

ТЕМА НА ПРОЕКТА
Разпознаване на обекти от видео чрез
Naar класификатори от OpenCV

Автор:

Галина Цанкова Станева
МГ "Д-р Петър Берон", Варна, 9 клас

Научен ръководител:

доц. д-р Галина Момчева
ВСУ "Черноризец Храбър"

Абстракт

Проектът *Разпознаване на обекти от видео чрез Хаар класификатори от OpenCV* използва *Хаар каскадни класификатори* от *OpenCV* за създаване на собствен софтуер на *Python*, който разпознава лица и други обекти от видео.

Abstract

The project *Video object detection using Haar classifiers from OpenCV* uses *Haar cascade classifiers* from *OpenCV* to create software on *Python* that detects faces and other objects in a video.

Съдържание

1	Въведение	5
2	Материали и методи	6
2.1	Избраният език – Python	6
2.2	Избраната библиотека - OpenCV	7
2.3	Разпознаване на лица с OpenCV в Python чрез Нааг каскадни класификатори	8
2.4	Разпознаване на други обекти с OpenCV в Python чрез тре- ниране на собствени Нааг каскадни класификатори	10
2.4.1	Подготовка на данните за трениране	10
2.4.2	Трениране на каскадния класификатор	11
3	Разпознаване на лица от видео	12
3.1	Работа с програмата	12
3.2	Стъпки	12
3.3	Реализация на Python	14
3.4	Резултати	16
4	Разпознаване на други обекти от видео	17
4.1	Разпознаване на банани от видео	18
4.2	Реализация на Python	18
4.3	Резултати	20
4.4	Разпознаване на стенни часовници от видео	20
4.5	Реализация на Python	21
4.6	Резултати	23

5	Бъдещо развитие	24
5.1	ImageNet	24
6	Заключение	26

Списък на фигурите

1	Нааг функции-свойства за (а)ъгъл, (б)линия, (в) правоъгълник	8
2	Всяка функция-свойство се нанася на изображението	9
3	Примерен описващ текстов файл за негативни примерни изображения	10
4	Примерен описващ текстов файл за позитивни примерни изображения	11
5	Примерен вход	12
6	Някои от лицата, намерени в откъс от <i>Where the Hell is Matt?</i> <i>2008</i>	17
7	Разпределение на лица през кадрите в откъс от <i>Where the Hell</i> <i>is Matt? 2008</i>	18
8	Някои от бананите, намерени в откъс от <i>How to Keep Bananas</i> <i>Fresh for Longer</i>	21
9	Някои от бананите, намерени във видео на плод-зеленчук . . .	22
10	Някои от откритите часовници през кадрите на първото видео	24
11	Някои от откритите часовници през кадрите на второто видео	25
12	Някои от откритите часовници през кадрите на третото видео	25
13	Визуализация на някои синонимни множества в <i>ImageNet</i> . .	26

1 Въведение

Разпознаването на обекти в компютърното зрение е процесът за намиране на обекти от света, който ни заобикаля, като например лица, колела и сгради, в изображения и видео. В този проект, с *разпознаване* се обръщаме към

Задачата, която си поставяме в тази разработка, е да намираме различни обекти от видео. За тази цел използваме *Haar каскадни класификатори* от *OpenCV*, за да създадем собствен софтуер, който разпознава лица, банани и стенни часовници от видео.

Разпознаването на обекти чрез *Haar каскадни класификатори* е ефективен начин за намиране на обекти, предложен от Пол Виола и Майкъл Джоунс[3]. Таблица по-долу сравнява ефективността на класификатори за лица и техните черти, тренирани чрез този метод, спрямо други публични класификатори [6]. Таблицата съдържа информация за минималната големина на модела, който може да бъде засечен, и времето за обработка на целият набор от данни в секунди при използването на *Windows XP* с *Intel Duo T5200* процесор. Класификаторите са тествани върху *CMU*[7] (721 изображения на лица) и *Yale Face*[8] (165 изображения на лица) базите данни.

Данните от таблицата показват, че използваният в проекта класификатор за лице в анфас *haarcascade_frontalface_default* (първи ред) може да се нареди на трето място по ефективност, малко след *haarcascade_frontalface_alt* (втори ред) и *haarcascade_frontalface_alt2* (четвърти ред).

Мишена	Препратка	Наличност	Големина	Етапи	<i>CMU</i>	<i>Yale Face</i>
Лице в анфас	[13, 9]	[12]	24x24	25	66.6 сек	6.3 сек
Лице в анфас	[13, 9]	[12]	20x20	21	70.8 сек	8.9 сек
Лице в анфас	[13, 9]	[12]	20x20	46	60.4 сек	10.1 сек
Лице в анфас	[13, 9]	[12]	20x20	20	63.8 сек	9.0 сек
Лице в профил	[14]	[15]	20x20	26	81.7 сек	14.5 сек
Глава и рамена	[16]	[12]	22x18	30	125.5 сек	11.7 сек
Глава и рамена	[16]	[15]	22x20	19	194.8 сек	19.6 сек
Ляво око	[10]	[15]	18x12	20	62 сек	10.7 сек
Дясно око	[10]	[15]	18x12	20	65.8 сек	10.1 сек
Ляво око	[17]	[12]	20x20	20	109.6 сек	8.7 сек
Дясно око	[17]	[12]	20x20	20	112.8 сек	8.6 сек
Око	[21]	[12]	20x20	24	44.7 сек	5.2 сек
Двойка очи	[10]	[15]	45x11	19	24.3 сек	2.8 сек
Двойка очи	[10]	[15]	22x5	17	31.8 сек	3.4 сек
Двойка очи	[22]	[15]	35x16	19	29.2 сек	3.5 сек
Нос	[10]	[15]	18x15	20	47.8 сек	13.6 сек
Уста	[10]	[15]	25x15	20	42.8 сек	13.8 сек
Уста	[11]	[12]	32x18	18	12.1 сек	2.5 сек

Мишена	Препратка	Наличност	Големина	Етапи	Резултат
Лице в анфас	[13, 9]	[12]	24x24	25	12.8 сек
Лице в анфас	[13, 9]	[12]	20x20	21	14.2 сек
Лице в анфас	[13, 9]	[12]	20x20	46	16.8 сек
Лице в анфас	[13, 9]	[12]	20x20	20	16.4 сек
Лице в профил	[14]	[15]	20x20	26	18.8 сек
Глава и рамена	[16]	[12]	22x18	30	9.8 сек
Глава и рамена	[16]	[15]	22x20	19	38.3 сек
Ляво око	[10]	[15]	18x12	20	9.0 сек
Дясно око	[10]	[15]	18x12	20	8.6 сек
Ляво око	[17]	[12]	20x20	20	9.7 сек
Дясно око	[17]	[12]	20x20	20	9.3 сек
Око	[21]	[12]	20x20	24	9.6 сек
Двойка очи	[10]	[15]	45x11	19	5.7 сек
Двойка очи	[10]	[15]	22x5	17	4.7 сек
Двойка очи	[22]	[15]	35x16	19	8.0 сек
Нос	[10]	[15]	18x15	20	8.9 сек
Уста	[10]	[15]	25x15	20	13.0 сек
Уста	[11]	[12]	32x18	18	18.9 сек

2 Материали и методи

2.1 Избраният език – Python

За реализация на програмите сме използвали езика за програмиране *Python*[25].

Python. Той поддържа множество пакети за обработка на изображения, из-

Мишена	Големина	Етапи	<i>CMU</i>	<i>Yale Face</i>	Собствени резултати
Лице в анфас	24x24	25	66.6 сек	6.3 сек	11.9 сек
Лице в анфас	20x20	21	70.8 сек	8.9 сек	14.2 сек
Лице в анфас	20x20	46	60.4 сек	10.1 сек	17.6 сек
Лице в анфас	20x20	20	63.8 сек	9.0 сек	17.1 сек
Лице в профил	20x20	26	81.7 сек	14.5 сек	20.9 сек
Глава и рамена	22x18	30	125.5 сек	11.7 сек	9.8 сек
Глава и рамена	22x20	19	194.8 сек	19.6 сек	40.1 сек
Ляво око	18x12	20	62 сек	10.7 сек	8.8 сек
Дясно око	18x12	20	65.8 сек	10.1 сек	9.0 сек
Ляво око	20x20	20	109.6 сек	8.7 сек	9.7 сек
Дясно око	20x20	20	112.8 сек	8.6 сек	9.1 сек
Око	20x20	24	44.7 сек	5.2 сек	9.7 сек
Двойка очи	45x11	19	24.3 сек	2.8 сек	5.2 сек
Двойка очи	22x5	17	31.8 сек	3.4 сек	4.2 сек
Двойка очи	35x16	19	29.2 сек	3.5 сек	7.2 сек
Нос	18x15	20	47.8 сек	13.6 сек	9.3 сек
Уста	25x15	20	42.8 сек	13.8 сек	11.3 сек
Уста	32x18	18	12.1 сек	2.5 сек	12.8 сек

куствен интелект, визуализиране на математически функции и много други, като например *OpenCV* [26], *PyBrain* [27] и *Plotly* [28].

Python ни дава възможност за работа с логически оператори, масиви, вектори, структури от данни, функции, цикли, както и сложни обекти от специализирани пакети, като например вградените в *OpenCV* класификатори.

2.2 Избраната библиотека - OpenCV

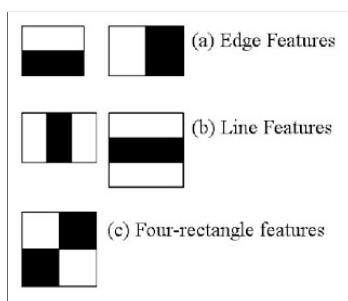
OpenCV (Open Source Computer Vision Library) е софтуерна библиотека с отворен код за *компютърно зрение* и *машинно обучение*. Библиотеката разполага с повече от 2500 оптимизирани алгоритми, набор от такива за компютърно зрение и машинно обучение. Тези алгоритми могат да се използват за откриване и разпознаване на лица, идентифициране на обекти, класифициране на човешките действия в клипове, проследяване на движението във видео, извличане 3D модели на обекти и други.

OpenCV поддържа *Python*, *C++*, *C*, *Java* и *MATLAB* интерфейси, като работи на *Windows*, *Linux*, *Android* и *Mac OS*.

2.3 Разпознаване на лица с *OpenCV* в *Python* чрез Хаар каскадни класификатори

Разпознаването на обекти чрез *Haar* каскадни класификатори е ефективен метод, предложен от Пол Виола и Майкъл Джоунс през 2001 [3]. Този подход е базиран на машинно обучение, в което каскадната функция се тренира с множество позитивни и негативни примери.

Първоначално алгоритъмът има нужда от много позитивни (съдържащи лица) и негативни изображения (несъдържащи лица) за трениране на класификатора. След това, от тези примери се изваждат характеризирани белези. В случая, *OpenCV* определя стойностите на *Haar* функциите-свойства, като всяка такава *функция-свойство* е стойност, определена от разликата на сумата на пикселите под бял правоъгълник с тази на пикселите под черен правоъгълник (фигура 1).

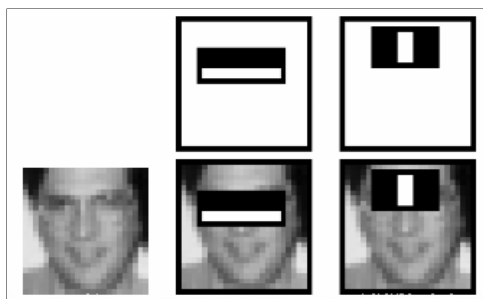


Фигура 1: Хаар функции-свойства за (а)ъгъл, (б)линия, (в) правоъгълник

За всяка функция-свойство се намира най-добрата гранична стойност за определяне на едно изображение като положително или отрицателно. Поради честите грешки, избират се функциите-свойства, които най-добре различават изображение, съдържащо лице, от изображение без такова. Финал-

ният класификатор, който се използва, е сума от тези функции-свойства, които можем да наречем слаби класификатори, защото те, сами по себе си, не могат да намерят лице в изображение, но в комбинация създават силен класификатор.

На този етап за всеки прозорец 24 на 24 пиксела се нанасят 6000 функции-свойства и се проверява дали има лице(фигура 2). Това е неефективно.



Фигура 2: Всяка функция-свойство се нанася на изображението

Затова се въвежда концепцията за *каскади* на класификатори. Вместо да се нанасят 6000 функции-свойства на един регион от изображение, функциите-свойства се групират и се нанасят на етапи. Ако изображението се провали на първия етап, не се продължава.

Нааг каскадният класификатор за разпознаване на лица на *OpenCV* съдържа повече от 6000 функции-свойства, групирани на 38 етапа. Той се намира във файла `opencv/sources/data/haarcascades/haarcascade_frontalface_default.xml` и може директно да се използва в програми.

2.4 Разпознаване на други обекти с OpenCV в Python

чрез трениране на собствени Наг каскадни класификатори

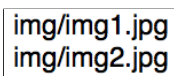
OpenCV включва както детектор, така и треньор, давайки възможност за трениране на собствени класификатори за различни обекти като коли, самолети, банани... Тук е обяснено как става това.

2.4.1 Подготовка на данните за трениране

Нужни са множество позитивни и негативни примерни изображения. Негативните отговарят на изображения без желанния обект. Позитивните - на изображения, съдържащи желанния обект. Негативните могат да бъдат подготвени ръчно, докато позитивните се създават, използвайки *opencv_createsamples* функцията.

2.4.1.1 Негативни примери

Негативните примери са взети от случайни изображения, които не съдържат обекта, който се иска да бъде засечен. Те са записани в специален описващ текстов файл, където всеки ред съдържа името на файл с негативно изображение. Този текстов файл трябва да се създаде ръчно (фигура 3).



Фигура 3: Примерен описващ текстов файл за негативни примерни изображения

2.4.1.2 Позитивни примери

Позитивните примери се създават чрез *OpenCV* функцията *opencv_createsamples*.

Те могат да бъдат създадени от единствено изображение, съдържащо обекта, или от колекция такива.

Трябва да се отбележи, че е добре да разполагаме с голяма колекция от позитивни изображения преди да ги обработим във функцията, за да получим добър класификатор. Например, можем да получим много добър класификатор за напълно еднообразни обекти, като логото на *OpenCV* например, но имаме нужда от стотици, дори хиляди изображения, за да получим класификатори за обекти като лица.

Позитивните примери, подобно на негативните, също са описани в текстов файл, където обаче, освен името на файла се описва колко обекта има в него и се дават съответно координатите на правоъгълниците, ограждащи всеки обект (фигура 4).

img/img1.jpg	1	140	100	45	45
img/img2.jpg	2	100	200	50	50
		50	30	25	25

Фигура 4: Примерен описващ текстов файл за позитивни примерни изображения

2.4.2 Трениране на каскадния класификатор

Тренирането на класификатора става с *OpenCV* функцията *opencv_traincascade*.

Тя предлага множество операции, но тук ще разгледаме само нужните за получаване на работещ класификатор.

Командата *data <ном_към_папка>* показва къде да се запази класификаторът след трениране. *vec <име_на_файл>* подава на програмата името на *vec* файла (който е резултат от функцията *opencv_createsamples*), в който са запазени положителните примери. *bg <име_на_файл>* подава на прог-

рамата името на файла, в който са описани негативните примери. *numPos* <брой_позитивни_примери> и *numNeg* <брой_негативни_примери> отговарят съответно на броя позитивни и броя негативни примери, които да се използват за всяка фаза от тренирането.

Накрая функцията *opencv_traincascade* връща тренираният каскаден класификатор като *xml* файл в зададената директория.

3 Разпознаване на лица от видео

Написахме програма на езика *Python*, която намира лица от видео, като използва осигуреният от *OpenCV*, вече трениран, Хаг каскаден класификатор. Програмата връща намерените лица, заедно с диаграма, визуализираща промяната на броя лица, намерени през кадрите.

3.1 Работа с програмата

За вход, програмата изисква път към видео файла, в който ще търси лица и класификатора, който ще използва за тази цел (фигура 5).

```
Documents/video.mp4
Documents/haarcascade_frontalface_default.xml
```

Фигура 5: Примерен вход

Получените от програмата лица и диаграмата, показваща изменението на количеството намерени лица през кадрите, се запазват в посочена в кода директория, която може да се променя.

3.2 Стъпки

- Получаваме входните стойности (с функцията *sys.argv[индекс]*) - два стринга, отговарящи на пътя до видеото, с което ще се работи, и пътя

до каскадния класификатор, който ще се използва.

- Чрез *OpenCV* функцията *VideoCapture* зареждаме в променливата *video_object* видеото от дадения път.
- Чрез *OpenCV* функцията *CascadeClassifier* зареждаме в променливата *faceCascade* каскадния класификатор за лице от дадения път.
- Започваме цикъл, в който обхождаме всеки кадър от видеото.

Запазваме текущия кадър в променливата *image* чрез *OpenCV* функцията *imread*, като следваме пътя в стринга *pathw*.

Чрез *OpenCV* функцията *cvtColor* превръщаме изображението в черно-бяло в променливата *gray*.

Във *faces* запазваме резултатите от общата функция за разпознаване на обекти *detectMultiScale*. След като я викаме заедно с каскадния класификатор за лица, тя ще разпознава лица.

Посочваме, че ще работим с черно-бялото изображение *gray*.

Посочваме стойността на *ScaleFactor*, тоест с колко ще се прескача между различни големина на изображението[31].

Посочваме стойността на *minNeighbors*, която посочва колко обекти са най-малко трябва да засечени около текущия, за да бъде определен за лице[32].

Посочваме стойността на *minSize*, тоест колко най-малка може да е големината на лицето – по-малки обекти се игнорират.

Извеждаме броят на намерените лица в текущия кадър.

Обхождаме *faces*, където са се запазили координатите *x* и *y* на долният ляв край, ширината *w* и височината *h* на всеки правоъгълник от кадъра, който съдържа лице.

Присвояваме на ново изображение *imgcrop* изрязаната част от *image*, която съдържа лице.

Записваме изображението *imgcrop* в избрана директория.

- Използваме функции от *plotly*, за да създадем диаграма, която показва изменението на броя на намерените лица през кадрите.
- Запазваме получената графика под името *video-faces-data.png*.
- Извеждаме *Done!*, когато сме приключили.

3.3 Реализация на Python

```
import sys

sys.path.append("/Users/Gale/.virtualenvs/cv/lib/python2.7/site-packages")

import cv2

import plotly.plotly as py
py.sign_in('gale_st', 'i2wp2o4poq')

import plotly.graph_objs as go

videoPath = sys.argv[1]
cascPath = sys.argv[2]

video_object = cv2.VideoCapture(videoPath)
faceCascade = cv2.CascadeClassifier(cascPath)

i = 1
```

```

j = 1

resx = []
resy = []
success = True
while success:
    success, frame = video_object.read()
    if success:
        pathw = "/Users/Gale/Documents/ObjectDetector/return-face/shots/cadur_"
        + str(i) + ".bmp"
        cv2.imwrite(pathw, frame)
        i = i + 1
        imagePath = pathw;

        image = cv2.imread(imagePath)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        faces = faceCascade.detectMultiScale(
            gray,
            scaleFactor=1.24,
            minNeighbors=5,
            minSize=(30, 30),
            flags = cv2.CASCADE_SCALE_IMAGE
        )

        print "Found {0} faces!".format(len(faces))

```

```

resx.append(i)
resy.append(len(faces))

for (x, y, w, h) in faces:
    imgcrop = image[y:(y+h), x:(x+w)]
    cv2.imwrite("/Users/Gale/Documents/ObjectDetector/return-face/
found-faces/face" + str(j) + ".jpg", imgcrop)
    j = j + 1

trace = go.Scatter(
    x = resx,
    y = resy,
    name = 'faces'
)

data = [trace]
py.plot(data, filename='faces-line-mode')
py.image.save_as(data, 'video-faces-data.png')
print("Done!")

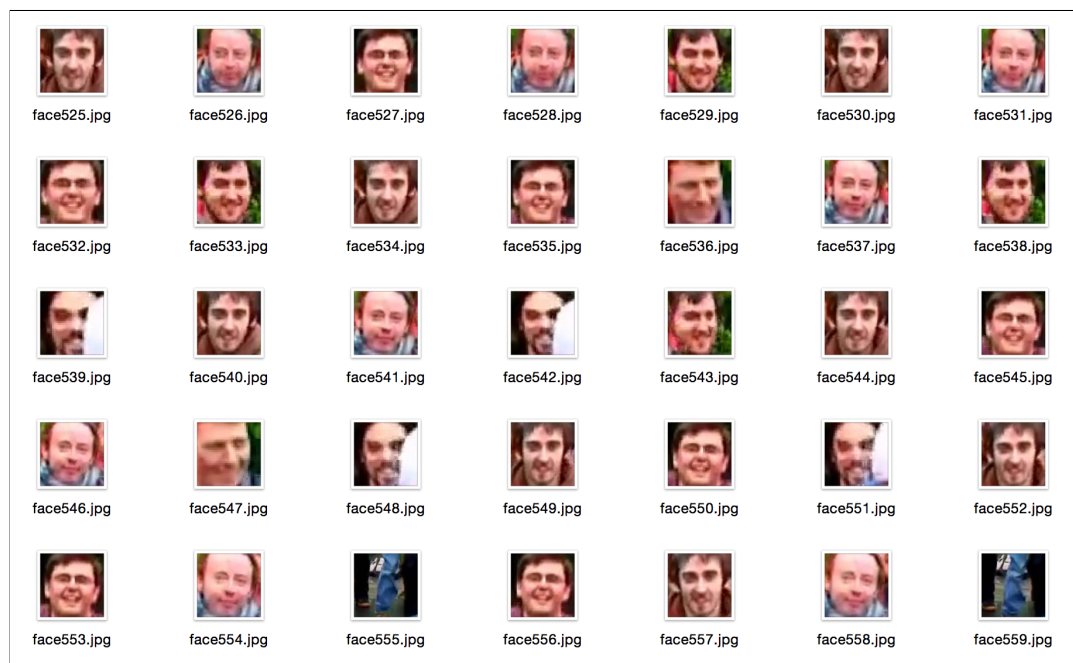
```

3.4 Резултати

Програмата извежда колко лица е намерила за всеки кадър. Запазва намерените в директорията, посочена във функцията на *OpenCV* *imwrite*. Създава диаграма, която показва изменението на броя на намерените лица през кадрите, давайки ни възможност да придобием добра представа кога е имало най-много и кога - най-малко хора.

Ще разгледаме резултати, получени от 56 секунден откъс от видеото *Where the Hell is Matt? 2008*[36][37]. Програмата откри 5082 лица[38] (фи-

гура 6), като грешките бяха около 7%. Получихме диаграма, показваща изменението на количеството намерени лица през кадрите [39] (фигура 7).

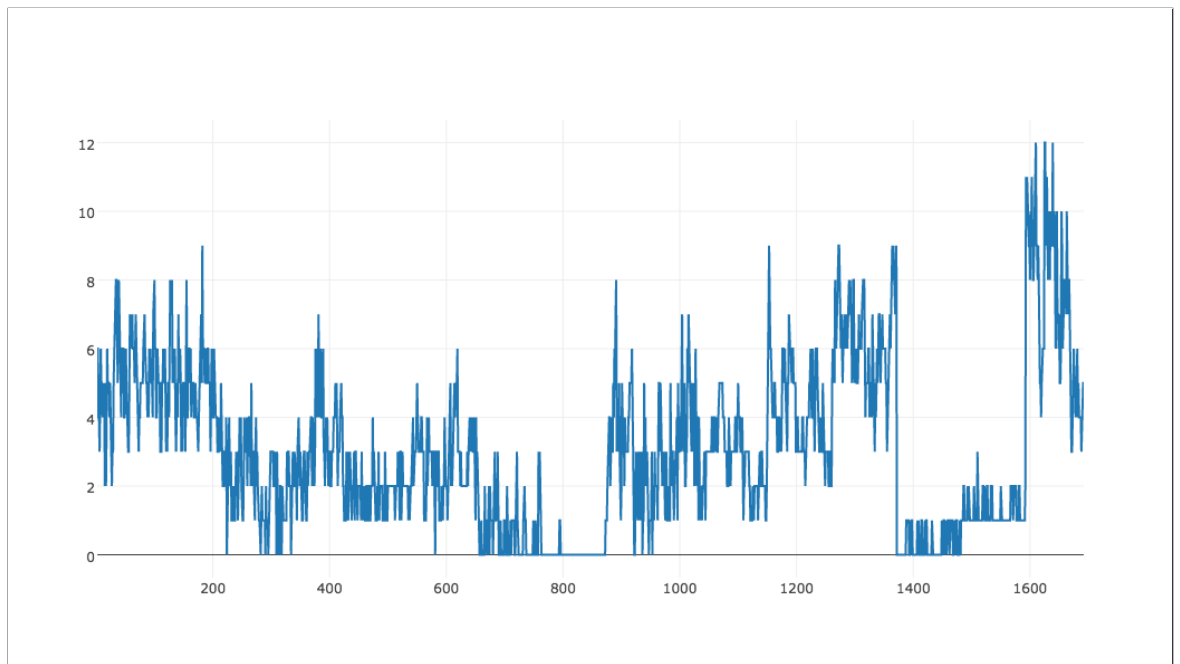


Фигура 6: Някои от лицата, намерени в откъс от *Where the Hell is Matt?* 2008

4 Разпознаване на други обекти от видео

В секцията *Материали и методи* се запознахме с начини, по които може да тренираме собствени *Naar каскадни класификатори*, които да използваме за намиране на различни обекти. Тук разглеждаме две наши програми, които намират банани и стенни часовници от видео.

И двете програми пропуснаха някои от обектите в тестовите видеа. Бъдещото трениране с множество изображения от *ImageNet*[29] ще осигури много по-ефективни класификатори.



Фигура 7: Разпределение на лица през кадрите в откъс от *Where the Hell is Matt?* 2008

4.1 Разпознаване на банани от видео

Работата с програмата и реализацията ѝ на *Python* са аналогични на тези, описани в подсекцията *Разпознаване на лица от видео*. Каскадният класификатор, който се използва, е трениран от *Robin Mehner*[33][34].

4.2 Реализация на Python

```
import sys

sys.path.append("/Users/Gale/.virtualenvs/cv/lib/python2.7/site-packages")

import cv2

videoPath = sys.argv[1]
```

```

cascPath = sys.argv[2]

video_object = cv2.VideoCapture(videoPath)
bananaCascade = cv2.CascadeClassifier(cascPath)

i = 1
j = 1

success = True
while success:
    success, frame = video_object.read()
    if success:
        pathw = "/Users/Gale/Documents/ObjectDetector/return-banana/shots/cadur_"
        + str(i) + ".bmp"
        cv2.imwrite(pathw, frame)
        i = i + 1
        imagePath = pathw;

        image = cv2.imread(imagePath)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        bananas = bananaCascade.detectMultiScale(
            gray,
            scaleFactor=1.24,
            minNeighbors=5,
            minSize=(30, 30),
            flags = cv2.CASCADE_SCALE_IMAGE

```

```

)

print "Found {0} bananas!".format(len(bananas))

for (x, y, w, h) in bananas:
    imgcrop = image[y:(y+h), x:(x+w)]
    cv2.imwrite("/Users/Gale/Documents/ObjectDetector/return-banana/
found-bananas/banana" + str(j) + ".jpg", imgcrop)
    j = j + 1

print("Done!")

```

4.3 Резултати

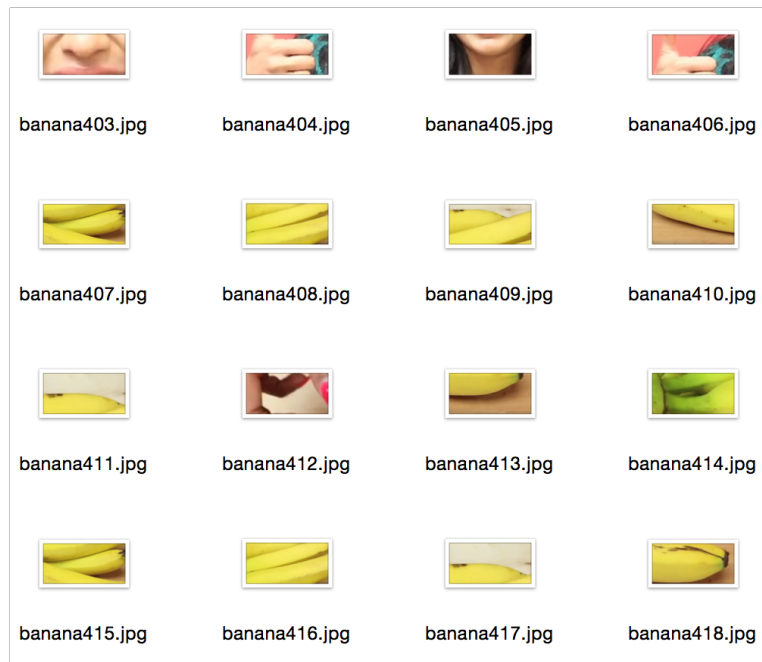
Програмата извежда колко лица е намерила за всеки кадър. Запазва ги в директорията, посочена във функцията на *OpenCV imwrite*.

Ще разгледаме резултати, получени от 24 секунден откъс от видеото *How to Keep Bananas Fresh for Longer*[40][41]. Програмата откри 3534 банана, като около 35% от тях бяха грешки[43](фигура 8).

Ще разгледаме и резултати, получени от собствено видео на плод-зеленчук [42]. Програмата откри 277 банана, като около 75% от тях бяха грешки[?](фигура 9).

4.4 Разпознаване на стенни часовници от видео

Работата с програмата и реализацията ѝ на *Python* са аналогични на тези, описани в подсекцията *Разпознаване на лица от видео*. Каскадният класификатор, който се използва, е трениран от *Celal Caoyn Elgun*[35].



Фигура 8: Някои от бананите, намерени в откъс от *How to Keep Bananas Fresh for Longer*

4.5 Реализация на Python

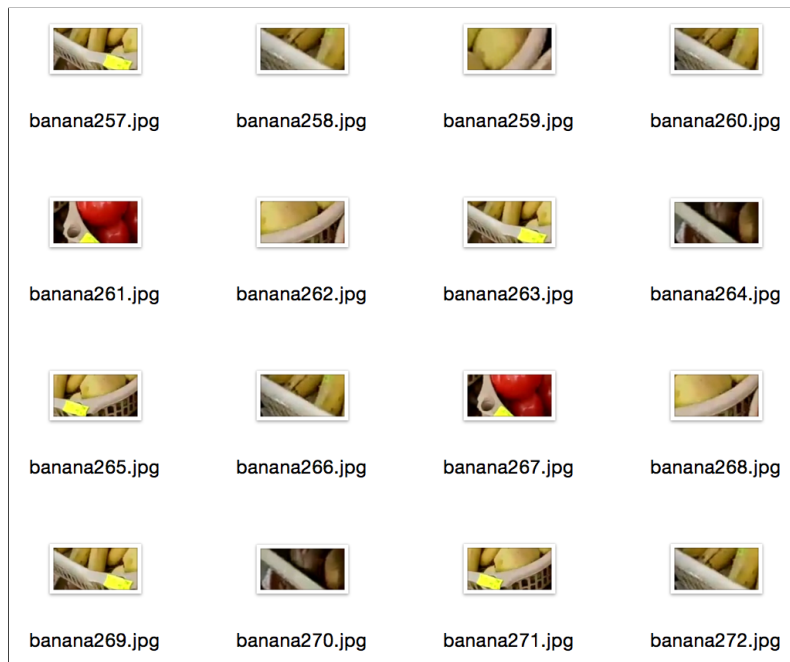
```
import sys

sys.path.append("/Users/Gale/.virtualenvs/cv/lib/python2.7/site-packages")

import cv2

videoPath = sys.argv[1]
cascPath = sys.argv[2]

video_object = cv2.VideoCapture(videoPath)
wallclockCascade = cv2.CascadeClassifier(cascPath)
```



Фигура 9: Някои от бананите, намерени във видео на плод-зеленчук

```
i = 1
```

```
j = 1
```

```
success = True
```

```
while success:
```

```
    success, frame = video_object.read()
```

```
    if success:
```

```
        pathw = "/Users/Gale/Documents/ObjectDetector/return-wallclock/shots/cadur_"
        + str(i) + ".bmp"
```

```
        cv2.imwrite(pathw, frame)
```

```
        i = i + 1
```

```
        imagePath = pathw;
```

```

image = cv2.imread(imagePath)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

wallclocks = wallclockCascade.detectMultiScale(
    gray,
    scaleFactor=1.24,
    minNeighbors=5,
    minSize=(30, 30),
    flags = cv2.CASCADE_SCALE_IMAGE
)

print "Found {0} wallclocks!".format(len(wallclocks))

for (x, y, w, h) in wallclocks:
    imgcrop = image[y:(y+h), x:(x+w)]
    cv2.imwrite("/Users/Gale/Documents/ObjectDetector/return-wallclock/
found-wallclocks/wallclock" + str(j) + ".jpg", imgcrop)
    j = j + 1

print("Done!")

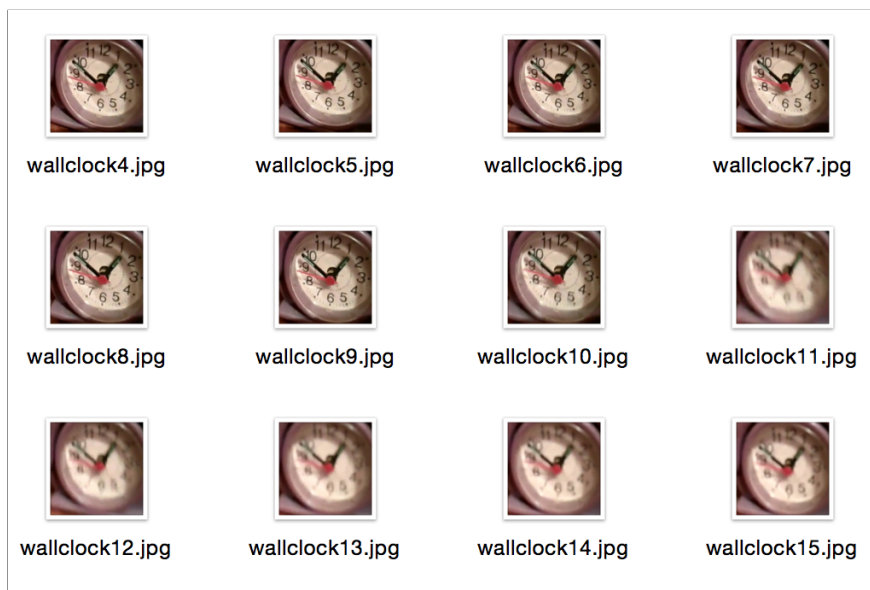
```

4.6 Резултати

Програмата извежда колко лица е намерила за всеки кадър. Запазва ги в директорията, посочена във функцията на *OpenCV imwrite*.

Ще разгледаме резултатите, получени от собствени видеа на различни часовници. За първото[45] програмата откри 20 часовника през кадрите, като нямаше грешки[46] (фигура 10). При второто[47] програмата откри 57

часовника през кадрите, като 50 от тях не бяха сниманият часовник, а части от циферблата му, вероятно заради използването на римски вместо арабски цифри[48] (фигура 11). За третото видео[49] програмата откри 9 часовника през кадрите, като нямаше грешки [50] (фигура 12).



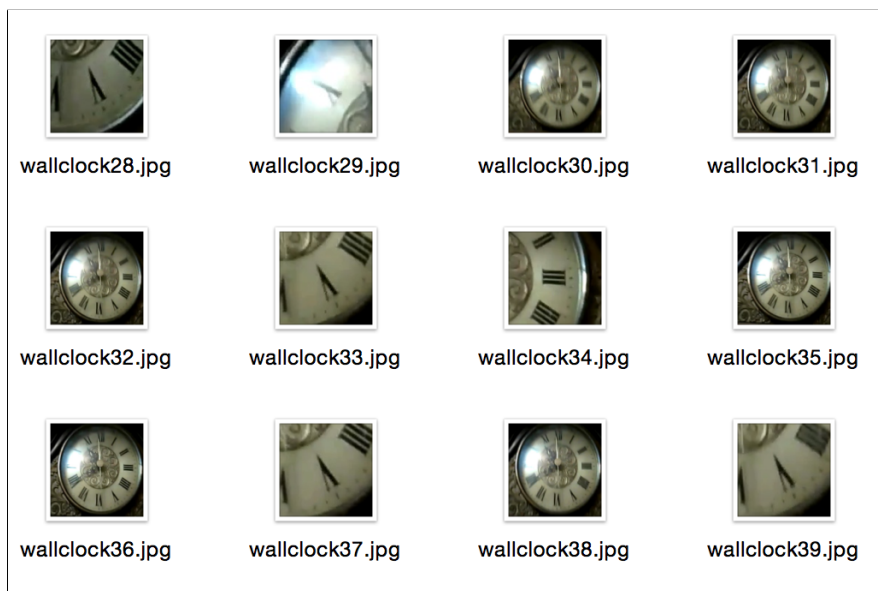
Фигура 10: Някои от откритите часовници през кадрите на първото видео

5 Бъдещо развитие

Поради недостатъчното трениране, класификаторите за банани и стенни часовници не винаги разпознават правилните обекти. Затова, *ImageNet*[29] - база данни, организирана по съдържанието на изображенията, които съдържа, е следваща стъпка при тренирането на по-добри класификатори.

5.1 ImageNet

ImageNet[29] е база данни на изображения, организирани според йерархията на *WordNet*[30]. Всяко понятие в *WordNet*, по възможност описано от

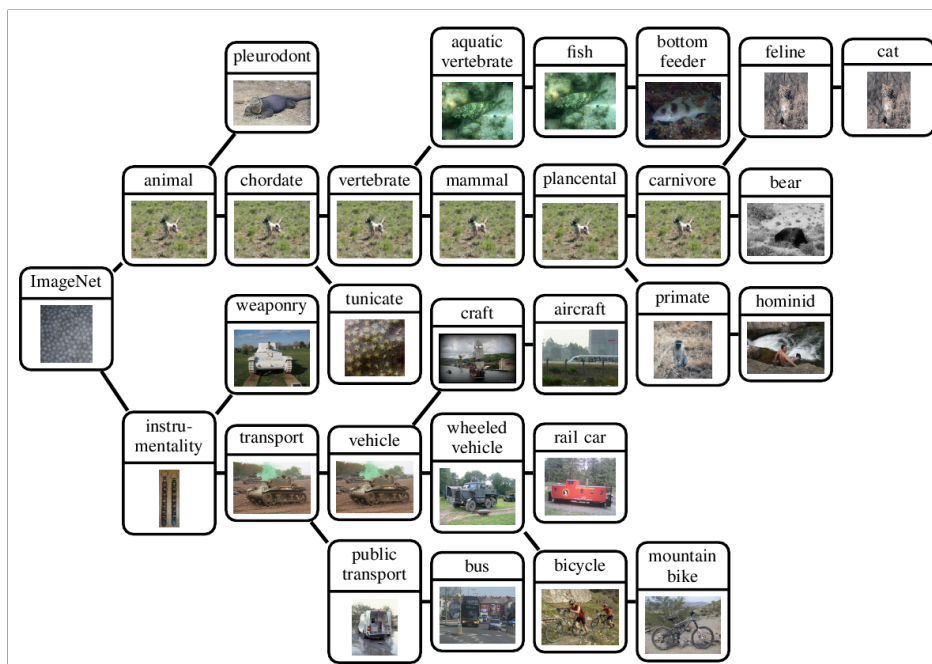


Фигура 11: Някои от откритите часовници през кадрите на второто видео



Фигура 12: Някои от откритите часовници през кадрите на третото видео

няколко думи или фрази, се нарича синонимно множество. Съществуват повече от 100 000 синонимни множества в *WordNet*, повечето от които са съществителни имена (80 000 +). *ImageNet* се стреми да представи средно по 1000 изображения, които илюстрират всяко синонимно множество. За-
ради това *ImageNet* е изключително полезна за създаването на прецизен софтуер в сферата на компютърното зрение.



Фигура 13: Визуализация на някои синонимни множества в *ImageNet*

В бъдеще ще се тренират класификатори за всякакви обекти, като се използват подходящи изображения от *ImageNet* базата данни.

Планира се създаването на уеб приложение, което, с помощта на тези тренирани класификатори, открива желаните от потребителя обекти във видео.

6 Заключение

Целта на проекта е да се разработи софтуер, който намира различни обекти във видео. Затова, написахме три програми на езика *Python*, като използвахме *Нааг каскадни класификатори* от библиотеката за компютърно зрение *OpenCV*. В първата програма разпознаваме лица на хора от видео чрез вграденият класификатор на *OpenCV*. В следващите две разпознаваме

банани и стенни часовници от видео чрез ръчно тренирани класификатори. Планираме тренирането на колекция от класификатори за всяко *синонимно множество* от *ImageNet* базата данни и направата на уеб приложение, в което да могат да се използват за намиране на различни обекти от видео.

Литература

- [1] Joshi P. *OpenCV By Example*, Packt Publishing 2016, page 32 *Object Detection*
- [2] Howse J. *OpenCV Computer Vision with Python*, Packt Publishing 2013, Chapter 4 Appendix B *Generating Haar Cascades for Custom Targets*
- [3] Paul Viola and Michael Jones *Rapid Object Detection using a Boosted Cascade of Simple Features* 2001
- [4] P.F. Felzenszwalb, R.B. Girshick, D. McAllester and D. Ramanan *Object Detection with Discriminatively Trained Part Based Model* PAMI 2010
- [5] Constantine Papageorgiou, Tomaso Poggio *A Trainable System for Object Detection* 2000
- [6] *A Comparison of Face and Facial Feature Detectors Based on the Viola-Jones General Object Detection Framework*
- [7] Carnegie Mellon University, CMU/VACS image database: Frontal face images http://vasc.rh.cmu.edu/idb/html/face/frontal_images/index.html
- [8] vision.ucsd.edu/content/yale-face-database
- [9] R. Lienhart, A. Kuranov, V. Pisarevsky *Empirical analysis of detection cascades of boosted classifiers for rapid object detection DAGM'03, Magdeburg, Germany, 2003, p. 297-304*
- [10] M. Castrillon, O. Deniz, C. Guerra, M. Hernandez *Real-time detection of multiple faces at different resolutions in video streams Journal of Visual Commuication and Image Representation (2007) 130-140*
- [11] Luhong Liang, Xiaoxing Liu, Yibao Zhao, Xiaobo Pi, and Ara V. Nefian *Speaker independent audio-visual continuous speech recognition International Conference on Multimedia and Expo 2002, p. 25-28*
- [12] Intel, *Intel Open Source Computer Vision Library v1.1.0* <http://sourceforge.net/projects/opencvlibrary> (October 2008)
- [13] R.Lienhart, L.Liang, A.Kuranov *A detector tree of boosted classifiers for real-time object detection and tracking IEEE ICME2003, p. 277-280*
- [14] D.Bradley *Profile face detection*, <http://www.davidbradley.info/publications/bradley-iurac-03.swf>
- [15] A.Reimondo *Haar cascades repository*, <http://alereimondo.no-ip.org/OpenCV/34> (2007)

- [16] Kruppa, H., Castrillón Santana, M., Schiele, B. Fast and robust face finding via local context Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS), p. 157–164 (2003)
- [17] Yu, S. *Tree-based 20x20 eye detectors*, <http://yushiqi.cn/research/eyedetection/> (2009)
- [18] Wimmer, M. Eyefinder, <http://www9.in.tum.de/>
- [19] Urtho *Eye detector*, <http://ww38.face.urtho.net/>
- [20] Shan, T. Security and surveillance, <http://www.itee.uq.edu.au/~sas/people.html>
- [21] Hameed, S. *Eye cascade*, <http://umich.edu/~shameem>
- [22] Bediz, Y., Akar, G.B. View point tracking for 3d display systems 3th European Signal Processing Conference, EUSIPCO-2005, (2005)
- [23] http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html
- [24] http://docs.opencv.org/master/dc/d88/tutorial_traincascade.html#gsc.tab=0
- [25] <https://www.python.org/>
- [26] <http://opencv.org/>
- [27] <http://pybrain.org/>
- [28] <https://plot.ly/>
- [29] <http://image-net.org/>
- [30] <https://wordnet.princeton.edu/>
- [31] <http://answers.opencv.org/question/10654/how-does-the-parameter-scalefactor-in-detectmultiscale-affect-face-detection/>
- [32] <http://stackoverflow.com/questions/22249579/opencv-detectmultiscale-minneighbors-parameter>
- [33] https://github.com/mrnugget/opencv-haar-classifier-training/tree/master/trained_classifiers
- [34] <http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>
- [35] <http://alereimondo.no-ip.org/OpenCV/34>
- [36] <https://www.youtube.com/watch?v=zlfKdbWwruY>

- [37] <https://drive.google.com/file/d/0B0oxvFkeBdDIVOp2e1lMNU5oazA/view?usp=sharing>
- [38] <https://drive.google.com/drive/folders/0B0oxvFkeBdDISkY4UE82LXd0ZkE>
- [39] https://plot.ly/2/~gale_st/
- [40] <https://www.youtube.com/watch?v=-visWu3V0gA>
- [41] <https://drive.google.com/file/d/0B0oxvFkeBdIc2pvbDVfaX1hb1U/view?usp=sharing>
- [42] <https://drive.google.com/open?id=0B0oxvFkeBdDIc1I3Zz1RNHBpcDA>
- [43] <https://drive.google.com/folderview?id=0B0oxvFkeBdDIaDVjVWNuZ19JY1E&usp=sharing>
- [44] <https://drive.google.com/folderview?id=0B0oxvFkeBdDIYUc4M1Zxal9jWjA&usp=sharing>
- [45] <https://drive.google.com/file/d/0B0oxvFkeBdDIHRkcWowbzhkaWc/view?usp=sharing>
- [46] <https://drive.google.com/folderview?id=0B0oxvFkeBdDITTh0eXBzRkU4cjg&usp=sharing>
- [47] <https://drive.google.com/file/d/0B0oxvFkeBdDIUGIyVlFBRUNOTlE/view?usp=sharing>
- [48] <https://drive.google.com/folderview?id=0B0oxvFkeBdDIRDV1NHBYaEV2SzQ&usp=sharing>
- [49] <https://drive.google.com/open?id=0B0oxvFkeBdDI1l2cWM4VWd4Zms>
- [50] <https://drive.google.com/folderview?id=0B0oxvFkeBdDISkFIMONIYUpSbjA&usp=sharing>