OpenCV. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: CV_Image_11_001

Дескриптор, детектор: Гістограма орієнтованих векторів (Histogram of Oriented Gradients, HOG)

OpenCV Docs

HOG cv::HOGDescriptor Struct Reference

Також дивись Learn OpenCV

```
# Завантаження бібліотек
import numpy as np
import sys
import skimage.io as io
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 5
```

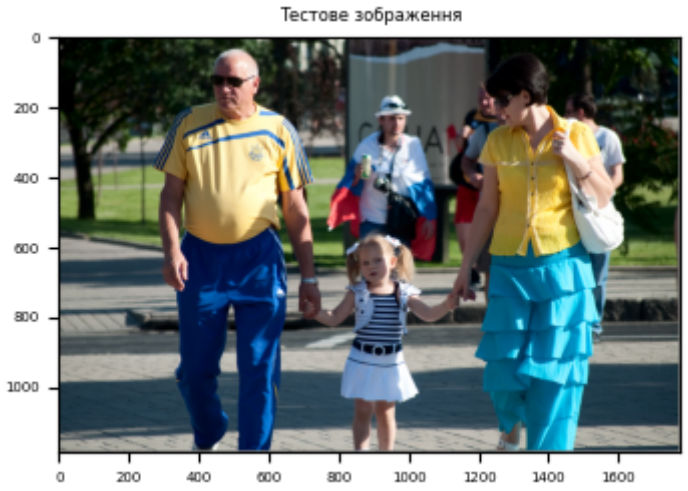
```
# Імпорт CV
import cv2 as cv
```

Завантаження зображення

```
## %%script false
# Завантаження зображення
path = './IMAGES/'
# filename = 'Lena_Gray.jpg'
# filename = 'Lenna.png'
# filename = 'per00057.png'

#filename = 'Lenna.png'
#filename = 'Faces_two_.jpg'
filename = 'Faces_many__.jpg'
#filename = 'Run_image_2.jpg'
test_img_ = io.imread(path+filename)
#  Визначення стркутури та розміру зображення
print('---- TEST IMAGE  -----')
print('IMAGE SHAPE', test_img_.shape, 'IMAGE SIZE', test_img_.size)
irows_num = test_img_.shape[0]  # кількість рядків
iclms_num = test_img_.shape[1]  # кількість колонок
print('ROWS NUMBER', irows_num, 'CLMS NUMBER', iclms_num)
fig, ax = plt.subplots(figsize=(4, 4))
plt.imshow(test_img_)
plt.title("Тестове зображення")
plt.show()
```

```
---- TEST IMAGE  -----
IMAGE SHAPE (1188, 1778, 3) IMAGE SIZE 6336792
ROWS NUMBER 1188 CLMS NUMBER 1778
```




```python
### Завантаження за допомогою CV
test_img = cv.imread(path+filename, 0)
if test_img  is None:
    sys.exit("Could not read the image.")

# Wait for a key press and close the window
while True:
    cv.imshow('Display window', test_img)
    if cv.waitKey(1) == ord('q'):
        break
cv.destroyAllWindows()
```

Вычисляем градиенты

```python
# Обчислюємо градієнти (СОБЕЛЬ)
imag = np.float32(test_img) / 255.0
# Calculate gradient
gx = cv.Sobel(imag, cv.CV_32F, 1, 0, ksize=1)
gy = cv.Sobel(imag, cv.CV_32F, 0, 1, ksize=1)
```

```python
# Обчислюємо амрплітуди та кути (в град)
mag, angle = cv.cartToPolar(gx, gy, angleInDegrees=True)
```

Демонструємо

```
# Wait for a key press and close the window
while True:
    cv.imshow('GX', gx)
    cv.imshow('GY', gy)
    cv.imshow('G amplitud', mag)
    if cv.waitKey(1) == ord('q'):
        break

cv.destroyAllWindows()
```

Обчислення HOG

```
winSize = (64, 64)
blockSize = (16, 16)
blockStride = (8, 8)
cellSize = (8, 8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
nlevels = 64
hog =
cv.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSig
ma,
                        histogramNormType,L2HysThreshold,gammaCorrection,nlevels)

#compute(img[, winStride[, padding[, locations]]]) -> descriptors

winStride = (8, 8)
padding = (8, 8)
locations = ((10, 20),)

hist = hog.compute(test_img, winStride, padding, locations)
```

```
print(hist.shape)
print('Формат HOG', hist.shape)
print('Размер HOG  (байт)', sys.getsizeof(hist))
print('Обчислена гістограма градієнтів \n', hist)
```

```
(1764,)
Формат HOG (1764,)
Размер HOG  (байт) 7168
Обчислена гістограма градієнтів
 [0.04637223 0.01107085 0.00725341 ... 0.12465177 0.01556181 0.04984713]
```

```
# HOG Face Detector
hog_det = cv.HOGDescriptor()
hog_det.setSVMDetector(cv.HOGDescriptor_getDefaultPeopleDetector())
```

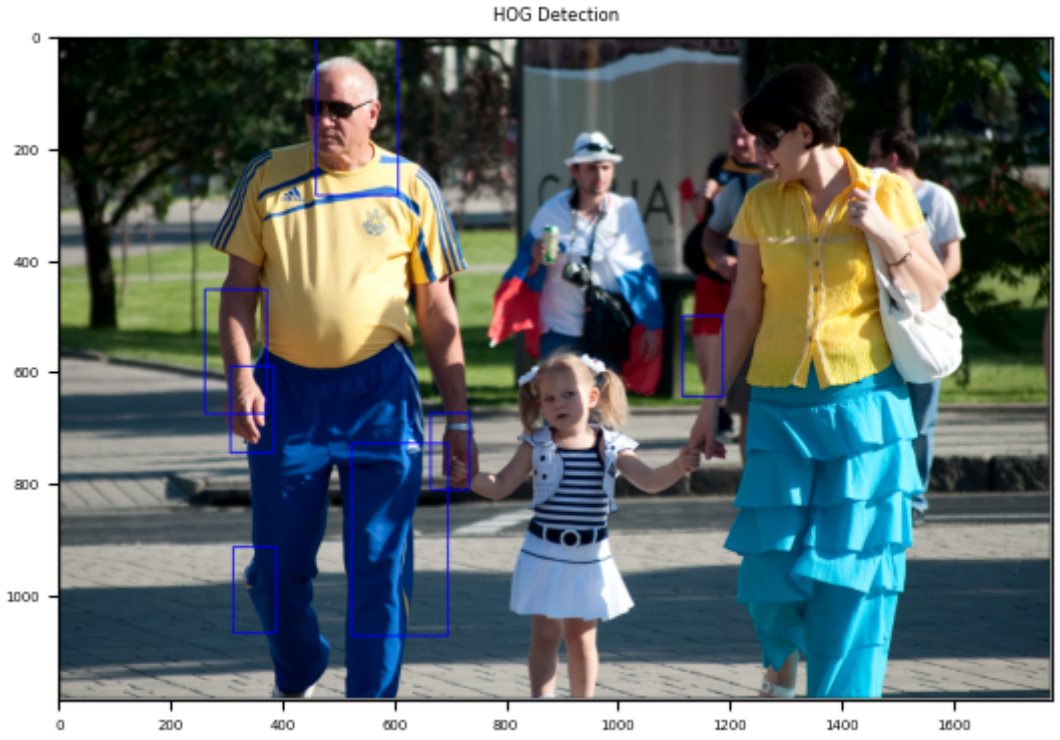
```
# Оновлення зображення
test_img__  = np.array(test_img_)
# Виявлення людей на зображенні (!!! змінюємо groupThreshold)
(rects, weights) = hog_det.detectMultiScale(test_img, winStride=(4, 4), padding=
(8, 8), scale=1.05, groupThreshold = 10.0)

# Параметри BOXів
print ('Кількість прямокунтників', len(rects))
print ('Кут X   Кут Y Ширина  Висота')
for (x, y, w, h) in rects:
    print (x, y, w, h)

# Рисование прямоугольников вокруг обнаруженных объектов
for (x, y, w, h) in rects:
    cv.rectangle(test_img__, (x, y), (x + w, y + h), (0, 0, 255), 2)

# Отображение изображения с обнаруженными объектами
plt.imshow(test_img__)
plt.title('HOG Detection')
plt.show()
```

```
Кількість прямокунтників 7
Кут X   Кут Y Ширина  Висота
525 729 172 344
461 0 147 286
262 454 111 222
1117 501 72 145
308 591 78 155
666 675 69 137
313 914 77 154
```


HOG Detection
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600