ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет по телекомуникации

Специалност: Телекомуникации

Курсова работа по Съвременни биометрични технологии

Тема:

Изследване и разработка на подход за разпознаване на 2D лица

Студенти:

инж. Николай Станимиров Проданов	Фак. №: 111321045	Група: 232
инж. Чани Димов Димов	Фак. №: 111321018	Група: 232

Преподавател: **доц. д-р инж. Агата Манолова**

Дата: .....

Подпис: .....

София, 2021

Съдържание

Увод

В наши дни правителствените агенции инвестират значително количество ресурси за подобряване на системите за сигурност за предотвратяване на терористични атаки, които имат за цел да се възползват от недостатъци и слабости в днешните механизми за безопасност. Процедурите за удостоверяване, базирани на значки или пароли, са твърде лесни за хакване.

Биометрията представлява валидна алтернатива. Биометричните системи обработват сурови/необработени данни, за да извлекат шаблон, който е по-лесен за обработка и съхранение, но носи по-голямата част от необходимата информация. Това е много атрактивна технология, защото може да бъде интегрирана във всяко приложение, изискващо сигурност или контрол на достъпа, ефективно елиминирайки рисковете, свързани с по-малко напреднали технологии, които се основават на това, което човек има или знае, а не на това кой в действителност е човек.

Може би най-често срещаните биометрични данни са пръстови отпечатъци и лицеви разпознаване, но са изследвани и много други човешки характеристики: ирис, геометрия на пръста/дланта, глас, подпис. На фиг.1 са показани някои методи за удостоверение.

Фиг. 1 - Популярни методи за удостоверение

Повече от четири пети от потребителите (81%) по света са готови да използват биометрични данни за плащане и удостоверяване плащанията с пръстов отпечатък, а не с ПИН код, според проучване, поръчано от доставчик на биометрични технологии.

В настоящата курсова работа главната цел ще бъде насочена към изследване и разработка на подход за разпознаване на 2D лица. За изпълнението на тази цел са поставени следните задачи:

- Да се обобщят основните характеристики и параметри за лицево разпознаване;
- Да се класифицират видовете разпознаване;
- Да се представят възможностите на библиотеките OpenCV, Keras и Scikit learn;
- Да се обобщят стъпките при процеса на 2D разпознаване на лица;
- Да се изготви анализ на получените резултати, приложимост и изводи

Глава 1: Теоретична част

1.1 Системи за разпознаване на лица

Системите за разпознаване на лица са част от приложенията за обработка на лицеви изображения и тяхното значение като област на изследване нараства в последните десетилетия. Човешкото лице е основен фокус на вниманието в обществото, като играе основна роля в предаването на идентичност и емоция. Разпознаването на лица е технология, която разпознава човек по изображението на лицето му. Системите за разпознаване на лица могат да се използват за предотвратяване на престъпления, видеонаблюдение, сравнение на лица и други дейности свързани със сигурността.

Проблемът с разпознаването на човешкото лице е сложен и много предизвикателен, тъй като трябва да се справи с различни параметри, включително осветление, ориентация на позата, изражение, размер на главата, затъмняване на изображението и фон на лицето. Повечето от предложените методи за разпознаване на лица се занимават с 2D външен вид, като собствено лице (eigenfaces) и fisherfaces; тези методи са най-чувствителни към различна осветеност и поза. През последните години бяха въведени нови методи в обработката на изображения и 2D разпознаването на лица, които не са независими от тези фактори (осветеност, ориентация на позата, изражение).

Собствени лица (Eigenfaces) е метод, който се използва за разпознаване и откриване на лица чрез определяне на дисперсията на лица в колекция от изображения с лица и използване на тези вариации за кодиране и декодиране на лице чрез машинно обучение, без наличието на пълната информация, това намалява сложността на изчисленията. Машинното обучение е метод за анализ на данни, който автоматизира изграждането на аналитични модели. То е клон на изкуствения интелект, базиран на идеята, че системите могат да се учат от данни, да идентифицират модели и да вземат решения с минимална човешка намеса. Методът на собствените лица се базира на собствени вектори, те се извличат от ковариационната матрица на разпределението на

вероятностите върху високомерното векторно пространство на изображенията на лицата. Самите собствени лица образуват основен набор от всички изображения, използвани за конструиране на ковариационната матрица. Това води до намаляване на размерите, като позволява на по-малкия набор от основни изображения да представят оригиналните тренировъчни изображения. Класификацията може да бъде постигната чрез сравняване на това как лицата са представени от базовия набор.

Набор от собствени лица може да бъде генериран чрез извършване на математически процес, наречен анализ на главните компоненти (PCA) върху голям набор от изображения, изобразяващи различни човешки лица. Неформално, собствените лица могат да се считат за набор от "стандартизирани съставки за лице", получени от статистически анализ на много снимки на лица. Всяко човешко лице може да се счита за комбинация от тези стандартни лица. Например нечие лице може да се състои от средно лице плюс 10% от собствено лице номер 1, 55% от собствено лице номер 2 и дори -3% от собствено лице номер 3 в системата. Забележително е, че не са необходими много собствени лица, комбинирани заедно, за да се постигне точна вариация на лице. Освен това, тъй като лицето на човек не се записва от цифрова снимка, а вместо това като списък със стойности (една стойност за всяко собствено лице в използваната база данни), се заема много по-малко дисково пространство за лицето на всеки човек.

Създадените собствени лица изглеждат като светли и тъмни зони, които са подредени в определен модел. Този модел е начинът, по който различните характеристики на лицето се отделят, сортират и биват оценени. След това има модел за оценка на симетрията, проверява се дали има някакъв стил на окосмяване по лицето, къде е линията на косата, оценка на размера на носа и устата. Възможно е изготвянето на прости и сложни модели за идентифициране.

Фиг. 2 - Разпознаване на лица с помощта на собствени лица (PCA алгоритъм)

1.2 Подобряване (augmentation) и анализ на изображение с дълбоко обучение

Изкуственият интелект е свидетел на монументален растеж в преодоляването на пропастта между способностите на хората и машините. Както изследователи, така и ентусиасти работят върху множество области, за да изобретят иновативните решения в тях. Едно от многото такива области е областта на компютърното зрение.

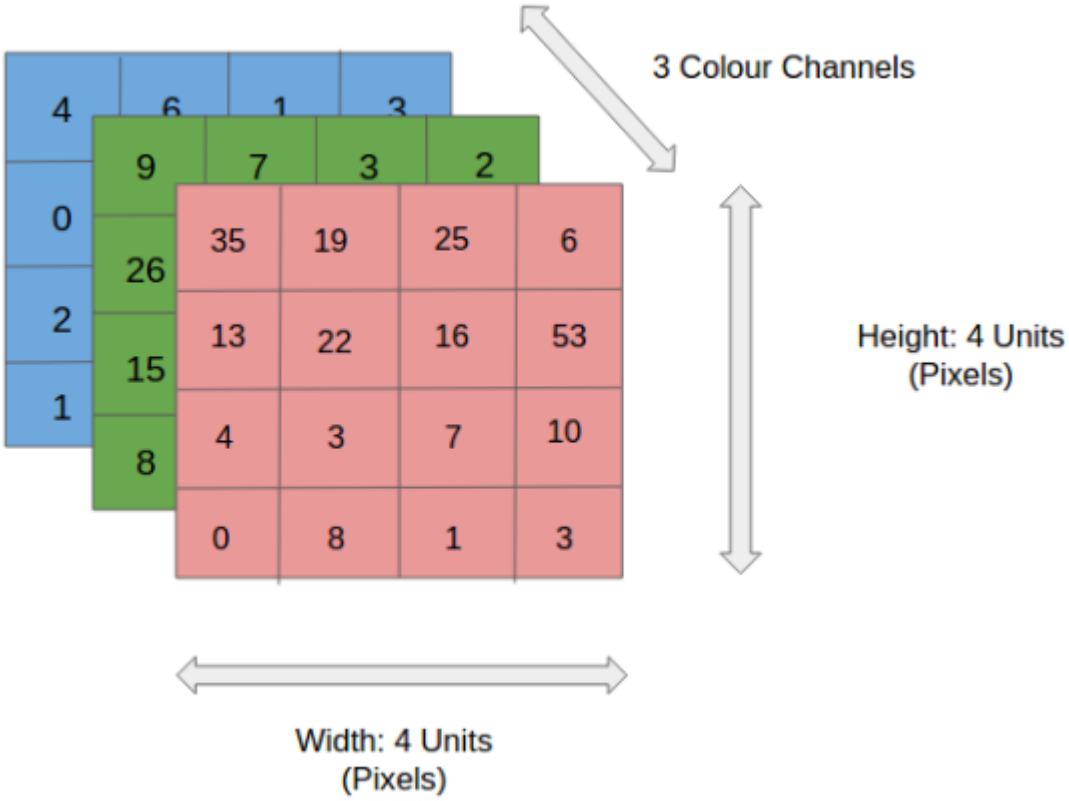
Програмата за тази област е да се даде възможност на машините да виждат света като хората, да го възприемат по подобен начин и дори да използват знанията за множество задачи като разпознаване на изображения и видео, анализ и класификация на изображения, медийно отдих, системи за препоръки , Обработка на естествен език и т.н. Напредъкът в компютърното зрение с метода на дълбоко обучение е конструиран и усъвършенстван с времето, той е концентриран главно върху един конкретен алгоритъм - конволюционна невронна мрежа.

Конволюционната невронна мрежа (ConvNet/CNN) е алгоритъм за дълбоко обучение, който може да приеме входно изображение, да присвои важност (усвоими тегла и отклонения) на различни аспекти/обекти в изображението и да може да ги разграничи едни от други. Необходимата предварителна обработка в ConvNet е много по-ниска в сравнение с други алгоритми за класификация. Докато при примитивните методи филтрите са ръчно проектирани, с достатъчно обучение, ConvNets имат способността да научат тези филтри/характеристики.

Архитектурата на ConvNet е аналогична на тази на модела на свързване на невроните в човешкия мозък и е вдъхновена от организацията на зрителната кора. Отделните неврони реагират на стимули само в ограничен регион на зрителното поле, известен като Рецептивно поле. Колекция от такива полета се припокриват, за да покрият цялата визуална област.

ConvNet е в състояние успешно да улови пространствените и времевите зависимости в изображение чрез прилагането на съответните филтри. Архитектурата изпълнява по-

добро приспособяване към набора от данни за изображение поради намаляването на броя на участващите параметри и възможността за повторна употреба на теглата. С други думи, мрежата може да бъде обучена да разбира по-добре сложността на изображението.



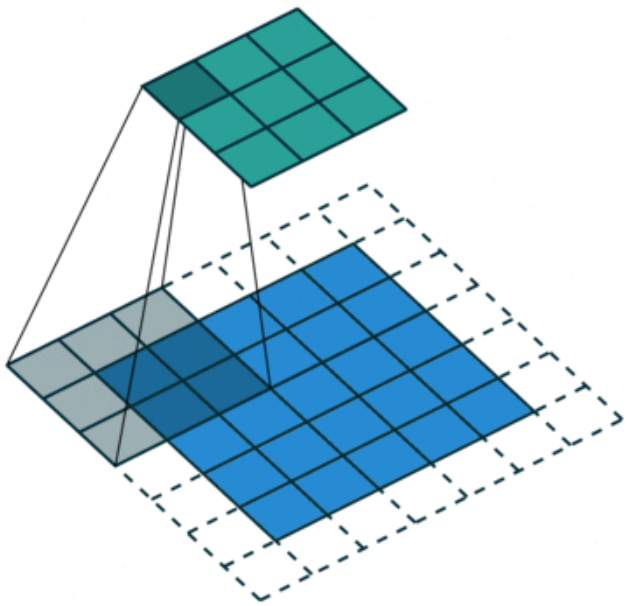
Фиг. 3 - RGB изображение

На фигурата имаме RGB изображение, което е разделено от трите си основни цвята - червено, зелено и синьо (Red, Green, Blue - RGB). Има редица такива цветови пространства, в които съществуват изображения - Сива скала, RGB, HSV, CMYK и т.н.

Можете да си представите колко изчислително интензивни биха станали нещата, след като изображенията достигнат размери, да речем 8K (7680×4320). Ролята на ConvNet е да намали изображенията във форма, която е по-лесна за обработка, без да губи функции, които са от решаващо значение за получаване на добро прогнозиране. Това е важно, когато искаме да проектираме архитектура, която не само е добра в изучаването на функции, но също така е мащабируема до масивни набори от данни.

В случай на изображения с множество канали (например RGB), ядрото има същата дълбочина като тази на входното изображение. Извършва се умножение на матрица и всички резултати се сумират с отклонението, за да се получи конволюционен канал с една дълбочина.

Целта на операцията конволюция е да извлече характеристиките от високо ниво - например ръбовете, от входното изображение. Не е необходимо конволюционните мрежи да се ограничават само до един конволюционен слой. Обикновено първият конволюционен слой е отговорен за улавянето на характеристиките от ниско ниво като цвят, градиентна ориентация и т.н. С добавените слоеве архитектурата се адаптира и към функциите на високо ниво, като ни дава мрежа, която има пълно разбиране на изображения в набора от данни, подобно на начина, по който бихме го направили.



Фиг. 4 - Конволюционна операция за анализ на информация

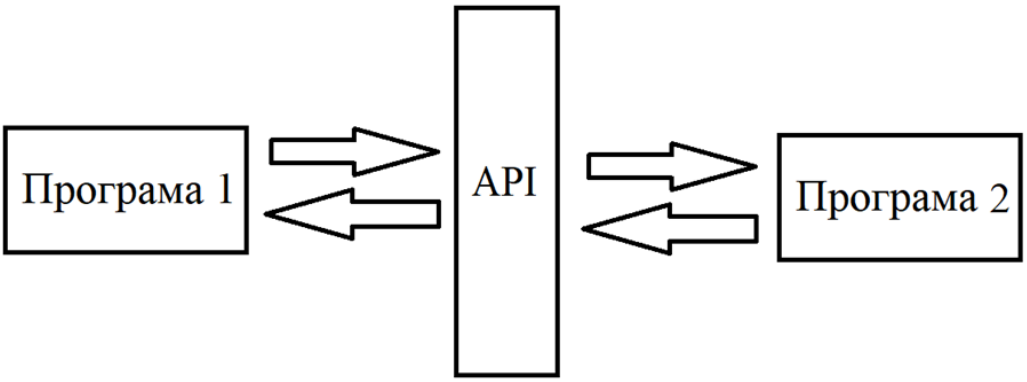
Дълбоките мрежи се нуждаят от голямо количество данни за обучение, за да постигнат добра производителност. За да се създаде мощен класификатор на изображения, използвайки много малко данни за обучение, обикновено се изисква подобряване на

изображението, за да се повиши производителността на дълбоките мрежи. Подобряване на изображението изкуствено създава обучителни изображения чрез различни начини на обработка или комбинация от множество обработки, като произволно завъртане, изместване, срязване и обръщане и т.н.

1.3 Генератор на модели за работа с изображения чрез Keras API

Докато дълбоките невронни мрежи са на мода, сложността на основните рамки е пречка за тяхното използване от разработчиците, които са нови в машинното обучение. Едно предложение е да се представят различни библиотеки и приложни интерфейси (API) които да скрият сложността на математическите модели и да представят лесни за използване функции методи.

API (application programming interface) или програмен приложен интерфейс е подход чрез който една система предоставя програмна функционалност на модули които вървят над нея или искат да я ползват. Пример за това може да е Windows API-а. Когато едно приложение иска да нарисува прозорец с бутони на екрана то извиква конкретните функции предоставени от Windows API-а и Windows операционната система изобразява прозорецът на екрана с коректните бутони. Друг пример са web базираните приложни интерфейси, които като Windows API-то, предлагат услуги или функции на потребителските приложения чрез които те могат да извършват някакви действия.



Фиг. 5 - API

Модели за класифициране изображения могат да бъде създаден с помощта на Keras API. Той е един от водещите библиотеки за невронни мрежи. Написан е на Python и поддържа множество мрежови невронни изчислителни машини като TensorFlow или Theano. Ако вземем в предвид TensorFlow която е библиотека за машинно обучение предоставяща огромен брой функции от ниски операции като работа с матрици до сложни математически модели. Keras абстрахира потребителското приложение от работата с функциите на TensorFlow директно, като му предоставя приложен интерфейс с по опростени функции и модели, които практически използват функциите на TensorFlow отдолу. Ако вместо TensorFlow използваме Theano или друга изчислителна библиотека която Keras поддържа то това няма да се отрази на потребителското ни приложен ние по никакъв начин (в идеалният случай) понеже то извиква функциите на Keras и не знае каква библиотека стои под него. Това позволява на Keras, за да бъде лесен за използване, модулен и лесен за разширяване.

Невронните слоеве, разходните функции, оптимизаторите, схемите за инициализация, функциите за активиране и схемите за регулиране са самостоятелни модули, които можете да комбинирате, за да създадете нови модели. Новите модули са лесни за добавяне, като нови класове и функции. Моделите се дефинират като Python обекти, а не в отделни конфигурационни файлове (но е възможно) на модела.

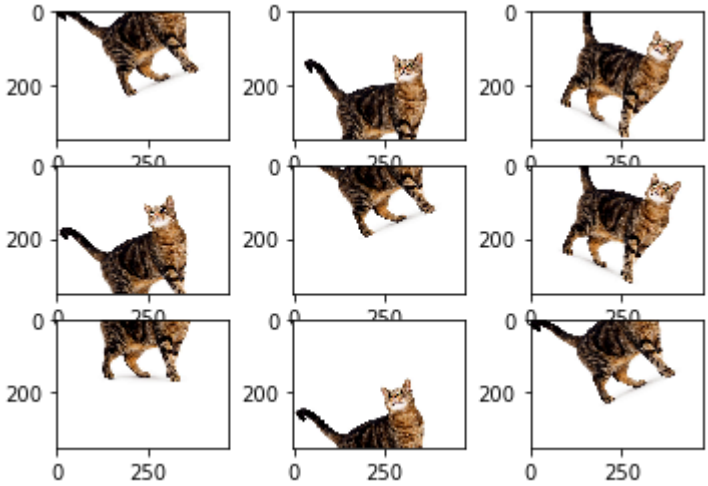
Най-големите причини за използване на Keras произтичат от неговите ръководни принципи, преди всичко този за удобен за потребителя. Отвъд лекотата на учене и

лесното изграждане на модели, Keras предлага предимствата на широкото внедряване, поддръжка за широк спектър от опции за внедряване на продукцията, интеграция с поне пет back-end(заден) двигателя (TensorFlow, CNTK, Theano, MXNet и PlaidML), и силна поддръжка за множество графични процесори и разпределено обучение. Освен това Keras се поддържа от Google, Microsoft, Amazon, Apple, Nvidia, Uber и други. Вътрешната структура на Keras не извършва свои собствени операции на ниско ниво, като тензорни продукти и конволюция, за тези работи разчита на back-end двигателя. Въпреки че Keras поддържа множество back-end двигатели, неговият основен (и по подразбиране) е TensorFlow, а основният му поддръжник е Google.

Keras има богат избор от предварително дефинирани типове слоеве и също така поддържа писането на нови собствени слоеве. Основните слоеве включват плътен (точков продукт плюс отклонение), активиране (трансферна функция или форма на неврон), отпадане (произволно задаване на част от входните единици на 0 при всяка актуализация на обучение, за да се избегне прекомерно монтиране), ламбда (обвиване на произволен израз като обект на слой), и няколко други. Конволюционните слоеве (използването на филтър за създаване на карта на характеристиките) се изпълняват от 1D до 3D и включват най-често срещаните варианти, като изрязване и транспонирани конволюционни слоеве за всяка размерност. 2D конволюция, която е вдъхновена от функционалността на зрителната кора, обикновено се използва за разпознаване на изображения. Слоевете за обединяване се изпълняват от 1D до 3D и включват варианти за максимално и средно разпределение. Локално свързаните слоеве действат като конволюционни слоеве. Съществуват и слоеве за езикова обработка за различни приложения. Шумните слоеве помагат да се избегне прекомерно монтиране.

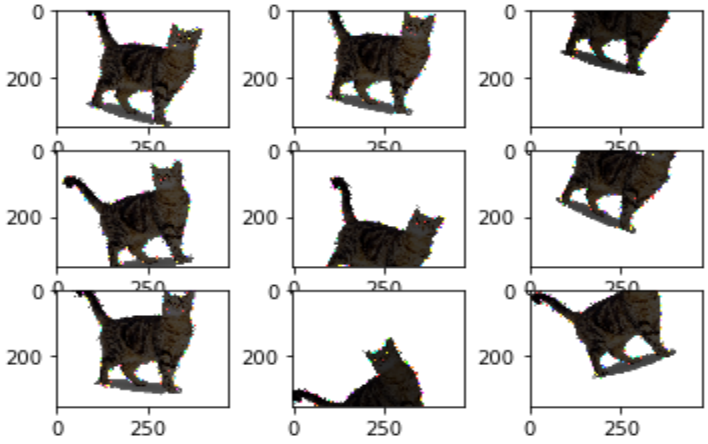
Освен стандартните техники за подобряване на данни, предоставени от генератора на данни за изображения в Keras, можем да използваме персонализирани функции за генериране на подобрение изображения. Например, може да искате да регулирате контраста на изображенията с помощта на разтягане на контраста.

Разтягане на контраста е техника за обработка на изображения, която подобрява контраста чрез преразпределяне (разтягане) на диапазона от стойности на интензитета на изображението до желания диапазон от стойности.



Фиг. 6 - Подобрени изображения чрез разтягане на контраста

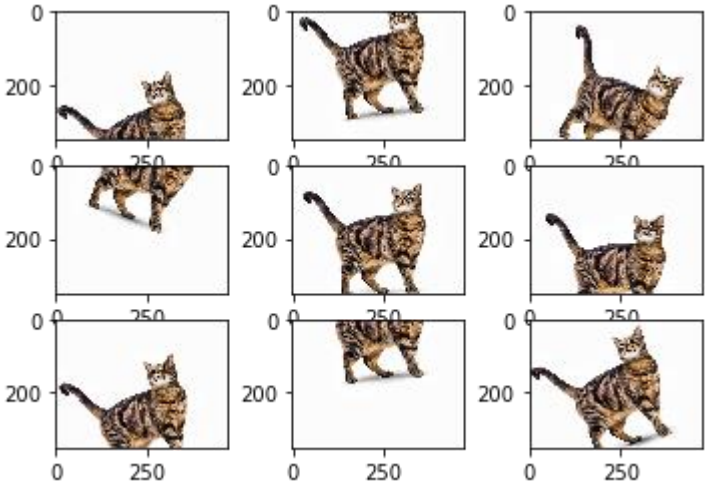
Изравняването на хистограмата е друга техника за обработка на изображения за увеличаване на глобалния контраст на изображението с помощта на хистограмата на интензитета на изображението. Изравненото изображение има функция за линейно кумулативно разпределение. Този метод не се нуждае от параметър, но понякога води до



Фиг. 7 - Подобрени изображения чрез изравняване на хистограмата

неестествено изглеждащо изображение.

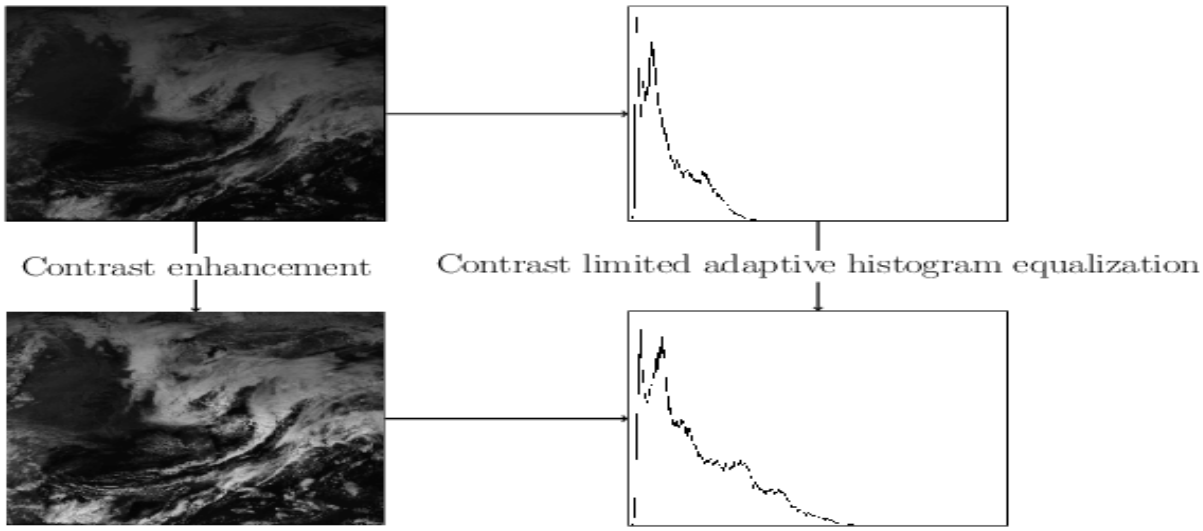
Алтернатива е ***адаптивно изравняване на хистограма***, което подобрява локалния контраст на изображението чрез изчисляване на няколко хистограми, съответстващи на различни участъци от изображение (различава се от обикновеното изравняване на хистограма, което използва само една хистограма за регулиране на глобалния контраст) и ги използва за локален контраст настройка. Въпреки това,



Фиг. 8 - Подобрени изображения чрез адаптивно изравняване на хистограмата

адаптивното изравняване на хистограма има тенденция да преусилва шума в относително хомогенни области на изображението.

Контрастно ограничено адаптивно изравняване на хистограмата е технология разработена, за да се предотврати прекомерното усилване на шума в резултат на адаптивното изравняване на хистограма. По същество той ограничава усилването на контраста от адаптивното изравняване на хистограма чрез изрязване на хистограмата на



Фиг. 9 - Подобрено изображение чрез контрастно ограничено адаптивно изравняване на хистограмата

предварително определена стойност, преди да се изчисли кумулативната функция на разпределение.

Глава 2: Инженерно решение на поставената задача

Една система за разпознаване на лице в реално време може генерално да се раздели на два компонента:

- Детекция на лице в текущото изображение
- Разпознаване на лицето в изображението

Разделението на двата процеса помага за бързодействието на системата. Процесът по детектиране може бързо да премахне изображенията които не съдържат лица. Другата функция на процесът на детекцията е да определи положението на лицето в изображението и по този начин да се подаде към разпознаващото устройство само участъка от изображението в което се намира лицето.

2.1 Детекция на лице

За детекцията ще се използва подходът на Виола Джонс. Това е абстрактен подход който се използва за бързо детектиране и локализиране на обект в 2D изображение.

Хар за характеристично-базиран каскаден класификатор е метод от машинното обучение където класифицираща функция се обучава с голям брой позитивни и негативни изображения. След обучението на функцията може да се подаде изображение и тя да определи дали изображението проявява всички или част признаците за които функцията е обучена. Този метод е публикуване от Пол Виола и Михаил Джоунс през 2001 в списанието "Rapid Object Detection using a Boosted Cascade of Simple Features".

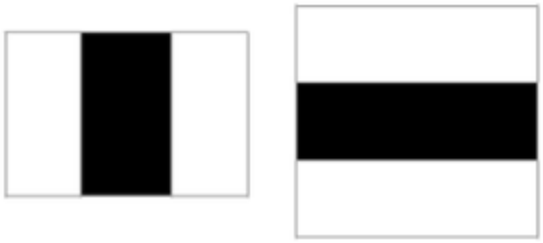
2.1.1 Признаци на Хаара

Идеята тук е да се извлекат признаци от даденото изображение, тези признаци да се сравнят с вече съществуващи признаци на човешки лица и по този начин да се определи дали и къде в даденото изображение се среща човешко лице. За тази цел се използват "признаци на Хаара". Това е метод за съставяне на кърнели или филтри с определени размер и форма с които да се открият признаци в разпределението на яркостта в

изображението. По този начин могат да се засекат и математически да се определят форми и контури в изображението като положението на носа, устни, вежди, очи и други подобни признаци. Предложен е от Алфред Хаара през 1909 година. Изчисляват се подобно на конволиционен кърнел. Има много филтри за извличане на признаци на Хаара но основните са:

Фиг. 10 - Признак по ръб (Edge feature)

При този кърнел се събират се стойностите на пикселите попадащи под белият правоъгълник и се изважда сборът на пикселите от черният правоъгълник. Големината на кърнела може да варира от 4 пиксела до това да обхваща цялото изображение.



Фиг. 11- Признак по линия (Line feature)

Подобно на миналото но тук се събират стойностите на пикселите от 2та бели правоъгълника и се изважда сборът от пикселите под черният правоъгълник.

Фиг. 12 - Признак с 4 правоъгълника (four rectangular feature)

Сборът на пикселите под белите правоъгълници се изважда от сборът на пикселите под черните правоъгълници,

Тъй като тези филтри могат да варират в размер и форма, извличане на признаци за всеки възможен филтър би изисквал изключителен изчислителен ресурс. Например в изображение 24x24, при използване на всички възможни комбинации за филтри, бихме получили вектор с над 160 хиляди различни признака. Нужен е алгоритъм за отделяне само на важните признаци, тези които идват от човешкото лице в изображението, и начин да се определи от кои филтри те се извличат. Затова за да се намали времето за изчисляване на признаците се използват концепции като интегрално изображение и "boost" трениращи алгоритми.

2.1.2 Интегрално изображение

Проблемът при изчисляването на признаците на Хаара е че се сумират едни и същи пиксели при прилагането на филтрите при времевата сложност за сумиране на пикселите под един правоъгълник става O(n2). Ако използваме интегрално изображение можем да намалим времето за изчисление близо до O(1).

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

а

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

б

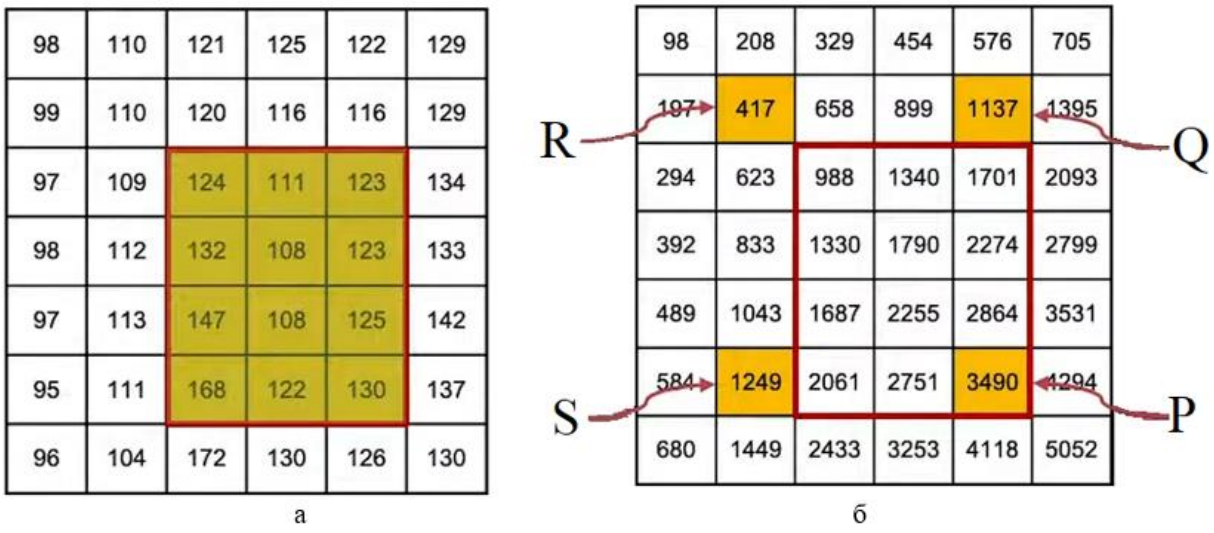
Фиг. 13
а – стойности на пикселите в матричен вид
б – интегралното изображение на а

На Фиг. 13 а – стойности на пикселите в матричен вид б – интегралното изображение на а са представени стойностите на пикселите на чернобяло изображение в матричен вид. А на Фиг. 13 а – стойности на пикселите в матричен вид б – интегралното изображение на а е представено интегралното изображение на изображението от Фиг. 13 а – стойности на пикселите в матричен вид б – интегралното изображение на а. На практика всеки пиксел от интегралното изображение представлява сбор от пикселът на същият ред и колона от нормалното изображение и всички пиксели които чиито редове и колони са по малки. Или изразено математически:

$$P(r,\ k) = \sum_{i=0}^{r}\sum_{j=0}^{k} p(i,j)$$

Където $P(r,\ k)$ е стойността на пиксела от интегралното изображениет, а $p(i,j)$ са пикселите от входното изображение.

По този начин представено изображението ни позволява лесно да намерим сумата на пикселите под всеки възможен правоъгълник. Такъв пример е разгърнат на Фиг. 14 а – изчисляване на сумата на пикселите в правоъгълника б – графично решение с интегрално изображение.



Фиг. 14
а – изчисляване на сумата на пикселите в правоъгълника
б – графично решение с интегрално изображение

Ако искаме да изчислим сборът на пикселите под жълтият правоъгълник то това лесно може да се намери с интегралното изображение. Ако стойността на пиксела P в интегралното изображение представлява сборът на всички пиксели от (0,0) до (5, 4) то от него можем да извадим стойностите на Q и S понеже те представляват сборът на пикселите около този правоъгълник и прибавим стойността на R. Или изразено математически това е:

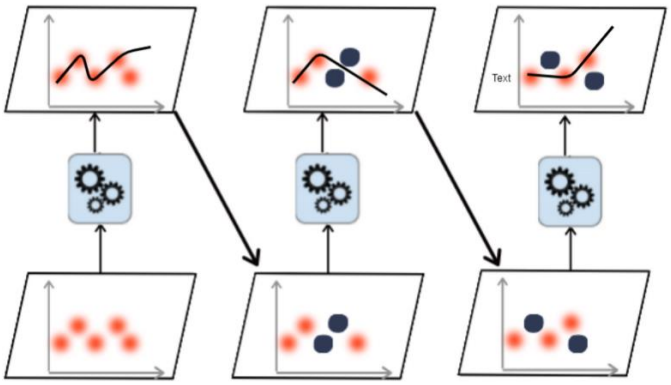
$$X = P - Q - S + R$$

По този начин с 3 прости операции (събиране, изваждане) може да се намери сборът на пикселите под всеки правоъгълник, независимо от размера му.

2.1.3 "boost" трениращи алгоритми и AdaBoost

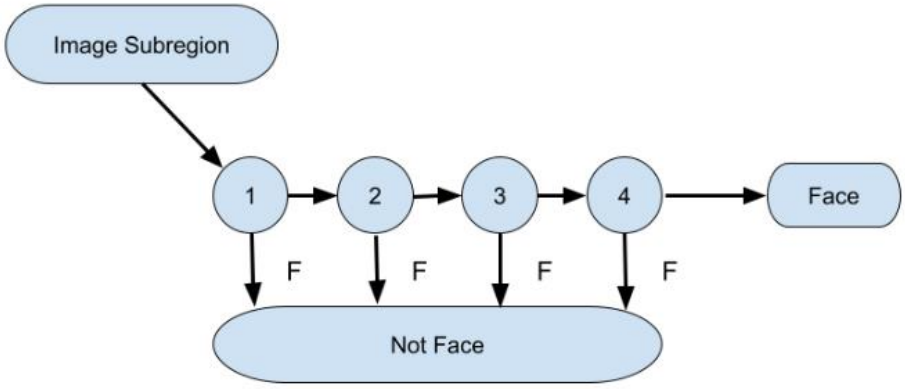
Boost алгоритмите са алгоритми които могат да комбинират няколко слаби класификатора (класифициращи функции) в един по силен класификатор. Като целта е с първите класификатори да се проверява за по-често срещани и по генерални признаци докато следващите класификатори проверяват по сложните признаци. Това позволява на детектиращят алгоритъм бързо да елиминира изображения в които не се виждат човешки лица.

AdaBoost или Adaptive Boosting е техника в машинното обучение. Дефинира се верига от класификатори, като всеки следващ класификатор се тренира от грешките на предходният. Това дава възможност на следващите класификатори да се специализират към по сложните случаи. Примерно за да се построи подобна верига, първият класификатор (примерно SVM) се тренира спрямо някакво множество от входни записи (данни). Към всеки запис се прибавя тегловен коефициент който трябва да е между 0 и 1. В началото всички записи започват с еднакви коефициенти. След обучението на първият класификатор коефициентите на грешно класифицираните записи се увеличават а коефициентите на правилно класифицираните записи се намаляват. При обучението на вторият класификатор от веригата, записите с по висок коефициент се взимат с приоритет и биха могли да се повторят по време на тренировката. Това се повтаря за всички класификатори във веригата. Броят на класификаторите в една верига е променлива величина но в повечето случаи е близък до броят на признаците в записа или изображението. Недостатъкът на този подход е че не би могъл да се тренира паралелно, което значително увеличава времето за обучение. След обучението за всеки класификатор се изчислява тегловен коефициент базиран на неговата точност. Това позволява да се елиминират класификатори следящи признаци които не носят информация за класификатора, тъй като те ще имат твърде малка точност.



Фиг. 15 – схема на трениране на класификатори с AdaBoost

2.1.4 Каскаден филтър



Фиг. 16 – Каскаден филтър

Това практически е полученият класификатор след прилагане на AdaBoost подхода за трениране. На Фиг. 16 е показана схемата на работа на подобен класификатор. Номерираните кръгове представляват индивидуалните класификатори във веригата На входа на веригата се подава регион (прозорец) от изображението той се проверява от първият филтър. Ако филтъра не открие признака за който е обучен в региона той го прескача и минава към следващият. Ако региона успее да мине през всички филтри успешно то във този регион е изобразен търсеният обект, в нашият случай човешко лице.

2.2 Разпознаване на лице

При разпознаването на лице ще се използва FaceNet моделът. Този модел се използва в системи за разпознаване, верификация и групиране на човешки лица. За целта се използва дълбока конволиционна мрежа която е специално тренирана да превръща подадените изображения във вектор, който представя признаците на изображението. Това позволява по прости алгоритми за класификация и регресия ефективно да работят с изображения посредством техните вектори.

Глава 3: Анализ на получените резултати, приложимост и изводи

3.1 Анализ на получените резултати

3.1.1 Детектор на изображение

В този пример ще сравним разгледаният детектор на Виола Джонс с по съвременен модел предложен Кайпен Занг и имплементиран с Keras. За целта се използват изображения на известни личности. Те се зареждат чрез OpenCV и се изпращат към съответният функцията за проверка показан на Фиг. 17. На функцията за проверка се подава изображението и неговата задача е да извика имплементацията на алгоритъма и да върне 1 ако то го е познало правилно или 0 ако не е.

```
def calculateAccuracy(predicate):
    def checkImage(path):
        img = cv2.imread(path)
        return predicate(img)
    base = 'dataset/train'; checks = []; images = []
    for folder in os.listdir(base):
        src = path.join(base, folder)
        if path.isdir(src):
            t=[path.join(src,i) for i in os.listdir(src)]
            images = images + t
    start = time.time()
    for i in range(len(images)):
        image: str = images[i]
        disp = image.replace("\\", "/")
        result = checkImage(image)
        print(f"({i+1}/{len(images)}) [Answer: {result}] {disp}")
        checks.append(result)
    end = time.time()
    return (sum(checks) / len(checks)) * 100, end - start
```

Фиг. 17 – функция за проверка на работата на детектиращ метод

По време на обработката се засича и времето за работа на алгоритъма. След обхождането на всички изображения, борят на правилно разпознатите изображения се разделя на броят на всички изображения и резултата се връща като проценти.

```
def haarMethod():
    detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    def check(img):
        grayScale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        boxes = detector.detectMultiScale(grayScale, 1.1, 4)
        return 1 if len(boxes) > 0 else 0
```

дите за

Фиг. 18 – метод на Виола Джонс

```
    detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    def check(img):
        grayScale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        boxes = detector.detectMultiScale(grayScale, 1.1, 4)
        return 1 if len(boxes) > 0 else 0
```

Фиг. 19 – метод на Кайпен Занг

Начините на извикване на имплментациите на двата метода са показани на Фиг. 18 и Фиг. 19.

```
accuracyHaar, timeTakenHaar = calculateAccuracy(haarMethod())
accuracyMtcnn, timeTakenMtcnn = calculateAccuracy(mtcnnMethod())
print(f"Haar -> Accuraccy: {accuracyHaar:.2f}% Execution time: {timeTakenHaar:.3f} s")
print(f"Mtcnn -> Accuraccy: {accuracyMtcnn:.2f}% Execution time: {timeTakenMtcnn:.3f} s")
```

Фиг. 20 – движещ код

На Фиг. 20 е показано парчето код който извиква анализиращата функция по веднъж за двата метода и извежда резултатите

```
Haar -> Accuraccy: 91.4% Execution time: 1.359 s
Mtcnn -> Accuraccy: 97.8% Execution time: 25.255 s
```

Фиг. 21 - резултати

От резултатите на Фиг. 21 ясно се виждат предимствата и недостатъците на двата метода. Метода на Виола Джонс въпреки неточността му в сравнения с по-съвременният метод на Кайпен Занг. Се справя близо 10 пъти по бързо. Това е така понеже метода на Виола Джонс е създаден през 2000 година и целта му е била да се използва в реално време в

устройства с ниски изчислителни възможности. Като тогавашните мобилни телефони. От друга стана методът на Кайпен Занг, който е разработен през 2015, е предвиден да работи върху съвременни видео карти които имат далеч по значителни изчислителни ресурси.

3.1.2 Разпознаване на изображение

Тук ще дадем кратък пример за имплементация на алгоритъм за разпознаване на изображения на Python. Целта е да се обучи модел който да може да разпознава лицата на известни личности.

Всички изображения предварително са детектирани и изрязани с помощта на метода на Кайпен Занг. Тук са представени само основни точки от кода но пълната версия е предоставена в приложението.

Подходът за разпознаваме на изображение може да се раздели на 2 стъпки (3 ако включим и детекцията). В първата стъпка от изображението трябва да се извлекат признаци, в такъв вид в какъвто те могат да бъдат сравними. Това става с функцията на

```
model = keras.models.load_model(getModel())
print('Loaded Model')

def getEmbedding(image: np.ndarray):
    image = image.astype('float32')
    mean, std = image.mean(), image.std()
    image = (image - mean) / std
    samples = np.expand_dims(image, axis=0)
    embedding = model.predict(samples)
    return embedding[0]
```

Фиг. 22 – извличане на признаци

Фиг. 22. В началото се зарежда FaceNet моделът. Това се прави само един път при зареждане на програмата. След това функцията getEmbedding(image: np.ndarray) се извиква за всяко изображение. Като изображенията са представени с многомерен масив на NumPy (3rd party библиотека за Python предоставяща функции и обекти за работа с матрици както в MATLAB). Преди извличането на признаците обаче трябва да се направи статистическа обработка на пикселите в изображението. Това става като първо се изчисли Z стойността

на всеки пиксел в изображението. На следващият ред се вика np.expand_dims, която практически поставя изображението в масив. Това се прави понеже моделът е направен да приема масив от изображения и да връща масив от признаците на всяко едно от тях. Накрая се извиква FaceNet модела с model.predict която активира модела.

```
xTrain, yTrain, xTest, yTest = loadEmbeddingsDataset()

encoderIn = sklearn.preprocessing.Normalizer(norm='l2')
xTrain = encoderIn.transform(xTrain)
xTest = encoderIn.transform(xTest)

encoderOut = sklearn.preprocessing.LabelEncoder()
encoderOut.fit(yTrain)
yTrain = encoderOut.transform(yTrain)
yTest = encoderOut.transform(yTest)
```

Фиг. 23 – Обработка на входните признаци и на класовете

Понеже извличането на признаци отнема значително време получените признаци от getEmbedding(image: np.ndarray) са записани (кеширани) във файл за по лесно зареждане. Това става с loadEmbeddingsDataset().

След изпълнението на функцията се създават четири променливи xTrain, yTrain, xTest, yTest. xTrain представлява масив от признаците извлечени от изображенията, а yTrain представлява имената на известната личност която е на съответното изображение. Подобно е и с xTest, yTest, с разликата че тези ще се използват при проверка за точността на обученият модел, а xTrain, yTrain ще се използват при обучаването му.

На следващите редова се прави обработка на признаците и етикетите като това се изисква за самият обучителен алгоритъм. За входните признаци се изпълнява L2 нормализация, която е подобна на нормална векторна нормализация. Което практически представлява изчисляване на разстоянието от център на декартова координатна система до позицията на върха на вектора (питагорова теорема) и разделянето на всяка компонента на вектора с големината му. Това се прави с цел сумата от всички компоненти във вектора да е 1-ца, като по този начин алгоритъмът по лесно би могъл да ги сравнява и класифицира.

След това се прави и обработка на етикетите. Това е защото оригиналните етикети са символен низ (string) а моделът може да връща само числа.

```
model = sklearn.svm.SVC(kernel='linear', probability=True)
model.fit(xTrain, yTrain)

trainPred = model.predict(xTrain)
testPred = model.predict(xTest)

scoreTrain = sklearn.metrics.accuracy_score(yTrain, trainPred)
scoreTest = sklearn.metrics.accuracy_score(yTest, testPred)

print('Accuracy: train=%.3f, test=%.3f' % (scoreTrain*100, scoreTest*100))
```

Фиг. 24 – Модел на машина с поддържащи вектори

За самият модел се използва класификатор на машина с поддържащи вектори, използвайки 3rd party библиотеката scikit learn. Задаваме му стандартните параметри. След това се тренира с помощта на метода fit, като се използват трениращите данни като входни аргументи. Накрая моделът се използва (model.predict) и на базата на получените от него резултати и реалните, се прави съпоставка и се извежда оценка на точността на модела.

```
Accuracy: train=100.000%, test=100.000%
```

Фиг. 25 – резултати от работата на модела

На Фиг. 25 е представена оценката за точността на моделът. Това биха изглеждали не правдоподобни ако не се вземе в предвид че този модел е трениран и тестван само с общо 93 изображения и FaceNet моделът който използвахме е предвиден за използване от големи компании като Facebook и Netflix, които използват модели можещи правилно да класифицират значително по голям обем информация от показаният в този пример.

3.2 Приложимост

Засичането и разпознаването на изображения намира голямо приложение в съвременният свят. Примери за това може да са разпознаване на човешки лица през охранителни камери в реално време с цел да се намери издирван престъпник. Откриване на заболявания по чертите на лицето на човека.

FaceNet, алгоритъмът който беше представен в тази работа, е силно надграден и разработен от Google. С това те успяват да направят нов рекорд за точност достигайки до 99.63% (± 0.0009) използвайки популярният „Labeled Faces in the Wild“ сет от изображения. Тази функционалност е вградена във приложението Google Photos и му дава възможност да подрежда снимките на потребителя както и да отбелязва личностите които са на дадена снимка както и да позволява търсене чрез името на даден човек (в кои снимки той може да бъде видян).

3.3 Изводи

Разпознаването на лица, ни позволява да превърнем зрителната информация, която до скоро само ние хората можехме да възприемем и анализираме, в математически модел. Това дава възможност на съвременните технологии да „виждат“ света по подобен на нас начин и да правят анализи върху него. Благодарение на изчислителните възможности на съвременните машини ние можем да представяме все по сложни математически модели и с това да развиваме възможностите на компютърното зрение и изкуственият интелект. Въпреки че даже и сега, със сложни математически модели и големи изчислителни възможности, моделите за класификация и взимане на решения са ограничени. Пример за това може да бъде как човек може да засече не съответствия в дадено изображение без да има пряк опит, или да достигне до напълно различен извод. Докато изкуственият интелект в момента не може да изведе решение което не е програмирано в него (Например да изведе дума при положение че трябва да даде процентен резултат).

Анализите и примерите в тази работа показват възможностите на машинното обучение и на компютърното зрение и как то може да помогне в живота на хората.

Използвана литература

[1] T. Phillips, „Survey: Four in five consumers globally would use a biometric payment card,“ [Онлайн]. Available: https://www.nfcw.com/whats-new-in-payments/survey-four-in-five-consumers-globally-would-use-a-biometric-payment-card/.

[2] C. Gürel, „DEVELOPMENT OF A FACE RECOGNITION SYSTEM,“ [Онлайн]. Available: https://www.researchgate.net/publication/265026957_DEVELOPMENT_OF_A_FACE_RECOGNITION_SYSTEM.

[3] A. F. Abate, „2D and 3D Face Recognition: A Survey,“ [Онлайн]. Available: https://www.researchgate.net/publication/220646065_2D_and_3D_Face_Recognition_A_Survey.

[4] P. Kamencay, „2D-3D Face Recognition Method Basedon,“ [Онлайн]. Available: https://journals.sagepub.com/doi/pdf/10.5772/58251.

[5] „Eigenface,“ [Онлайн]. Available: https://en.wikipedia.org/wiki/Eigenface.

[6] S. Lau, „Image Augmentation for Deep Learning,“ [Онлайн]. Available: https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2.

[7] M. Heller, „What is Keras? The deep neural network API explained,“ [Онлайн]. Available: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html.

[8] S. Saha, „A Comprehensive Guide to Convolutional Neural Networks,“ [Онлайн]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[9] „What is an API?,“ [Онлайн]. Available: https://www.mulesoft.com/resources/api/what-is-an-api.

[10] H. I. B. Idder, „Fig 1,“ [Онлайн]. Available: https://www.researchgate.net/figure/The-contrast-enhancement-by-contrast-limited-adaptive-histogram-equalization-method_fig1_276920560.

[11] M. Tyagi, „Viola Jones Algorithm and Haar Cascade Classifier,“ [Онлайн]. Available: https://towardsdatascience.com/viola-jones-algorithm-and-haar-cascade-classifier-ee3bfb19f7d8.

[12] „Cascade Classifier OpenCV doc,“ [Онлайн]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[13] A. Desarda, „Understanding AdaBoost,“ [Онлайн]. Available: https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe.

[14] A. Mittal, „Haar Cascades, Explained,“ [Онлайн]. Available: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d.

[15] K. Zhang, „Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks,“ [Онлайн]. Available: https://arxiv.org/abs/1604.02878.

`

Приложение

 # Пример за детектиране на лице в реално време (с камера на лаптоп)

```
import cv2

detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

camera: cv2.VideoCapture = cv2.VideoCapture(0)

while True:

    _, img = camera.read()

    grayScale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    boxes = detector.detectMultiScale(grayScale, 1.1, 4)

    for (x, y, w, h) in boxes:
        cv2.rectangle(img, (x, y), (x+w, y+w), (0,0,255), thickness=3)

    cv2.imshow('Camera', img)

    k = cv2.waitKey(30) & 0xff
    if k==27:
        break

camera.release()
```

 # Пример за изграждане на модел за класификация на лица с помощта на FaceNet

```
# Model: https://drive.google.com/u/0/uc?id=1PZ_6Zsy1Vb0s0JmjEmVd8FS99zoMCiN1&export=download

MODEL_URL =
"https://drive.google.com/u/0/uc?id=1PZ_6Zsy1Vb0s0JmjEmVd8FS99zoMCiN1&export=download"

from re import M
import urllib.request as req
import urllib.response as resp
from http.client import HTTPMessage
import os
import os.path as path
import cv2
from keras.optimizers import get
import numpy as np
import mtcnn
import keras
import sklearn.svm
import sklearn.preprocessing
import sklearn.metrics

def getModel():
    cache = path.join('cache')
    os.makedirs(cache, exist_ok=True)
```

```
    model = path.join(cache, 'facenet_keras.h5')
    if path.exists(model):
        return model

    with req.urlopen(MODEL_URL) as u:
        meta: HTTPMessage = u.info()
        size = int(meta["Content-Length"])
        print(f"Downloading facenet_keras.h5 Bytes: {size}")

        with open(model, 'wb') as f:
            bufferSize = 8192
            total = size
            downloaded = 0

            while True:

                buffer = u.read(bufferSize)
                if not buffer:
                    break

                f.write(buffer)
                downloaded += len(buffer)
                print(f"Downloading: {(downloaded * 1.0) / total * 100.0}%")
    return model

# from keras.models import load_model
# print(getModel())
# model = load_model('cache\\facenet_keras.h5')
# print(model.inputs)
# print(model.outputs)

detector = mtcnn.MTCNN()

def loadImage(path, size=(160,160)):
    img = cv2.imread(path)
    result = detector.detect_faces(img)
    if not result:
        return None
    x, y, w, h = result[0]['box']
    #cv2.rectangle(img, (x, y), (x+w, y+h), (0,0,255), thickness=3)
    x1 = abs(x)
    y1 = abs(y)
    x2 = x1 + w
    y2 = y1 + h

    img = img[y1:y2, x1:x2]
    img = cv2.resize(img, size)

    return img

# for pth in os.listdir(base):
#     fld = path.join(base, pth)
```

```
#     print(fld)
#     images = [loadImage(path.join(fld, f)) for f in os.listdir(fld)]
#     full = np.concatenate(images, axis=1)
#     cv2.imshow(pth,full)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()

def loadFromFolder(base):
    images = []
    for f in os.listdir(base):
        pth = path.join(base, f)
        img = loadImage(pth)
        if img is not None:
            images.append(img)
    return images

def loadDataset(base):
    x = []
    y = []

    for f in os.listdir(base):
        pth = path.join(base, f)
        if not path.isdir(pth):
            continue
        faces = loadFromFolder(pth)
        for face in faces:
            x.append(face)
            y.append(f)
        print(f"Loaded {len(faces)} samples for class {f}")
    return np.asarray(x), np.asarray(y)

DATASET = 'dataset/dataset.npz'
def cacheDataset():
    xTrain, yTrain = loadDataset('dataset/train')
    xTest, yTest = loadDataset('dataset/val')

    np.savez_compressed(DATASET, xTrain, yTrain, xTest, yTest)

def loadDataset():
    if not path.exists(DATASET):
        cacheDataset()

    data = np.load(DATASET)
    return data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']


EMBEDDINGS_DATASET = 'dataset/embeddings-dataset.npz'

def cacheEmbeddingsDataset():
    xTrain, yTrain, xTest, yTest = loadDataset()
    model = keras.models.load_model(getModel())
    print('Loaded Model')
```

```
    def getEmbedding(image: np.ndarray):
        image = image.astype('float32')
        mean, std = image.mean(), image.std()
        image = (image - mean) / std
        samples = np.expand_dims(image, axis=0)
        embedding = model.predict(samples)
        return embedding[0]

    def getEmbeddings(data: np.ndarray):
        return np.asarray([getEmbedding(point) for point in data])

    xTrainEmbeddings = getEmbeddings(xTrain)
    print(xTrainEmbeddings.shape)

    xTestEmbeddings = getEmbeddings(xTest)
    print(xTestEmbeddings.shape)

    np.savez_compressed(EMBEDDINGS_DATASET, xTrainEmbeddings, yTrain, xTestEmbeddings, yTest)

def loadEmbeddingsDataset():
    if not path.exists(EMBEDDINGS_DATASET):
        cacheEmbeddingsDataset()

    data = np.load(EMBEDDINGS_DATASET)
    return data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']

xTrain, yTrain, xTest, yTest = loadEmbeddingsDataset()

print(f'Dataset train: {xTrain.shape} test: {xTest.shape}')

encoderIn = sklearn.preprocessing.Normalizer(norm='l2')
xTrain = encoderIn.transform(xTrain)
xTest = encoderIn.transform(xTest)

encoderOut = sklearn.preprocessing.LabelEncoder()
encoderOut.fit(yTrain)
yTrain = encoderOut.transform(yTrain)
yTest = encoderOut.transform(yTest)

model = sklearn.svm.SVC(kernel='linear', probability=True)
model.fit(xTrain, yTrain)

trainPred = model.predict(xTrain)
testPred = model.predict(xTest)

scoreTrain = sklearn.metrics.accuracy_score(yTrain, trainPred)
scoreTest = sklearn.metrics.accuracy_score(yTest, testPred)

print('Accuracy: train=%.3f, test=%.3f' % (scoreTrain*100, scoreTest*100))
```

 # Сравнение на детектиращи алгоритми

```
import cv2
import os
import os.path as path
import time

import mtcnn

def calculateAccuracy(predicate):
    def checkImage(path):
        img = cv2.imread(path)
        return predicate(img)
    base = 'dataset/train'; checks = []; images = []
    for folder in os.listdir(base):
        src = path.join(base, folder)
        if path.isdir(src):
            images = images + [path.join(src, image) for image in os.listdir(src)]
    start = time.time()
    for i in range(len(images)):
        image: str = images[i]
        disp = image.replace("\\", "/")
        result = checkImage(image)
        print(f"({i+1}/{len(images)}) [Answer: {result}] {disp}")
        checks.append(result)
    end = time.time()
    return (sum(checks) / len(checks)) * 100, end - start


def haarMethod():
    detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    def check(img):
        grayScale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        boxes = detector.detectMultiScale(grayScale, 1.1, 4)
        return 1 if len(boxes) > 0 else 0
    return check


def mtcnnMethod():
    detector = mtcnn.MTCNN()
    def check(img):
        result = detector.detect_faces(img)
        return 1 if len(result) > 0 else 0

    return check


accuracyHaar, timeTakenHaar = calculateAccuracy(haarMethod())
accuracyMtcnn, timeTakenMtcnn = calculateAccuracy(mtcnnMethod())
print(f"Haar -> Accuracy: {accuracyHaar:.1f}% Execution time: {timeTakenHaar:.3f} s")
print(f"Mtcnn -> Accuracy: {accuracyMtcnn:.1f}% Execution time: {timeTakenMtcnn:.3f} s")
```