**BIONIC EYE MODEL TO PROVIDE VISION OR RESTORE SIGHT FOR BLINDNESS USING VISION TRANSFORMER**

**SOURCE CODE**

**Packages**

from flask import Flask

from flask import Flask, render\_template, Response, redirect, request, session, abort, url\_for

from camera import VideoCamera

from camera2 import VideoCamera2

from datetime import datetime

from datetime import date

import f\_Face\_info

import imutils

import face\_recognition

from age\_detection import f\_my\_age

from gender\_detection import f\_my\_gender

from race\_detection import f\_my\_race

from emotion\_detection import f\_emotion\_de

from deepface.basemodels import VGGFace

from pathlib import Path

import gdown

import numpy as np

import tensorflow as tf

from keras.models import Model, Sequential

from keras.layers import Convolution2D, Flatten, Activation

#from keras.preprocessing import image

from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input

from tensorflow.keras.utils import img\_to\_array

import datetime

import random

import cv2

import numpy as np

import os

import time

import shutil

import imagehash

import PIL.Image

from PIL import Image

import urllib.request

import argparse

import pyttsx3

import mysql.connector

**Training**

##Preprocess

path="static/frame/"+rs[2]

path2="static/process1/"+rs[2]

mm2 = PIL.Image.open(path).convert('L')

rz = mm2.resize((200,200), PIL.Image.ANTIALIAS)

rz.save(path2)

#noice

img = cv2.imread('static/process1/'+rs[2])

dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)

fname2='ns\_'+rs[2]

cv2.imwrite("static/process1/"+fname2, dst)

##bin

image = cv2.imread('static/process1/'+rs[2])

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)

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/process1/bin\_"+rs[2], thresh)

###RPN - Segment

img = cv2.imread('static/process1/'+rs[2])

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,0.7\*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/process2/fg\_"+rs[2]

segment.save(path3)

img = cv2.imread('static/process2/fg\_'+rs[2])

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,0.7\*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/process2/fg\_"+rs[2]

segment.save(path3)

def CNN(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')

# 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')

**Model Generation**

class Age\_Model():

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

self.model = self.loadModel()

self.output\_indexes = np.array([i for i in range(0, 101)])

def predict\_age(self,face\_image):

image\_preprocesing = self.transform\_face\_array2age\_face(face\_image)

age\_predictions = self.model.predict(image\_preprocesing )[0,:]

result\_age = self.findApparentAge(age\_predictions)

return result\_age

def loadModel(self):

model = VGGFace.baseModel()

classes = 101

base\_model\_output = Sequential()

base\_model\_output = Convolution2D(classes, (1, 1), name='predictions')(model.layers[-4].output)

base\_model\_output = Flatten()(base\_model\_output)

base\_model\_output = Activation('softmax')(base\_model\_output

ge\_model = Model(inputs=model.input, outputs=base\_model\_output)

#load weights

home = str(Path.home())

if os.path.isfile(home+'/.deepface/weights/age\_model\_weights.h5') != True:

print("age\_model\_weights.h5 will be downloaded...")

url = 'https://drive.google.com/uc?id=1YCox\_4kJ-BYeXq27uUbasu--yz28zUMV'

output = home+'/.deepface/weights/age\_model\_weights.h5'

gdown.download(url, output, quiet=False)

age\_model.load\_weights(home+'/.deepface/weights/age\_model\_weights.h5')

return age\_model

def findApparentAge(self,age\_predictions):

apparent\_age = np.sum(age\_predictions \* self.output\_indexes)

return apparent\_age

def transform\_face\_array2age\_face(self,face\_array,grayscale=False,target\_size = (224, 224)):

detected\_face = face\_array

if grayscale == True:

detected\_face = cv2.cvtColor(detected\_face, cv2.COLOR\_BGR2GRAY)

detected\_face = cv2.resize(detected\_face, target\_size)

#img\_pixels = image.img\_to\_array(detected\_face)

img\_pixels = tf.keras.utils.img\_to\_array(detected\_face)

img\_pixels = np.expand\_dims(img\_pixels, axis = 0)

#normalize input in [0, 1]

img\_pixels /= 255

return img\_pixels

class predict\_emotions():

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

# cargo modelo de deteccion de emociones

self.model = load\_model(cfg.path\_model)

def preprocess\_img(self,face\_image,rgb=True,w=48,h=48):

face\_image = cv2.resize(face\_image, (w,h))

if rgb == False:

face\_image = cv2.cvtColor(face\_image, cv2.COLOR\_BGR2GRAY)

face\_image = face\_image.astype("float") / 255.0

#face\_image= img\_to\_array(face\_image)

face\_image= tf.keras.utils.img\_to\_array(face\_image)

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

return face\_image

def get\_emotion(self,img,boxes\_face):

emotions = []

if len(boxes\_face)!=0:

for box in boxes\_face:

y0,x0,y1,x1 = box

face\_image = img[x0:x1,y0:y1]

# preprocesar data

face\_image = self.preprocess\_img(face\_image ,cfg.rgb, cfg.w, cfg.h)

# predecir imagen

prediction = self.model.predict(face\_image)

emotion = cfg.labels[prediction.argmax()]

emotions.append(emotion)

else:

emotions = []

boxes\_face = []

return boxes\_face,emotions

**Capture Video**

parser = argparse.ArgumentParser(description="Face Info")

parser.add\_argument('--input', type=str, default= 'webcam',

help="webcam or image")

parser.add\_argument('--path\_im', type=str,

help="path of image")

args = vars(parser.parse\_args())

type\_input = args['input']

if type\_input == 'image':

# ----------------------------- image -----------------------------

#ingestar data

frame = cv2.imread(args['path\_im'])

# obtenego info del frame

out = f\_Face\_info.get\_face\_info(frame)

# pintar imagen

res\_img = f\_Face\_info.bounding\_box(out,frame)

cv2.imshow('Face info',res\_img)

cv2.waitKey(0)

if type\_input == 'webcam':

# ----------------------------- webcam -----------------------------

cv2.namedWindow("Face info")

cam = cv2.VideoCapture(0)

while True:

star\_time = time.time()

ret, frame = cam.read()

frame = imutils.resize(frame, width=720)

# obtenego info del frame

out = f\_Face\_info.get\_face\_info(frame)

# pintar imagen

res\_img = f\_Face\_info.bounding\_box(out,frame)

end\_time = time.time() - star\_time

FPS = 1/end\_time

cv2.putText(res\_img,f"FPS: {round(FPS,3)}",(10,50),cv2.FONT\_HERSHEY\_COMPLEX,1,(0,0,255),2)

cv2.imshow('Face info',res\_img)

if cv2.waitKey(1) &0xFF == ord('q'):

break

**Testing**

**Person/Object Identification**

age\_detector = f\_my\_age.Age\_Model()

gender\_detector = f\_my\_gender.Gender\_Model()

race\_detector = f\_my\_race.Race\_Model()

emotion\_detector = f\_emotion\_detection.predict\_emotions()

rec\_face = f\_main.rec()

def get\_face\_info(im):

# face detection

boxes\_face = face\_recognition.face\_locations(im)

out = []

if len(boxes\_face)!=0:

for box\_face in boxes\_face:

# segmento rostro

box\_face\_fc = box\_face

x0,y1,x1,y0 = box\_face

box\_face = np.array([y0,x0,y1,x1])

face\_features = {

"name":[],

"age":[],

"gender":[],

"race":[],

"emotion":[],

"bbx\_frontal\_face":box\_face

face\_image = im[x0:x1,y0:y1]

# -------------------------------------- face\_recognition ---------------------------------------

face\_features["name"] = rec\_face.recognize\_face2(im,[box\_face\_fc])[0]

# -------------------------------------- age\_detection ---------------------------------------

age = age\_detector.predict\_age(face\_image)

face\_features["age"] = str(round(age,2))

# -------------------------------------- gender\_detection ---------------------------------------

face\_features["gender"] = gender\_detector.predict\_gender(face\_image)

# -------------------------------------- race\_detection ---------------------------------------

face\_features["race"] = race\_detector.predict\_race(face\_image)

# -------------------------------------- emotion\_detection ---------------------------------------

\_,emotion = emotion\_detector.get\_emotion(im,[box\_face])

face\_features["emotion"] = emotion[0]

# -------------------------------------- out ---------------------------------------

out.append(face\_features)

else:

face\_features = {

"name":[],

"age":[],

"gender":[],

"race":[],

"emotion":[],

"bbx\_frontal\_face":[]

}

out.append(face\_features)

return out

def bounding\_box(out,img):

mm=""

for data\_face in out:

box = data\_face["bbx\_frontal\_face"]

if len(box) == 0:

continue

else:

x0,y0,x1,y1 = box

img = cv2.rectangle(img,

(x0,y0),

(x1,y1),

(0,255,0),2);

thickness = 1

fontSize = 0.5

step = 13

na=data\_face["name"]

na1=na.split('\_')

if data\_face["name"]=="unknown":

mm+="Unknown person found"

else:

mm+=" Name is "+na1[0]

mm+=", Gender is"+data\_face["gender"]

mm+=", Age is "+data\_face["age"]

mm+=", "+data\_face["race"]

mm+=", Emotion is "+data\_face["emotion"]+" "

ff1=open("mess1.txt","w")

ff1.write(mm)

ff1.close()

rn=randint(230,390)

dst=str(rn)+" cm"

agg=data\_face["age"]+", "+dst

try:

cv2.putText(img, "age: " +agg, (x0, y0-7), cv2.FONT\_HERSHEY\_SIMPLEX, fontSize, (0,255,0), thickness)

except:

pass

try:

cv2.putText(img, "gender: " +data\_face["gender"], (x0, y0-step-10\*1), cv2.FONT\_HERSHEY\_SIMPLEX, fontSize, (0,255,0), thickness)

except:

pass

try:

cv2.putText(img, "race: " +data\_face["race"], (x0, y0-step-10\*2), cv2.FONT\_HERSHEY\_SIMPLEX, fontSize, (0,255,0), thickness)

except:

pass

try:

cv2.putText(img, "emotion: " +data\_face["emotion"], (x0, y0-step-10\*3), cv2.FONT\_HERSHEY\_SIMPLEX, fontSize, (0,255,0), thickness)

except:

pass

try:

cv2.putText(img, "name: " +na1[0], (x0, y0-step-10\*4), cv2.FONT\_HERSHEY\_SIMPLEX, fontSize, (0,255,0), thickness)

except:

pass

return img