Лабораторная работа №5

Компьютерное зрение на языке программирования Python

Распознавание лиц

В OpenCV есть и более «умные» инструменты, использующие машинное обучение. Один из них — модель Haar, которая умеет распознавать лица.

Чтобы воспользоваться ей, скачайте файл haarcascade_frontalface_default.xml, выложенный OpenCV на GitHub.

Также вам понадобится любое изображение с лицами людей. Мы воспользуемся кадром из «Матрицы»

Кадр: фильм «Матрица» / Warner Bros.

Сохраните эти файлы в папку с вашим скриптом, и тогда к ним можно будет обращаться по имени.

Скачать библиотеку можно с помощью инструментов вашей IDE или с помощью командной строки:

```
# Windows
pip install opencv-python
```

---

https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

Импортируйте OpenCV и создайте по переменной для модели Haar и изображения:

```
import cv2


# создаём переменную с файлом модели
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')


# читаем изображение
image = cv2.imread('matrix.jpg')
```

При работе с распознаванием и сравнением объектов используют обесцвеченные версии изображений. Обесцветим и наше:

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Чтобы обнаружить на картинке лица, используйте метод `detectMultiScale`, который применим к модели Haar.

```
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5)
```

Мы передаём методу следующие аргументы:

- `gray_image` — обесцвеченное изображение;
- `scaleFactor=1.1` — шаг масштабирования изображения. Дело в том, что в модели хранятся данные о лицах определённого размера. Если лица на изображении больше или меньше, то алгоритм не обнаружит их. Поэтому при анализе размер изображения меняется: чтобы лица на нём в какой-то момент стали того же размера, что и в модели. Чем меньше scaleFactor, тем точнее распознавание, но тем медленнее оно работает;

- `minNeighbors=5` — какое минимальное число совпадений с моделью должно быть на изображении, чтобы признать объект лицом. Чем больше этот аргумент, тем меньше лиц алгоритм будет обнаруживать, но вместе с тем уменьшается шанс принять за лицо какой-либо другой объект.

Метод `detectMultiScale` возвращает координаты полученных объектов. Используйте эти координаты и функцию `cv2.rectangle`, чтобы нарисовать вокруг лиц квадраты:

```
or (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
```

В переменной `faces` хранится список из четырёх элементов. Каждый из них — список с данными всех найденных объектов:

- `x` — координаты верхнего левого угла объекта по горизонтали.
- `y` — координаты верхнего левого угла объекта по вертикали.
- `w` — ширина объекта.
- `h` — высота объекта.

Функция `cv2.rectangle` получает следующие аргументы:

- `image` — цветное изображение, на котором мы рисуем квадрат.
- `(x, y)` — координаты верхнего левого угла квадрата.
- `(x+w, y+h)` — координаты нижнего правого угла квадрата.
- `(0, 0, 255)` — цвет квадрата в формате BGR.
- `2` — толщина линии квадрата в пикселях.

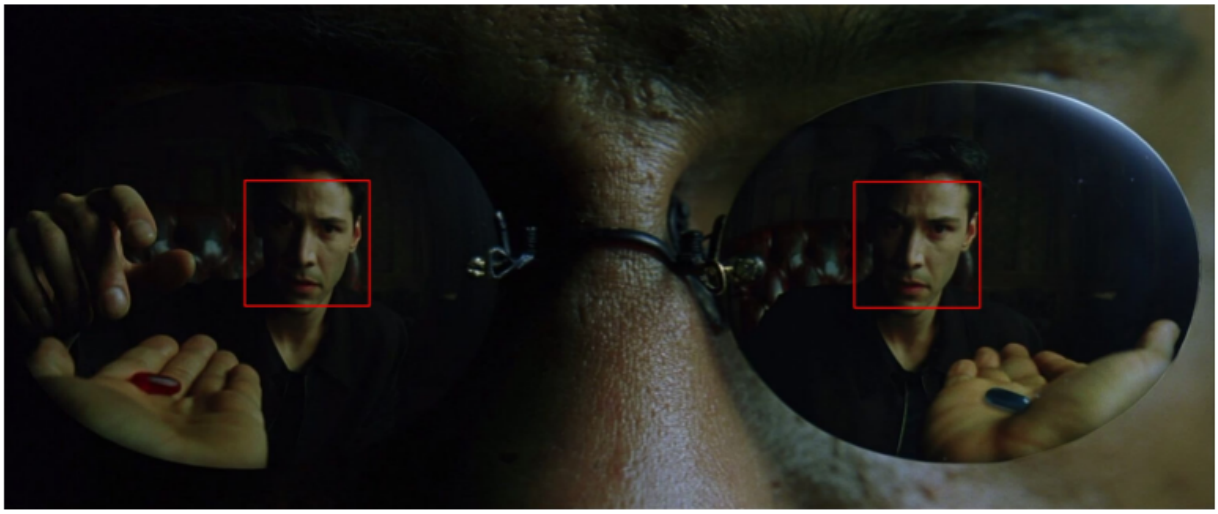
Посмотрим на получившееся изображение:

```
cv2.imshow('found_faces', image)
cv2.waitKey(0)
```

Итоговый код:

```
import cv2

# создаём переменную с файлом модели
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# читаем изображение
image = cv2.imread('matrix.jpg')

# обесцвечиваем изображение
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# находим лица на обесцвеченном изображении
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5)

# на цветном изображении рисуем квадраты там, где нашли лица на обесцвеченном
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

# открываем изображение в отдельном окне
cv2.imshow('found_faces', image)
cv2.waitKey(0)
```

Результат:



Тук-тук, Нео

Кадр: фильм «Матрица» / Warner Bros.

Всё получилось — лица в отражении обведены рамками.