#!/usr/bin/env python

'''

face detection using haar cascades

USAGE:

facedetect.py [--cascade <cascade\_fn>] [--nested-cascade <cascade\_fn>] [<video\_source>]

'''

# Python 2/3 compatibility

from \_\_future\_\_ import print\_function

import numpy as np

import cv2

# local modules

from video import create\_capture

from common import clock, draw\_str

def detect(img, cascade):

rects = cascade.detectMultiScale(img, scaleFactor=1.3, minNeighbors=4, minSize=(30, 30),

flags=cv2.CASCADE\_SCALE\_IMAGE)

if len(rects) == 0:

return []

rects[:,2:] += rects[:,:2]

return rects

def draw\_rects(img, rects, color):

for x1, y1, x2, y2 in rects:

cv2.rectangle(img, (x1, y1), (x2, y2), color, 2)

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

import sys, getopt

print(\_\_doc\_\_)

args, video\_src = getopt.getopt(sys.argv[1:], '', ['cascade=', 'nested-cascade='])

try:

video\_src = video\_src[0]

except:

video\_src = 0

args = dict(args)

cascade\_fn = args.get('--cascade', "../../data/haarcascades/haarcascade\_frontalface\_alt.xml")

nested\_fn = args.get('--nested-cascade', "../../data/haarcascades/haarcascade\_eye.xml")

cascade = cv2.CascadeClassifier(cascade\_fn)

nested = cv2.CascadeClassifier(nested\_fn)

cam = create\_capture(video\_src, fallback='synth:bg=../data/lena.jpg:noise=0.05')

while True:

ret, img = cam.read()

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

gray = cv2.equalizeHist(gray)

t = clock()

rects = detect(gray, cascade)

vis = img.copy()

draw\_rects(vis, rects, (0, 255, 0))

if not nested.empty():

for x1, y1, x2, y2 in rects:

roi = gray[y1:y2, x1:x2]

vis\_roi = vis[y1:y2, x1:x2]

subrects = detect(roi.copy(), nested)

draw\_rects(vis\_roi, subrects, (255, 0, 0))

dt = clock() - t

draw\_str(vis, (20, 20), 'time: %.1f ms' % (dt\*1000))

cv2.imshow('facedetect', vis)

if cv2.waitKey(5) == 27:

break

cv2.destroyAllWindows()