ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Курсова работа

Тема: Лицево засичане и лицево разпознаване посредством конволюционна невронна мрежа

Изработил:

Огнян Барух

471221021

СОФИЯ

2 0 2 4

СЪДЪРЖАНИЕ

УВОД

Машинното самообучение е основен фактор в множество софтуерни и хардуерни решения. Развитието на машините позволява да бъдат тренирани по-сложни алгоритми с цел постигане на по-точни резултати. Машинното самообучение навлиза в света на технологиите все повече и повече, което ни позволява да заменим човешките усилия с работа на машини. То е използвано както за лични проекти, така и за глобални решения с оглед подобряване на услуги като градски транспорт, имейл, персонални асистенти, преводи и други.

Един от подразделенията на машинното самообучение е лицевото разпознаване и лицевото разпознаване. Тези технологии позволяват автоматизация на много съществуващи процеси, както и елиминират човешката грешка. Съществуват множество приложения на лицевото разпознаване - контрол на достъп до жилищни сгради или бизнес сгради, паспортен контрол на летища и др.

Целта на курсовата работа е да демонстрира работата на лицевото разпознаване чрез трениране на конволюционна невронна мрежа, както и нейното тестване.

ГЛАВА I. ТЕОРЕТИЧНА ОБОСНОВКА

1.1 Лицево засичане

Лицевото засичане е технология, използвана в множество решения за идентифициране на човешки лица в изображения или във видео връзки на живо. То произлиза от засичането на обекти, като в случая търсените обекти са човешки лица. Един алгоритъм за лицево засичане се тренира върху множество изображения на различни човешки лица – различни полове, различни раси, различни черти на лицето. При изпълнение алгоритъмът обхожда пиксел по пиксел даденото изображение и сравнява пикселите със съществуващите снимки на лица, за да открие приликите между тях. В зависимост от приликите между потенциалното засечено лице и снимките на познатите лица, алгоритъмът "взема решение" дали даденият обект е лице или не, като резултатът е процентът сигурност, че разглежданият обект е лице. Съществуват алгоритми, които се тренират по време на изпълнение – при засечено лице алгоритъмът го добавя към множеството от познати лица. По този начин всяко следващо разпознато лице довежда до по-точни резултати. Лицевото засичане намира множество приложения в различни сфери – системи за лицево разпознаване, автоматичен фокус във фотографията, разпознаване на емоции, както и четене по устни. С развитието на алгоритмите и моделите за лицево засичане, то набира все по-голяма популярност в различни решения.

1.2 Лицево разпознаване

Лицевото разпознаване използва резултатите от лицевото засичане, за да сравни непознато лице с познати такива. За разлика от лицевото разпознаване, което отговаря на въпроса: "Чие е това лице", лицевото разпознаване отговаря на въпроса: "Това ли е правилното лице?". Един алгоритъм за лицево разпознаване сравнява характерните черти на непознатото и познатите лица – разстоянието между зениците на двете очи,

разстоянието между външния и вътрешния ъгъл на всяко око, разстоянието между носа и устата и други. Тази информация се записва като вектор със 128 измерения, а впоследствие се сравняват данните от всички измерения, за да се верифицира непознатото лице. Лицевото разпознаване намира широко приложение в системи, свързани със сигурността, като това могат да бъдат както лични системи, така и публични такива. Лицевото разпознаване се използва за контрол на достъпа до различни помещения, както и за отключване на нашите смартфони.

1.3 NumPy

NumPy е най-използваната библиотека за работа с многомерни масиви, написана на Python. Библиотеката предлага множество функционалности, свързани със създаване и обработване на масиви с много измерения. Освен това NumPy има много добре описана документация, която улеснява използването на функционалностите, които библиотеката предлага. Тя позволява запазването на такива масиви в специални файлове с разширение *.npz*, което представлява компресиран асоциативен масив. Друга широко използвана функционалност е, че масивът, използван в NumPy, е обект, който съдържа и друга лесно достъпна информация като брой на измеренията, големина на всяко измерение и други.

1.4 Конволюционни невронни мрежи (КНМ)

Конволюционните невронни мрежи са вдъхновени от биологичните процеси, тъй като връзката между невроните наподобява организацията на зрителната кора на бозайниците. КНМ са проектирани да откриват сложни характеристики във визуалните данни и напоследък бележат значителен напредък главно в разпознаването на изображения. Те трансформират слой по слой стойностите на пикселите на изображенията в резултат, който ги класифицира към един от класовете, дефинирани заедно с тренировъчните

входни данни. Някои слоеве извършват фиксирани математически операции; други съдържат параметри, настройвани така, че резултатите, изчислявани от мрежата, да съответстват) на класа, към който принадлежи съответното изображение на входа.

1.5 Алгоритми за лицево засичане

Съществуват много алгоритми, създадени и оптимизирани специално за целта да засичат лица в снимки. Според много източници MTCNN е един от най-добрите модели откъм ефективност и постигнати резултати. Този модел е предназначен за разпознаване на лица и техните ключови точки, като използва многостепенна каскадна конволюционна мрежа.

MTCNN използва каскада от три мрежи за разпознаване на лица и лицеви ключови точки:

- *PNet (Proposal Network)*: Сканира изображението и предлага кандидат региони за лице.
- *RNet (Refine Network)*: Усъвършенства предложените региони за лице от PNet.
- *ONet (Output Network)*: Разпознава лицеви ключови точки (очи, нос, уста) и предоставя финално усъвършенстване на ограничителните рамки.

1.6 Алгоритми за лицево разпознаване

Както при лицевото засичане, така и при лицевото разпознаване невронните мрежи водят до най-точни резултати. Един от най-известните модели за лицево разпознаване е VGG-Face, разработен от специалисти от Оксфордски университет. Алгоритъмът за лицево разпознаване приема изображение, съдържащо лице и създава т.нар. "*face embedding*", което представлява вектор в 128 измерения, като всяко измерения представлява различна част от този *face embedding*. Този вектор пази информация за много разстояния между характеристичните части на човешкото лице –

разстоянието между зениците на двете очи, разстоянието между вътрешния и външния ъгъл на всяко око и други.

При трениране на алгоритъма за лицево разпознаване се използва метода "*triplet loss*". За неговата реализация се използват три изображения на човешки лица – базова снимка, вярна снимка и грешна снимка. Базовата снимка е снимката, върху която се тренира, вярната снимка е на същото лице, а грешната снимка е на друго лице. По този начин алгоритъмът се "учи" как да разграничава отделните лица по стойностите от различните измерения на вектора *face embedding*. Функцията на *triplet loss* изглежда така:

$$L\,(A,\ P,\ N)\ =\ max\,(\,||\,f(A)\ -\ f(P)\,||\ -\ ||\,f(A)\ -\ f(N)\,||\ +\ \alpha,\ 0)$$

Формула 1.1: Формула за функцията на triplet loss

В тази формула A е базовата снимка, P е вярната снимка, а N е грешната. Функцията f представлява функцията за извличане на *face embedding* от изображение на лице, а α е допустимата граница между вярната и грешната снимка. Тази граница се въвежда, за да има по-голямо разграничение между отделни лица и да може да се разграничават хора, които си приличат в лицето. За трениране на алгоритъма за лицево разпознаване се използва следната формула:

$$J\ =\ \sum_{i=1}^{M} L(A^{(i)},\ P^{(i)},\ N^{(i)})$$

Формула 1.2: Функция за трениране на модела за лицево разпознаване

Резултатът от тази формула е сумата от резултатите на функцията L с всички снимки от колекцията от снимки за трениране. Този резултат се записва във файл с разширение *.npz* и впоследствие се използва при изпълнението на алгоритъма върху тестови лица.

1.7 Алгоритъм за проверка на лице кандидат

Всяко лице кандидат се сравнява с познатите лица, като се изчислява косинусът на ъгъла между двата вектора – лицевата "отливка" на лицето кандидат и лицевата "отливка" на познатото лице. За намирането на косинус на два вектора се използва функцията "`cosine`" (*Формула. 1.3*) от библиотеката "`SciPy`". "`SciPy`" е библиотека с отворен код за по-сложни математически функции.

$$1 - \frac{u \cdot v}{||u||_2||v||_2}$$

Формула 1.3: Формула на функцията "cosine" от библиотеката "SciPy"

Във *Формула 1.3* "*u*" и "*v*" са двата вектора, чиито косинус се търси. В числителя се намира скаларното произведение на двата вектора, а в знаменателя се намира евклидовото разстояние между двата вектора. Тъй като косинус от 0º е 1, то косинусът на двата вектора се изважда от 1, за да може близките вектори да имат стойност близка до 0, а далечните – близка до 1. Така две лица на един и същ човек ще дадат резултат под 0,5 след изпълнение на функцията "`cosine`", а две лица на различни хора ще дадат резултат над 0,5. След изследване на резултатите при тестване с различен брой хора и различен брой изображения се достигна до най-точен праг от 0,3 – ако резултатът от функцията "`cosine`" е по-малък от 0,3, то двете лица са на един и същ човек.

ГЛАВА II. ИЗИСКВАНИЯ И РЕАЛИЗАЦИЯ НА КУРСОВАТА РАБОТА

2.1 Изисквания към курсовата работа

• Курсовата работа да включва програма, която да позволи на потребителя да достъпи камерата на машината си и да снима изображения на потребителя, с които да се тренира моделът.

• Курсовата работа да включва модел за лицево засичане, който да засича всички лица на хора в едно изображение и да ги запазва в масив. При липса на лица в дадено изображение, програмата прекратява изпълнението си.

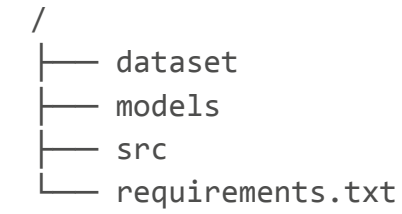
• Курсовата работа да включва програма, която да приема изображение чрез файл с възможни разширения *.jpg*, *.png* и *.jpeg*. Програмата трябва да засече всички лица в изображението и да провери кои лица са на познати хора. Алгоритъмът трябва да връща масив с познатите засечени лица, “Unknown”, ако сред засечените лица няма нито едно познато, или “No faces detected”, ако няма засечени лица.

• Курсовата работа да включва програма, която да тренира алгоритъма, използвайки снимките на познати лица.

• Курсовата работа да включва програма, която да тества алгоритъма със снимки, различни от тези, с които е трениран моделът. Резултатът трябва да се принтира в конзолата и да се запазва във файл, който потребителят да достъпва.

2.2 Файлово и функционално разпределение на курсовата работа

Курсовата работа е съставена от няколко отделни модула, както и от няколко конфигурационни файла, които се изпълняват при стартиране на програмата. Основната файлова структура на сорс кода е изобразена във *Фиг. 2.1*.



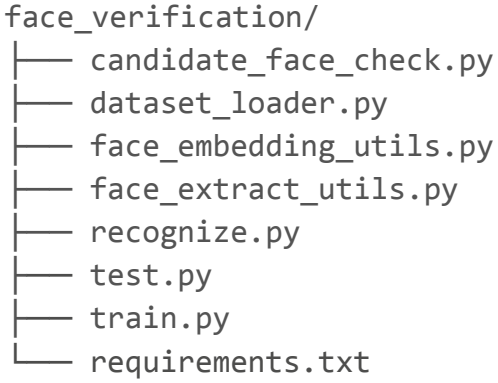
Фиг 2.1: Файлова структура на кода на курсовата работа

Директорията “dataset” съхранява всички изображения - както за трениране, така и за тестване. В директорията “models” се запазват лицевите “отливки” на лицата, с които се тренира моделът. Директорията “src” съдържа Python програмите, описани в изискванията за курсовата работа. Файлът “requirements.txt” съдържа необходимите външни библиотеки за изпълняване на приложението. В него на отделен ред са дефинирани имената на всички библиотеки, както и техните версии, които да се изтеглят.

2.3 Програмна реализация на курсовата работа

2.3.1 Файлова структура

В директорията “src” Python кодът е разпределен в отделни файлове според функционалностите, които изпълнява. Файловата структура на директорията е изобразена във *Фиг. 2.2*:



Фиг. 2.2: Файлова структура на кода на алгоритъма за лицево разпознаване

2.3.2 Създаване на изображенията за трениране

Първата стъпка за създаване на модел за лицево разпознаване е да се създаде набор от снимки, върху които да се тренира моделът. Може да се използва готов набор от изображения, но за улеснение на потребителят е създадена програма за снимане на потребителя. Всички изображения се съхраняват в директорията “dataset”, като тя е разделена на поддиректории, които са имената на хората. Python програмата, отговорна за създаване на колекцията от снимки, е с името “dataset_loader.py”. При изпълнение на програмата се подават двете имена на човека, който ще си прави снимки, така че неговите снимки да бъдат запазени в папка с неговото име, като има 2 начина, по които могат да се подадат аргументите, които са показани във *Фиг. 2.3*.

```
$ python3 dataset_loader.py -f Ognian -l Baruh
$ python3 dataset_loader.py --fname Ognian --lname Baruh
```

Фиг. 2.3: Примерни команди за стартиране на програмата за създаване на изображения

При изпълнение на програмата с грешни аргументи или без такива се изкарва грешка, изобразена във *Фиг. 2.4*.

```
$ python3 dataset_loader.py
Wrong usage!
Correct usage:
$ python3 dataset_loader.py -f <FirstName> -l <LastName>
or
$ python3 dataset_loader.py --fname <FirstName> --lname
<LastName>
```

Фиг. 2.4: Съобщение за грешка при изпълнение на програмата за създаване на изображения

При изпълнение на програмата с “`-h`” или “`--help`” се изкарва наръчник за стартиране на програмата, който е изобразен във *Фиг. 2.5*.

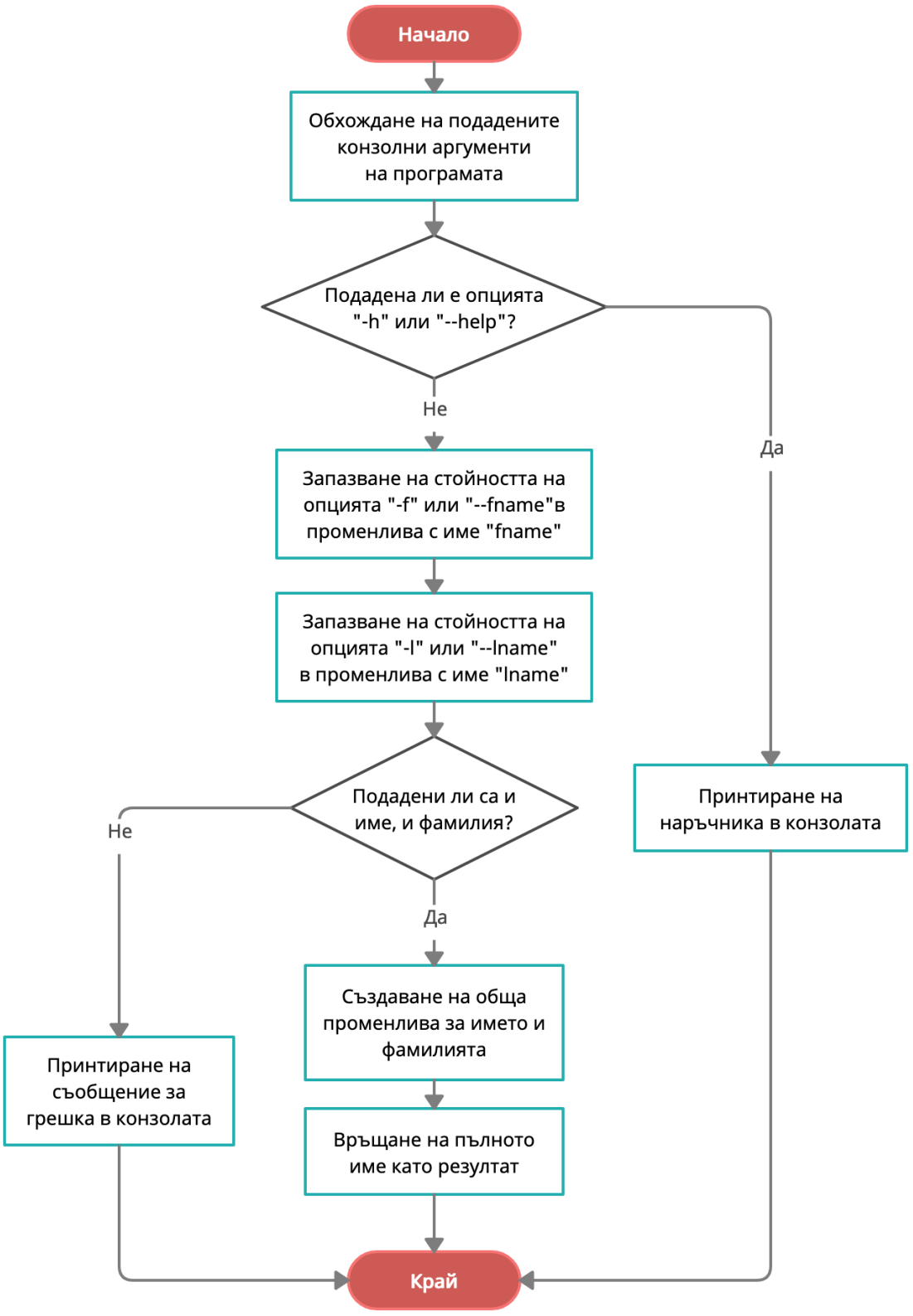
```
$ py dataset_loader.py -h
-h, --help    ->    shows this screen
-f, --fname    ->    first name of the person
-l, --lname    ->    last name of the person
Note: There can't be two people with the same full name!
```

Фиг. 2.5: Наръчник за изпълнение на програмата за създаване на изображения

При нормално изпълнение на програмата процесът за създаване на изображение за трениране е показан на *Фиг. 2.6*.

Фиг. 2.6: Блокова схема на програмата за създаване на изображения

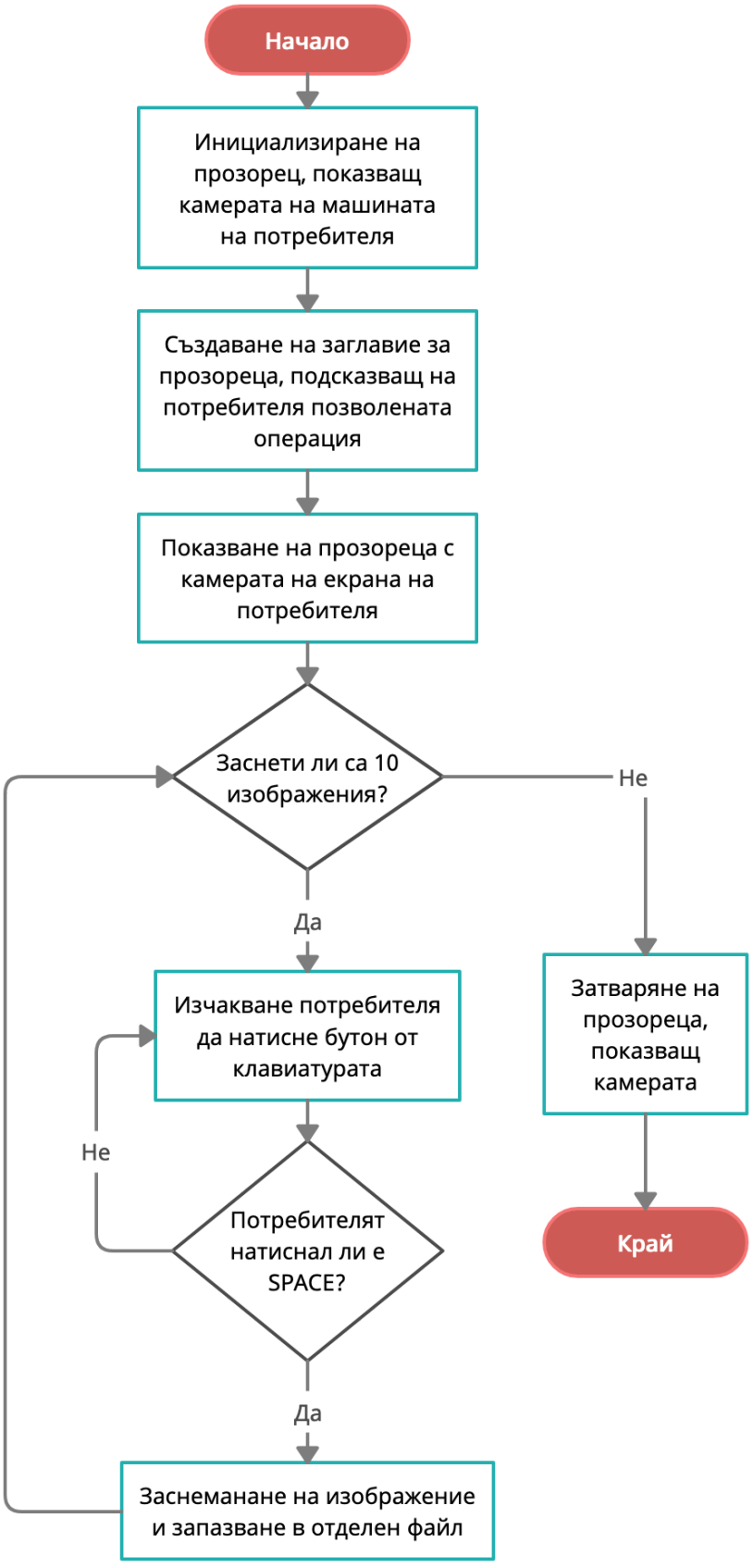
Първата функция, която се изпълнява при стартиране на програмата е функцията "`get_name`", показана във *Фиг. 2.7*, която има за цел да обработи подадените аргументи от потребителя и да генерира пълното име, събирайки първото и последното име на потребителя. Обработката на аргументите се осъществява чрез библиотеката "`getopt`", която позволява лесно използване на подадените аргументи.



Фиг. 2.7: Блокова схема на функцията “get_name”

Втората функция е с наименование “create_dir” и има за цел да създаде директория, в която ще се пазят снимките на потребителя, като името на тази директория е името на потребителя, което той е подал при стартиране на програмата. Като аргумент на функцията се подава пълното име на потребителя, като в рамките на функцията това име се добавя към константата “PATH”, в която е зададен пътят към папката, съдържаща всички изображения за трениране на модела за лицево разпознаване. Създаването на директорията става посредством функцията “mkdir” от вградената библиотека “os”. Като резултат функцията връща пълния път към директорията, където ще се съхраняват потребителските снимки.

Последната функция в програмата е с най-голяма функционалност, като тя е отговорна за снимането на потребителя. За тази цел се използва библиотеката OpenCV, която позволява да се отвори отделен прозорец, в който да се покаже камерата на устройството на потребителя (ако потребителят използва настолен компютър, то ще трябва да свърже камера или да превключи на лаптоп). Функцията приема като аргумент пълния път към директорията, където ще се запазят изображенията. Снимките се заснемат с натискане на бутона “SPACE”. Програмата изпълнява цикъл 10 пъти, което позволява на потребителя да създаде 10 изображения на своето лице. Те биват запазени в създадената директория за снимки на потребителя, като името на всяко едно от изображенията е “image_<пореден-номер-на-изображението>.jpg”. При приключване на програмата се затваря прозорецът с камерата и всички направени снимки се запазват. Наименованието на функцията е “create_images” и тя е изобразена във *Фиг. 2.8*.



Фиг. 2.8: Блокова схема на функцията "create_images"

2.3.3 Помощни функции за лицево засичане

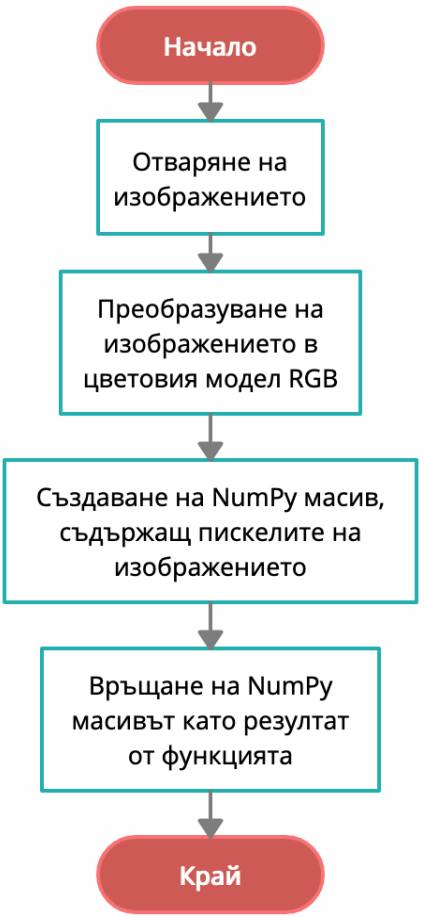
При изпълнението на алгоритъм за лицево разпознаване или лицево разпознаване първата стъпка е да се отделят всички лица от оригиналната снимка. Този процес включва изваждане на лицето от снимката и изправянето му, така че лицето да не е под определен ъгъл. Резултатът от алгоритъм за лицево засичане са координатите на правоъгълник, в който се намира лицето. Посредством тези координати, правоъгълникът може да бъде отрязан от оригиналната снимка и да бъде запазен в нов файл или да бъде използван по време на изпълнението на програмата. За реализацията на проекта е създаден отделен файл, съдържащ всички функционалности, свързани с алгоритъма за лицево засичане, чието име е "`face_extract_utils.py`". В жизнения цикъл на приложението този скрипт не се изпълнява директно, а функции му се използват в други програми като помощни функции. "`face_extract_utils.py`" съдържа 4 помощни функции, като те се използват както в тренирането на алгоритъма за лицево разпознаване, така и в неговото изпълнение.

Първата помощна функция във файла е "`get_pixels_from_file`" (*Фиг. 2.9*), като тя служи за отваряне на изображението и запазването на неговите пиксели в триизмерен масив. Функцията приема като аргумент името на файла и връща като резултат пикселите на съответното изображение.

Функцията за отваряне на изображения конвертира снимката в цветна, за да могат да се използват и черно-бели. Това е необходимо, тъй като всеки модел за машинно самообучение има предварително дефинирани размери на входните данни. Моделът за лицево разпознаване очаква изображения с дължина 160 пиксела, ширина 160 пиксела и с 3 стойности за цвят, които има само цветовият модел RGB (Red – червено, Green – зелено, Blue – синьо). В повечето случаи черно-белите снимки имат само една стойност за цвят от 0 до 255, тъй като в цветовия модел RGB сивият цвят се получава при еднакво количество от трите цвята. Преобразувайки едно изображение от черно-бяло

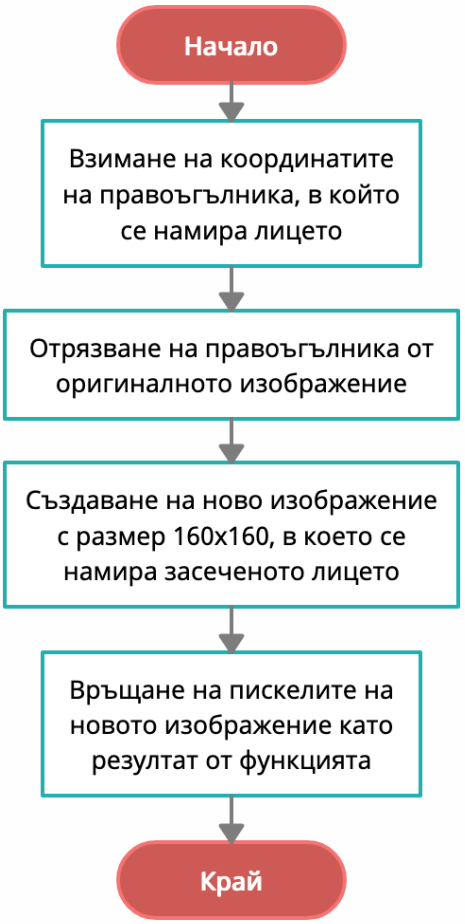
в цветно, функциите заменят единичната стойност със списък от 3 такива – по една за червения, зеления и синия цвят.

След като изображението е преобразувано в цветовия модел RGB, се изваждат пикселите му и се запазват в специални масив от библиотеката “NumPy”. В тази библиотеката масивът се разглежда като клас, който има различни атрибути за брой на измеренията, големина на всяко измерение, брой на всички елементи, тип на елементите и други, както и различни функции, които позволяват по-лесно манипулиране на масива и елементите му. Тези допълнителни функционалности позволяват NumPy масивите да намират широка употреба в моделите за машинно обучение. Освен това моделът за лицево разпознаване приема само NumPy масиви, затова преобразуването на изображението в такъв е наложително.



Фиг. 2.9: Блокова схема на функцията “get_pixels_from_file”

Втората помощна функция във файла се използва за отрязване на лицето от изображението и нейното име е “`crop_face`” (*Фиг. 2.10*). Тя получава два задължителни аргумента и един незадължителен. Двата задължителни аргумента са координатите на правоъгълника, в който се намира лицето, и пикселите на оригиналното изображение. Последният аргумент е “`required_size`”, чиято стойност по подразбиране е `(160,160)`, тоест изображенията на лицата ще бъдат с дължина 160 пиксела и ширина 160 пиксела. Като резултат функцията връща ново изображение, което съдържа само правоъгълника, в който се намира засеченото лице. За изрязване на лицето от изображението се използва операторът “`:`”, който приема начален и краен елемент на масив и отделя този сегмент от оригиналния масив.



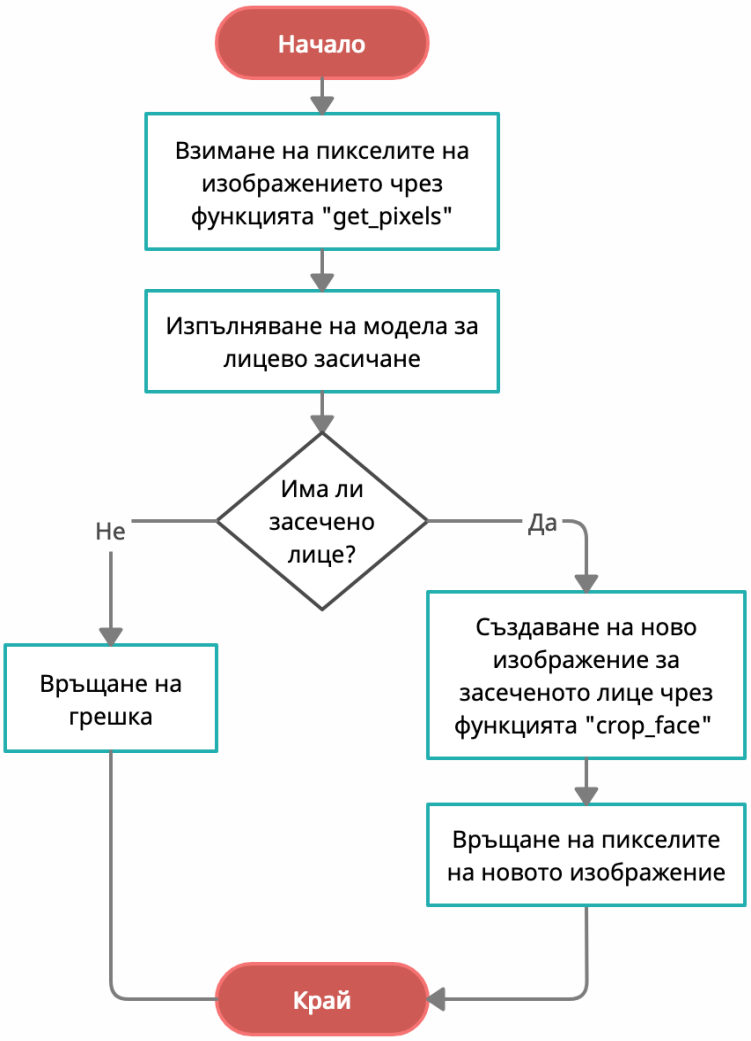
Фиг. 2.10: Блокова схема на функцията “crop_face”

Третата функция във файла се използва само в тренирането на модела за лицево разпознаване и нейното име е “extract_single_face”. Тази функция засича само едно лице в изображението и се използва при трениране с цел оптимизиране на бързодействието на процеса на трениране, тъй като обработката на десетки снимки отнема немалко време. Тя приема като аргументи името на файла, където е изображението, и модела за лицево засичане, а връща масив, съдържащ пикселите на засеченото лице. Функцията “extract_single_face” използва “get_pixels”, за да получи пикселите на изображението. Впоследствие се изпълнява моделът за лицево засичане, който връща като резултат координатите на правоъгълника, в който се намира лицето. Засеченото лицето се изрязва от оригиналното изображение посредством функцията “crop_face”. Ако алгоритъмът за лицево засичане върне като резултат празен масив, то това означава, че не е засечено лице в изображението и функцията “extract_single_face” връща грешка. Тъй като тази функция се използва при трениране на модела за лицево разпознаване, то е очаквано всяко изображение от колекцията за трениране да съдържа лице, затова при обработването на грешката на потребителската конзола се изписва съобщение, изобразено във *Фиг. 2.11*.

```
$ python3 train.py
The face detection model didn't detect a face in image_4.jpg in
Ognian Baruh/
```

Фиг. 2.11: Грешка при липса на засечено лице

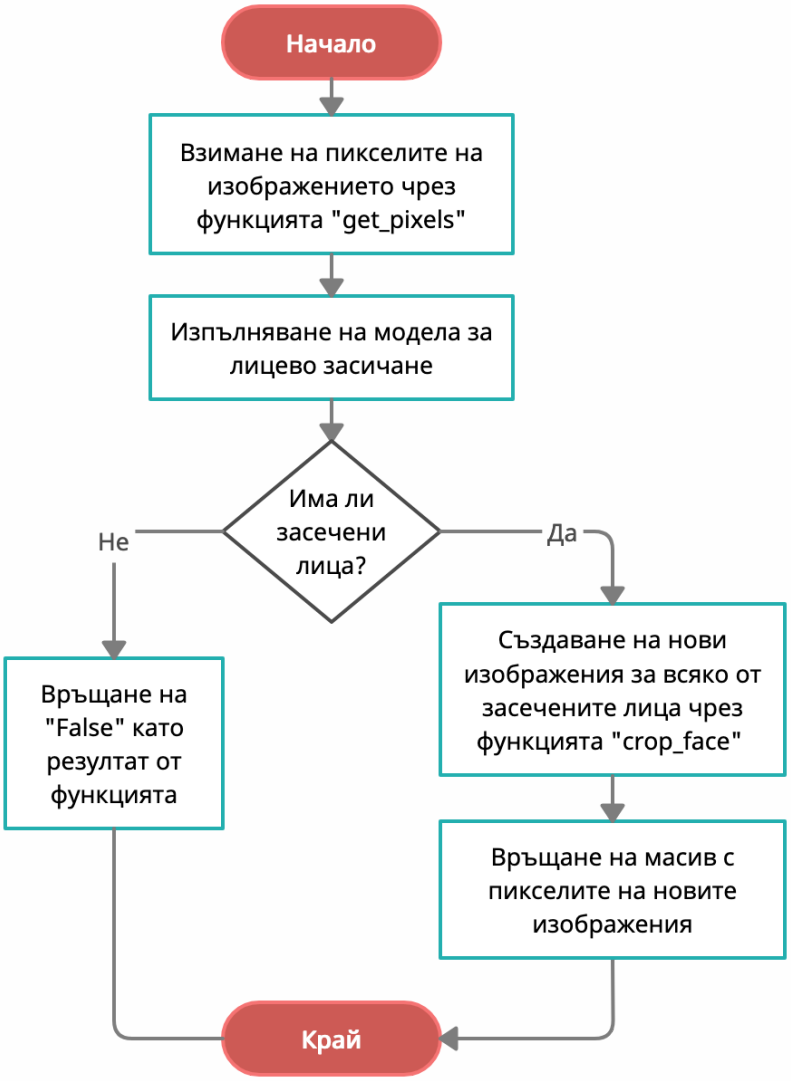
Блоковата схемата на функцията “extract_single_face” е изобразена във *Фиг. 2.12*.



Фиг. 2.12: Блокова схема на функцията “extract_single_face”

Последната функция, с наименование “extract_multiple_faces” (*Фиг. 2.13*), се използва само при изпълнението на алгоритъма за лицево разпознаване, като тя засича множество лица в изображение, а не само едно. Функцията приема името на файла, в което се намира изображението, и модела за лицево засичане и връща масив от нови снимки на отделните лица в оригиналното изображение. Функцията “extract_multiple_faces” се използва за изображения, заснети от камерата, тъй като те могат да съдържат повече от един човек. Ако няма нито едно засечено лице, функцията връща

“`False`”, което при обработката връща резултат от изпълнението на модела за лицево разпознаване: “`No faces detected`”. Снимките в резултатния масив са от тип NumPy масив, за да могат да бъдат подадени на модела за лицево разпознаване, когато трябва да се проверят отделните хора в изображението.



Фиг. 2.13: Блокова схема на функцията “extract_multiple_faces”

2.3.4 Помощни функции за лицево разпознаване

При алгоритмите за лицево разпознаване моделът трябва да постави едно лице в едно 128-измерно пространство и да се намери до кое е най-близко, докато при алгоритмите за лицево разпознаване моделът трябва да пресметне дали дадено лице е достатъчно близко до някое друго, така че да бъде познато. За да се постави едно лице в 128-измерно пространство се създава т.нар. *face embedding* (“отливка” на лицето). Тази лицева “отливка” представлява 128 стойности, които описват лицето на един човек. Това е начинът, по който машината се “учи” да разпознава или да верифицира лица. След извличане на лице от изображение, това лице се подава на модел за лицево разпознаване, който създава тази “отливка” на лицето. Функцията, който създава “отливката” е в отделен файл с име “`get_face_embedding`” (*Фиг. 2.14*). Функцията приема отрязаната снимка на лице, както и модел за лицево разпознаване и връща като резултат “отливката” на лицето. Нейният скелет изглежда по следния начин:

Фиг. 2.14: Блокова схема на функцията “get_face_embedding”

ГЛАВА III. ИЗПЪЛНЕНИЕ И РЕЗУЛТАТИ

3.1 Колекция от данни

Колекцията от изображения, използвана за тренирането и тестването на моделът за лицево разпознаване, е разположена в директорията “dataset”. Вътре в нея са поместени две поддиректории, които служат за съхраняването на изображенията за трениране и тестване. Във “train” и “test” директориите са поставени отделни папки за всеки човек, с когото ще бъде тествана системата. В “test” директорията има и папка “unknown”, в която са поместени непознати за системата лица, така че да се тества поведението ѝ при изображения, които не познава. Файловото разпределение на колекцията от данни е изобразено във *Фиг. 3.1*.

```
dataset/
├── train/
├────── chris_hemsworth/
├────── mark_ruffalo/
├────── robert_downey_jr/
├────── scarlett_johansson/
├── test/
├────── chris_hemsworth/
├────── mark_ruffalo/
├────── robert_downey_jr/
├────── scarlett_johansson/
├────── unknown/
```

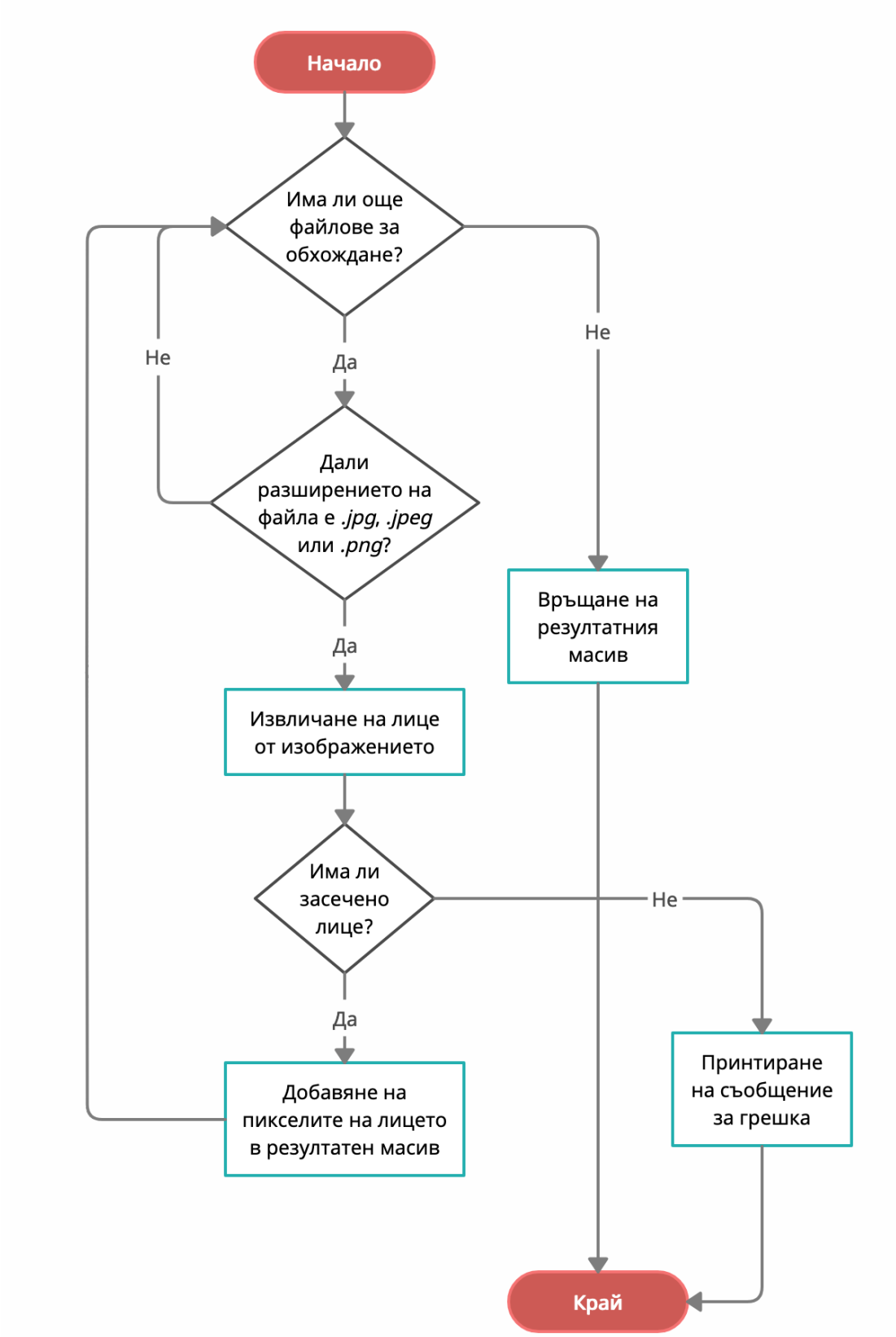
Фиг. 3.1: Файлово разпределение на колекцията от данни

3.2 Трениране на модела за лицево разпознаване

Тренирането на модела за лицево разпознаване е най-фундаменталната стъпка за постигането на точни резултати при изпълнение. Важно е да бъдат подадени достатъчно тренировъчни материали, за да се постигне прецизност при изпълнението. Алгоритъмът за лицево разпознаване се тества с по около

50 изображения на човек, което предразполага за достатъчно голяма прецизност при изпълнението в реална обстановка. Той се “тренира”, като извлича лицевите “отливки” от всяко изображение. Така моделът “знае” къде се намира всяко лице в пространството от 128 измерения и при изпълнение може да се верифицира дали тестваното лице е познато или не. Скриптът за трениране на модела е с име “`train.py`”, като той съдържа три функции – “`load_faces`”, “`load_dataset`” и “`save_embeddings`”.

Функцията “`load_faces`” (*Фиг. 3.2*) извлича лицето от всяко изображение в подадената директория, като всички тези снимки са на един човек, а името на този човек е името на директорията. Функцията приема като аргумент името на директорията и модела за лицево засичане и връща като резултат масив с лицата от всички изображения в директорията. Преди да се извлече лице от изображението, функцията проверява дали файлът е с разширение *.jpg*, *.jpeg* или *.png*, за да е сигурна че се работи с изображение, а не с файл от някакъв друг тип. Ако няма засечено лице в изображението, функцията принтира съобщение за грешка (*Фиг. 2.11*), тъй като тази функция се изпълнява само в рамките на тренирането на модела за лицево разпознаване, а тогава се очаква всяко изображение от колекцията за трениране да съдържа лице. След извличане на всички лица от изображенията в директорията, функцията връща масив, съдържащ тези лица.

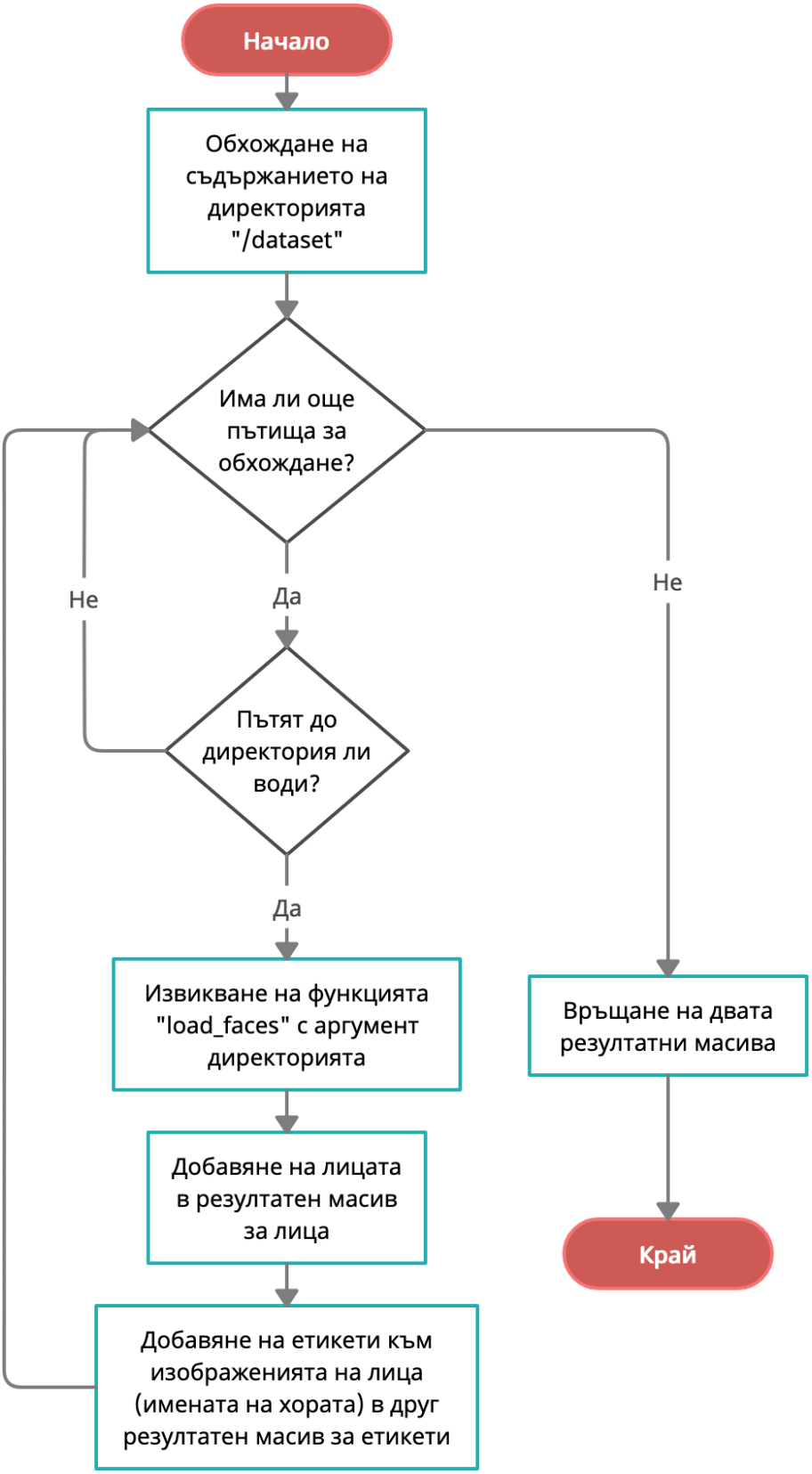


Фиг. 3.2: Блокова схема на функцията "load_faces"

Функцията “load_dataset” (*Фиг. 3.4*) извиква “load_faces” за всяка поддиректория в папката “dataset”, където се намират всички тренировъчни изображения. Всяка поддиректория представлява изображенията на отделен човек. Аргументите на функцията са името на директорията, в която са запазени всички изображения за трениране, която в контекста на дипломната работа е “dataset”, и моделът за лицево засичане, който се инициализира при стартиране на програмата за трениране и се подава към помощните функции. Функцията връща един асоциативен масив, като ключовете са имената на познатите лица, а стойностите са масиви, съдържащи всички лицеви “отливки” за съответния човек. При обхождане на директорията “dataset” функцията “load_dataset” проверява дали всеки път води до директория, използвайки функцията “isdir” от вградената библиотека “os”. Примерен резултат, който връща функцията, е изобразен във *Фиг. 3.3*. В него “<face-pixels-from-image-1>” са пикселите на всяко лице, изрязано от изображението, като в един масив се съдържат само снимките на един човек, като името му се използва за ключ.

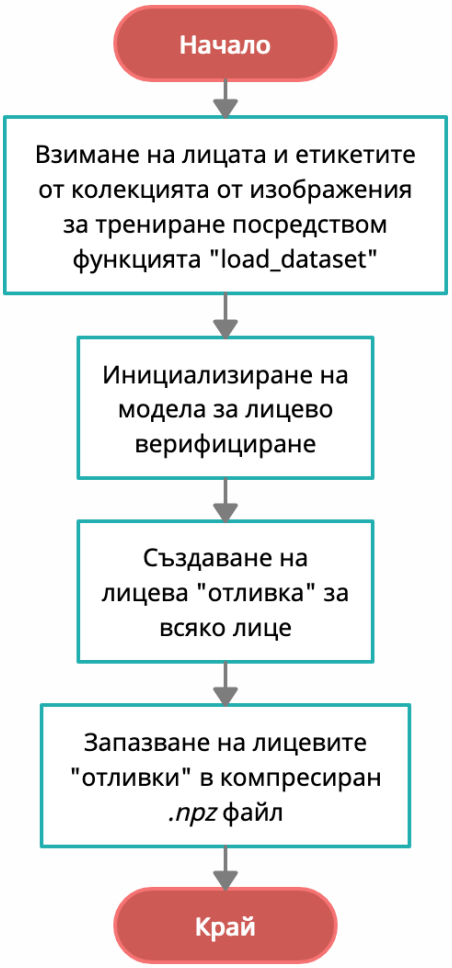
```
{
    Човек-1: [
        <face-pixels-from-image-1>,
        <face-pixels-from-image-2>,
        ...
        <face-pixels-from-image-10>
    ],
    Човек-2: [
        <face-pixels-from-image-1>,
        <face-pixels-from-image-2>,
        ... ,
        <face-pixels-from-image-10>
    ]
}
```

Фиг. 3.3: Примерен резултат от функцията “load_dataset”



Фиг. 3.4: Блокова схема на функцията "load_dataset"

Функцията “save_embeddings” (*Фиг. 3.5*) е отговорна за събирането на всички “отливки” на лицето от колекцията от изображения за трениране и запазването им в специален файл с разширение *.npz*. Тренирането на модела за лицево разпознаване е препоръчително да се осъществява на лаптоп или настолен компютър, тъй като използва много ресурси, които един микрокомпютър на теория може да предостави, но процесът ще отнеме прекалено дълго време. Процесът за трениране използва функцията “load_dataset”, за да получи всички изображения на лица от колекцията за трениране. Впоследствие се създават лицеви “отливки” за всички лица чрез функцията “get_face_embedding”. След изпълнение на функцията, всички лицеви “отливки” се запазват в специален файл с разширение *.npz*. Това представлява архивиран файл, като всичко, запазено в него, се съхранява под формата на асоциативен масив, и при разархивиране на този файл отделните елементи могат да се достъпят с ключовете по подразбиране “arr_0”, “arr_1”, “arr_2” и т.н. Ако са подадени ключове при архивирането, то тогава ключовете по подразбиране се заменят с подадените. За архивиране може се използват функциите “savez” – за обикновено архивиране, и “savez_compressed” – за компресиране преди архивирането, а за разархивиране – “load” от библиотеката “NumPy”. В контекста на дипломната работа се използва “savez_compressed”, като при архивиране се подава ключ “trainData” за речника от лицеви “отливки”.



Фиг. 3.5: Блокова схема на процеса на трениране на модела за лицево разпознаване

3.3 Изпълнение на алгоритъма за лицево разпознаване

След като моделът за лицево разпознаване е трениран, той може да бъде инициализиран и използван в реална ситуация. В контекста на тази курсова работа алгоритъмът за лицево разпознаване се изпълнява върху изображения, които са част от набора от изображения. Помощните функционалностите за лицевото разпознаване са разделени в две функции в два файла – "`recognize.py`" и "`candidate_face_check.py`".

В първия файл се намира функцията "`recognize_faces`", която приема като аргументи изображението, заснето от камерата и тренирания модел. Тя извлича всички лица от изображението и им създава лицеви "отливки", които да се сравнят с познатите лицеви "отливки" от колекцията от изображения за трениране. Ако няма засечено лице в изображението, функцията връща "`No faces detected`" без въобще да се изпълнява алгоритъмът за лицево разпознаване.

Във втория файл е разположена функцията "`check_candidate_faces`", която се извиква от "`recognize_faces`" и сравнява лицата от изображението от камерата с познатите лица, заложени в колекцията. Функцията приема 3 аргумента – речника с всички лицеви "отливки" на тренираните лица, масив с лицевите "отливки" на лицата в изображението от камерата и праг (число от 0 до 1), под който две лица ще бъдат определени като един човек. При изпълнение на функцията "`check_candidate_faces`" се обхождат лицевите "отливки" на кандидатите. Всеки кандидат се сравнява с всеки познат човек, като се изпълнява вложен цикъл, обхождащ всички хора в речника с лицеви "отливки". След това се изпълнява функцията "`cosine`" за "отливката" кандидат и всяка от отливките на съответния познат човек и се намира средно аритметично на резултите на функцията "`cosine`". Двойният вложен цикъл е изобразен във *Фиг 3.6*. В имплементацията не се използват временни променливи, а се използва силата на скриптовия език Python да се пишат вложени цикли на един ред.

```
инициализиране на масив scores
за всеки елемент candidate в candidate_embeddings
   инициализиране на временен речник temp_dict
   за всяка двойка name, known_embeddings в trainData
      за всеки елемент known в known_embeddings
         извикване на cosine с candidate, known
      извикване на mean с всички резултати от функцията cosine
      запазване на резултата от mean в временна променлива t_mean
      добавяне на name, t_mean в temp_dict
   добавяне на temp_dict в scores
```

Фиг. 3.6: Псевдокод за създаване на асоциативен масив с резултати

Резултатите, съхранявани в променливата “scores” (*Фиг. 3.7*), се използват, за да се създаде резултатен масив “names”, съдържащ имената на познатите лица, които съвпадат с кандидатите.

```
[
   {
      'Познат-човек-1': 0.20623254179954528,
      'Познат-човек-2': 0.40129605531692521
   },    // Кандидат 1
   {
      'Познат-човек-1': 0.51230716642703621,
      'Познат-човек-2': 0.26734365280065493
   }     // Кандидат 2
]
```

Фиг. 3.7: Примерна стойност на променливата “scores”

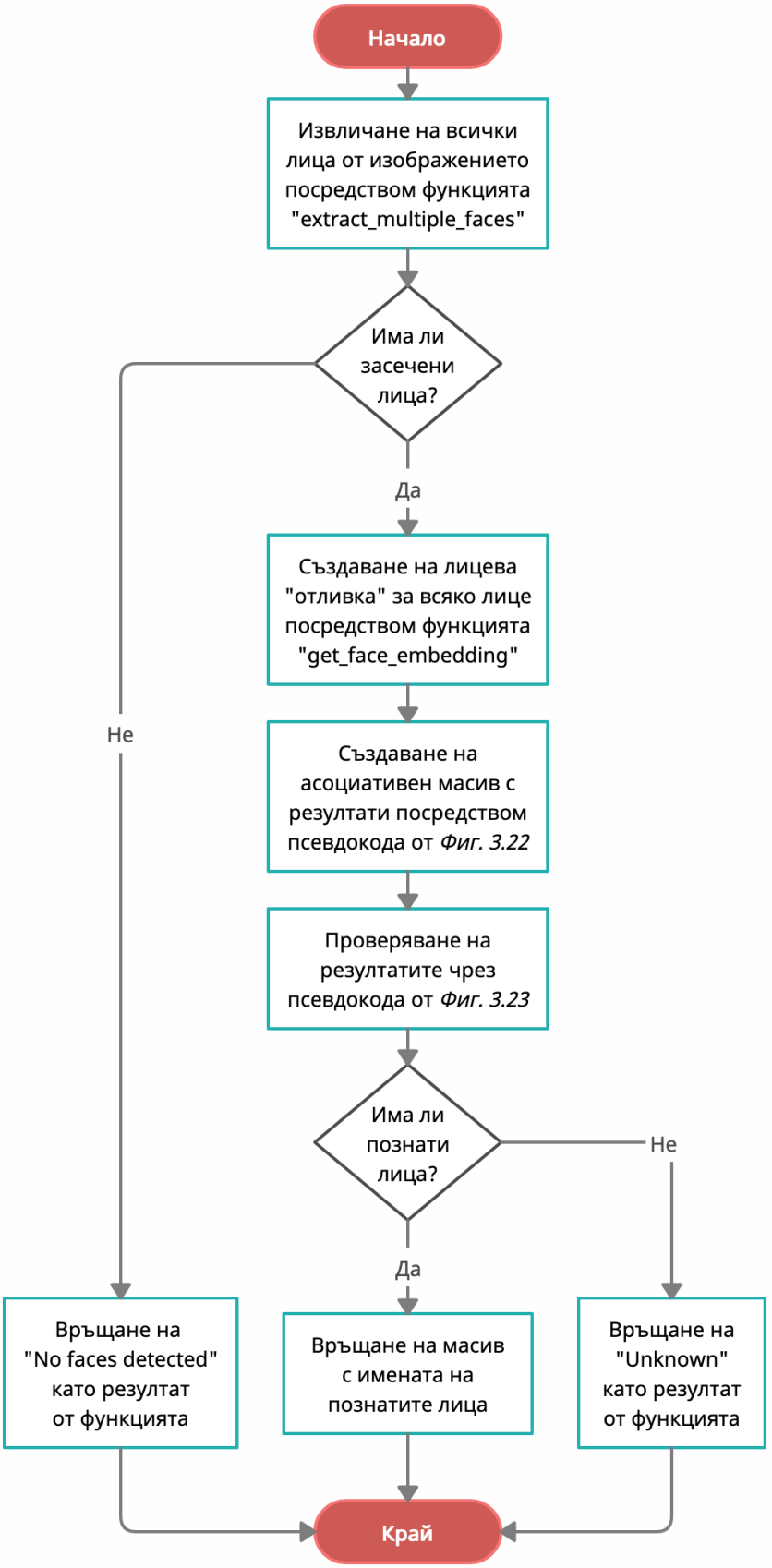
Ако при някои от кандидатите резултатът за някое от познатите лица е под 0,3, то тези две лица съвпадат и името на човека се добавя в резултатния масив. Псевдокодът, проверяващ резултатите е изобразен във *Фиг. 3.8*.

```
инициализиране на масив names
за всеки елемент score_dict в scores
    за всяка двойка name, score в score_dict
        ако score < threshold
            добавяне на name в names
```

Фиг. 3.8: Псевдокод на проверяване на резултатите от функцията "cosine"

Ако няма нито едно познато лице в изображението, то тогава функцията връща резултат "Unknown". Ако има познати лица, функцията връща резултатния масив, съдържащ техните имена.

Блоковата схема на процесът за лицево разпознаване, имплементиран във функциите "recognize_faces" и "check_candidate_faces" е изобразена във *Фиг. 3.9*.



Фиг. 3.9 Блокова схема на процеса на лицево разпознаване

Последната програма “test.py” има за цел да тества невронната мрежа. В него се инициализират моделите за лицево засичане и лицево разпознаване, както и се зареждат “отливките” на познатите лица. Основната функционалност включва обхождането на директорията с изображения за тестване и да се тества всяко едно от тях. Резултатите се запазват в масив от речници, изобразени във *Фиг. 3.10*.

```
[
   {
      'file_path':
“./dataset/test/robert_downey_jr/robert_downey_jr13.png,
      'predicted': <резултатът-от-модела>,
      'actual': “robert-downey_jr”
   },    // Резултат 1
   {
      'file_path':
“./dataset/test/robert_downey_jr/robert_downey_jr14.png,
      'predicted': <резултатът-от-модела>,
      'actual': “robert_downey_jr”,
   },    // Резултат 2
]
```

Фиг. 3.10: Списък с резултати от тестването на моделът за лицево разпознаване

След тестване с всички изображения, се създава статистика, която включва бройката на тестовите изображения, бройката на познати лица, както и процент точност. Резултатите се изписват в конзолата и се запазват в текстови файл. Статистиката от изпълнението на програмата за тестване на моделът за лицево разпознаване е изобразен във *Фиг. 3.11*.

```
----- Stats -----
Test count - 60
Predicted count - 58
Percentage - 96.67
```

Фиг. 3.11: Статистика от тестването на моделът за лицево разпознаване