0]

Email = data[4]

else:

return 'Incorrect username / password !'

return vno,Email

def sendmsg(targetno,message):

import requests

requests.post("http://smsserver9.creativepoint.in/api.php?username=fantasy&password=596692&to=" + targetno + "&from=FSSMSS&message=Dear user your msg is " + message + " Sent By FSMSG FSSMSS&PEID=1501563800000030506&templateid=1507162882948811640")

@app.route("/userlogin", methods=['GET', 'POST'])

def userlogin():

error = None

if request.method == 'POST':

username = request.form['uname']

password = request.form['password']

session['uname'] = request.form['uname']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("SELECT \* from regtb where username='" + username + "' and Password='" + password + "'")

data = cursor.fetchone()

if data is None:

alert = 'Username or Password is wrong'

return render\_template('goback.html', data=alert)

else:

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb where username='" + username + "' and Password='" + password + "'")

data = cur.fetchall()

return render\_template('UserHome.html', data=data )

@app.route("/UserHome")

def UserHome():

uname = session['uname']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb where username='" + uname + "' ")

data = cur.fetchall()

return render\_template('UserHome.html', data=data)

@app.route("/UserReport")

def UserReport():

uname = session['uname']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb where username='" + uname + "' and Status !='Paid' ")

data = cur.fetchall()

return render\_template('UserReport.html', data=data)

@app.route("/UserSearch", methods=['GET', 'POST'])

def UserSearch():

if request.method == 'POST':

date = request.form['date']

uname = session['uname']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cur = conn.cursor()

cur.execute("SELECT \* FROM entrytb where UserName='"+uname+"' and date='"+date+"'")

data = cur.fetchall()

return render\_template('UserReport.html', data=data)

@app.route("/pay")

def pay():

did = request.args.get('did')

session['bid'] = did

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("SELECT FineAmount FROM entrytb where id ='" + did + "'")

data = cursor.fetchone()

if data:

Amt = data[0]

return render\_template('Payment.html', Amount=Amt)

@app.route("/Payment", methods=['GET', 'POST'])

def Payment():

did = session['bid']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("Update entrytb set Status='Paid' where id='" + did + "'")

conn.commit()

conn.close()

alert = "Payment Sucessful! "

return render\_template('goback.html', data=alert)

@app.route("/numupdate", methods=['GET', 'POST'])

def numupdate():

if request.method == 'POST':

num = request.form['num']

uname = session['uname']

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

cursor = conn.cursor()

cursor.execute("Update regtb set VehicleNo='"+ num +"' where Username='"+ uname +"'")

conn.commit()

conn.close()

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberhelmetdb')

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT \* FROM regtb where UserName='"+uname+"' ")

data = cur.fetchall()

return render\_template('UserHome.html', data=data)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, use\_reloader=True)

import cv2

import numpy as np

from skimage.filters import threshold\_local

import tensorflow as tf

from skimage import measure

import imutils

import pytesseract

import re

import mysql.connector

import datetime

import time

def sort\_cont(character\_contours):

"""

To sort contours from left to right

"""

i = 0

boundingBoxes = [cv2.boundingRect(c) for c in character\_contours]

(character\_contours, boundingBoxes) = zip(\*sorted(zip(character\_contours, boundingBoxes),

key=lambda b: b[1][i], reverse=False))

return character\_contours

def segment\_chars(plate\_img, fixed\_width):

"""

extract Value channel from the HSV format of image and apply adaptive thresholding

to reveal the characters on the license plate

"""

V = cv2.split(cv2.cvtColor(plate\_img, cv2.COLOR\_BGR2HSV))[2]

T = threshold\_local(V, 29, offset=15, method='gaussian')

thresh = (V > T).astype('uint8') \* 255

thresh = cv2.bitwise\_not(thresh)

# resize the license plate region to a canoncial size

plate\_img = imutils.resize(plate\_img, width=fixed\_width)

thresh = imutils.resize(thresh, width=fixed\_width)

bgr\_thresh = cv2.cvtColor(thresh, cv2.COLOR\_GRAY2BGR)

# perform a connected components analysis and initialize the mask to store the locations

# of the character candidates

labels = measure.label(thresh, neighbors=8, background=0)

charCandidates = np.zeros(thresh.shape, dtype='uint8')

# loop over the unique components

characters = []

for label in np.unique(labels):

# if this is the background label, ignore it

if label == 0:

continue

# otherwise, construct the label mask to display only connected components for the

# current label, then find contours in the label mask

labelMask = np.zeros(thresh.shape, dtype='uint8')

labelMask[labels == label] = 255

cnts = cv2.findContours(labelMask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if imutils.is\_cv2() else cnts[1]

# ensure at least one contour was found in the mask

if len(cnts) > 0:

# grab the largest contour which corresponds to the component in the mask, then

# grab the bounding box for the contour

c = max(cnts, key=cv2.contourArea)

(boxX, boxY, boxW, boxH) = cv2.boundingRect(c)

# compute the aspect ratio, solodity, and height ration for the component

aspectRatio = boxW / float(boxH)

solidity = cv2.contourArea(c) / float(boxW \* boxH)

heightRatio = boxH / float(plate\_img.shape[0])

# determine if the aspect ratio, solidity, and height of the contour pass

# the rules tests

keepAspectRatio = aspectRatio < 1.0

keepSolidity = solidity > 0.15

keepHeight = heightRatio > 0.5 and heightRatio < 0.95

# check to see if the component passes all the tests

if keepAspectRatio and keepSolidity and keepHeight and boxW > 14:

# compute the convex hull of the contour and draw it on the character

# candidates mask

hull = cv2.convexHull(c)

cv2.drawContours(charCandidates, [hull], -1, 255, -1)

\_, contours, hier = cv2.findContours(charCandidates, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

contours = sort\_cont(contours)

addPixel = 4 # value to be added to each dimension of the character

for c in contours:

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

if y > addPixel:

y = y - addPixel

else:

y = 0

if x > addPixel:

x = x - addPixel

else:

x = 0

temp = bgr\_thresh[y:y + h + (addPixel \* 2), x:x + w + (addPixel \* 2)]

characters.append(temp)

return characters

else:

return None

class PlateFinder:

def \_\_init\_\_(self):

self.min\_area = 4500 # minimum area of the plate

self.max\_area = 30000 # maximum area of the plate

self.element\_structure = cv2.getStructuringElement(shape=cv2.MORPH\_RECT, ksize=(22, 3))

def preprocess(self, input\_img):

imgBlurred = cv2.GaussianBlur(input\_img, (7, 7), 0) # old window was (5,5)

gray = cv2.cvtColor(imgBlurred, cv2.COLOR\_BGR2GRAY) # convert to gray

sobelx = cv2.Sobel(gray, cv2.CV\_8U, 1, 0, ksize=3) # sobelX to get the vertical edges

ret2, threshold\_img = cv2.threshold(sobelx, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

element = self.element\_structure

morph\_n\_thresholded\_img = threshold\_img.copy()

cv2.morphologyEx(src=threshold\_img, op=cv2.MORPH\_CLOSE, kernel=element, dst=morph\_n\_thresholded\_img)

return morph\_n\_thresholded\_img

def extract\_contours(self, after\_preprocess):

\_, contours, \_ = cv2.findContours(after\_preprocess, mode=cv2.RETR\_EXTERNAL,

method=cv2.CHAIN\_APPROX\_NONE)

return contours

def clean\_plate(self, plate):

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

thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

\_, contours, \_ = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

if contours:

areas = [cv2.contourArea(c) for c in contours]

max\_index = np.argmax(areas) # index of the largest contour in the area array

max\_cnt = contours[max\_index]

max\_cntArea = areas[max\_index]

x, y, w, h = cv2.boundingRect(max\_cnt)

rect = cv2.minAreaRect(max\_cnt)

rotatedPlate = plate

if not self.ratioCheck(max\_cntArea, rotatedPlate.shape[1], rotatedPlate.shape[0]):

return plate, False, None

return rotatedPlate, True, [x, y, w, h]

else:

return plate, False, None

def check\_plate(self, input\_img, contour):

min\_rect = cv2.minAreaRect(contour)

if self.validateRatio(min\_rect):

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

after\_validation\_img = input\_img[y:y + h, x:x + w]

after\_clean\_plate\_img, plateFound, coordinates = self.clean\_plate(after\_validation\_img)

if plateFound:

characters\_on\_plate = self.find\_characters\_on\_plate(after\_clean\_plate\_img)

if (characters\_on\_plate is not None and len(characters\_on\_plate) == 10):

x1, y1, w1, h1 = coordinates

coordinates = x1 + x, y1 + y

after\_check\_plate\_img = after\_clean\_plate\_img

return after\_check\_plate\_img, characters\_on\_plate, coordinates

return None, None, None

def find\_possible\_plates(self, input\_img):

"""

Finding all possible contours that can be plates

"""

plates = []

self.char\_on\_plate = []

self.corresponding\_area = []

self.after\_preprocess = self.preprocess(input\_img)

possible\_plate\_contours = self.extract\_contours(self.after\_preprocess)

for cnts in possible\_plate\_contours:

plate, characters\_on\_plate, coordinates = self.check\_plate(input\_img, cnts)

if plate is not None:

plates.append(plate)

self.char\_on\_plate.append(characters\_on\_plate)

self.corresponding\_area.append(coordinates)

if (len(plates) > 0):

return plates

else:

return None

def find\_characters\_on\_plate(self, plate):

charactersFound = segment\_chars(plate, 400)

if charactersFound:

return charactersFound

# PLATE FEATURES

def ratioCheck(self, area, width, height):

min = self.min\_area

max = self.max\_area

ratioMin = 3

ratioMax = 6

ratio = float(width) / float(height)

if ratio < 1:

ratio = 1 / ratio

if (area < min or area > max) or (ratio < ratioMin or ratio > ratioMax):

return False

return True

def preRatioCheck(self, area, width, height):

min = self.min\_area

max = self.max\_area

ratioMin = 2.5

ratioMax = 7

ratio = float(width) / float(height)

if ratio < 1:

ratio = 1 / ratio

if (area < min or area > max) or (ratio < ratioMin or ratio > ratioMax):

return False

return True

def validateRatio(self, rect):

(x, y), (width, height), rect\_angle = rect

if (width > height):

angle = -rect\_angle

else:

angle = 90 + rect\_angle

if angle > 15:

return False

if (height == 0 or width == 0):

return False

area = width \* height

if not self.preRatioCheck(area, width, height):

return False

else:

return True

class NeuralNetwork:

def \_\_init\_\_(self):

self.model\_file = "./model/binary\_128\_0.50\_ver3.pb"

self.label\_file = "./model/binary\_128\_0.50\_labels\_ver2.txt"

self.label = self.load\_label(self.label\_file)

self.graph = self.load\_graph(self.model\_file)

self.sess = tf.Session(graph=self.graph)

def load\_graph(self, modelFile):

graph = tf.Graph()

graph\_def = tf.GraphDef()

with open(modelFile, "rb") as f:

graph\_def.ParseFromString(f.read())

with graph.as\_default():

tf.import\_graph\_def(graph\_def)

return graph

def load\_label(self, labelFile):

label = []

proto\_as\_ascii\_lines = tf.gfile.GFile(labelFile).readlines()

for l in proto\_as\_ascii\_lines:

label.append(l.rstrip())

return label

def convert\_tensor(self, image, imageSizeOuput):

"""

takes an image and tranform it in tensor

"""

image = cv2.resize(image, dsize=(imageSizeOuput, imageSizeOuput), interpolation=cv2.INTER\_CUBIC)

np\_image\_data = np.asarray(image)

np\_image\_data = cv2.normalize(np\_image\_data.astype('float'), None, -0.5, .5, cv2.NORM\_MINMAX)

np\_final = np.expand\_dims(np\_image\_data, axis=0)

return np\_final

def label\_image(self, tensor):

input\_name = "import/input"

output\_name = "import/final\_result"

input\_operation = self.graph.get\_operation\_by\_name(input\_name)

output\_operation = self.graph.get\_operation\_by\_name(output\_name)

results = self.sess.run(output\_operation.outputs[0],

{input\_operation.outputs[0]: tensor})

results = np.squeeze(results)

labels = self.label

top = results.argsort()[-1:][::-1]

return labels[top[0]]

def label\_image\_list(self, listImages, imageSizeOuput):

plate = ""

for img in listImages:

if cv2.waitKey(25) & 0xFF == ord('q'):

break

plate = plate + self.label\_image(self.convert\_tensor(img, imageSizeOuput))

return plate, len(plate)

findPlate = PlateFinder()

model = NeuralNetwork()

cap = cv2.VideoCapture(0)

vnoo=''

from pynput.keyboard import Key, Controller

while (cap.isOpened()):

ret, img = cap.read()

if (vnoo != ""):

keyboard = Controller()

key = "q"

keyboard.press(key)

keyboard.release(key)

cap.release()

cv2.destroyAllWindows()

if ret == True:

cv2.imshow('original video', img)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

# cv2.waitKey(0)

possible\_plates = findPlate.find\_possible\_plates(img)

pytesseract.pytesseract.tesseract\_cmd = 'C:\\Program Files (x86)\\Tesseract-OCR\\tesseract.exe'

if possible\_plates is not None:

for i, p in enumerate(possible\_plates):

chars\_on\_plate = findPlate.char\_on\_plate[i]

recognized\_plate, \_ = model.label\_image\_list(chars\_on\_plate, imageSizeOuput=128)

print(recognized\_plate)

cv2.imshow('plate', p)

predicted\_result = pytesseract.image\_to\_string(p, lang='eng',

config='--oem 3 --psm 6 -c tessedit\_char\_whitelist = ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')

print(predicted\_result)

vno = re.sub(r"[^a-zA-Z0-9]", "", predicted\_result)

print(vno)

ts = time.time()

date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')

timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')

conn = mysql.connector.connect(user='root', password='', host='localhost', database='1numberfacedb')

cursor = conn.cursor()

cursor.execute("select \* from regtb where VehicleNo='" + str(vno) + "' ")

data = cursor.fetchone()

if data is None:

print("VehilceNo Not Found")

else:

conn = mysql.connector.connect(user='root', password='', host='localhost',

database='1numberfacedb')

cursor = conn.cursor()

cursor.execute(

"select \* from entrytb where Date='" + str(date) + "' and VehicleNo='" + str(vno) + "'")

data = cursor.fetchone()

if data is None:

conn = mysql.connector.connect(user='root', password='', host='localhost',

database='1numberfacedb')

cursor = conn.cursor()

cursor.execute(

"insert into entrytb values('','" + str(vno) + "','','" + str(date) + "','" + str(

timeStamp) + "')")

conn.commit()

conn.close()

print("Face Attendance Info Saved")

vnoo=vno

cap.release()

cv2.destroyAllWindows()

else:

cap.release()

cv2.destroyAllWindows()

print("Already Face Attendance Info Saved")

break

if cv2.waitKey(25) & 0xFF == ord('q'):

break

else:

Break

cap.release()

cv2.destroyAllWindows()

import numpy as np

import time

import cv2

import os

import numpy as np

args = {"confidence": 0.8, "threshold": 0.5}

flag = False

labelsPath = "./helmet.names"

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

final\_classes = ['Helmet']

np.random.seed(42)

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

dtype="uint8")

weightsPath = os.path.abspath("./yolov3-helmet.weights")

configPath = os.path.abspath("./yolov3-helmet.cfg")

# print(configPath, "\n", weightsPath)

flagg1 = 0

flagg2 = 0

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

ln = net.getLayerNames()

ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

vs = cv2.VideoCapture(0)

writer = None

(W, H) = (None, None)

flag = True

face\_cascade=cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

while True:

# read the next frame from the file

(grabbed, frame) = vs.read()

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

# of the stream

if not grabbed:

break

# if the frame dimensions are empty, grab them

if W is None or H is None:

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

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

swapRB=True, crop=False)

net.setInput(blob)

start = time.time()

layerOutputs = net.forward(ln)

end = time.time()

# initialize our lists of detected bounding boxes, confidences,

# and class IDs, respectively

boxes = []

confidences = []

classIDs = []

# loop over each of the layer outputs

for output in layerOutputs:

# loop over each of the detections

for detection in output:

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

# of the current object detection

scores = detection[5:]

classID = np.argmax(scores)

confidence = scores[classID]

# filter out weak predictions by ensuring the detected

# probability is greater than the minimum probability

if confidence > args["confidence"]:

# scale the bounding box coordinates back relative to

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

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

# the bounding box followed by the boxes' width and

# height

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

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

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

# and and left corner of the bounding box

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

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

# update our list of bounding box coordinates,

# confidences, and class IDs

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

confidences.append(float(confidence))

classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping

# bounding boxes

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

args["threshold"])

# ensure at least one detection exists

if len(idxs) > 0:

# loop over the indexes we are keeping

for i in idxs.flatten():

# extract the bounding box coordinates

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

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

if (LABELS[classIDs[i]] in final\_classes):

# playsound('alarm.wav')

if (flag):

# alert()

flag = False

# async\_email(LABELS[classIDs[i]])

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

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

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

confidences[i])

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

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

cv2.imshow("Color", frame)

flagg1 += 1

# print(flag)

if (flagg1 == 10):

print("Helmet Found!")

else:

flag = True

print(0)

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

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

for x, y, w, h in faces:

frame = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

cv2.imshow('Color', frame)

flagg2 += 1

# print(flag)

if (flagg2 == 10):

flagg2 = 0

print("Helmet Not Found!")

if cv2.waitKey(1) == ord('q'):

break

# release the webcam and destroy all active windows

vs.release()

cv2.destroyAllWindows()







