УНИВЕРЗИТЕТ,,ГОЦЕ ДЕЛЧЕВ” - ШТИП

ФАКУЛТЕТ ЗА ИНФОРМАТИКА

Катедра по компјутерско инженерство и комуникациски технологии

Штип

ДАРКО ЗЛАТЕВ

ДИГИТАЛНА ОБРАБОТКА НА СЛИКИ СО КОРИСТЕЊЕ НА МЕТОДИ БАЗИРАНИ НА ДЛАБОКО УЧЕЊЕ

-МАГИСТЕРСКИ ТРУД-

Штип, октомври 2018

УНИВЕРЗИТЕТ,,ГОЦЕ ДЕЛЧЕВ” - ШТИП

ФАКУЛТЕТ ЗА ИНФОРМАТИКА

Катедра по компјутерско инженерство и комуникациски технологии

Штип

ДАРКО ЗЛАТЕВ

ДИГИТАЛНА ОБРАБОТКА НА СЛИКИ СО КОРИСТЕЊЕ НА МЕТОДИ БАЗИРАНИ НА ДЛАБОКО УЧЕЊЕ

-МАГИСТЕРСКИ ТРУД-

Штип, октомври 2018

Комисија за оценка и одбрана:

Ментор: Вон. проф. д-р Игор Стојановиќ,
Факултет за информатика-
Универзитет „Гоце Делчев”

Член: проф. д-р Цвета Мартиновска Банде,
Факултет за информатика-
Универзитет „Гоце Делчев”

Член: Вон. проф. д-р Александар Крстев
Факултет за информатика-
Универзитет „Гоце Делчев”

Членови на комисија за оценка и одбрана:

Претседател: проф. д-р Цвета Мартиновска Банде,
Факултет за информатика-
Универзитет „Гоце Делчев”

Член: Вон. проф. д-р Игор Стојановиќ,
Факултет за информатика-
Универзитет „Гоце Делчев”

Член: Вон. проф. д-р Александар Крстев
Факултет за информатика-
Универзитет „Гоце Делчев”

Научно поле: Информатика

Научна област: Теориска информатика

Датум на одбрана: 24.12.2018

Листа на рецензирани и објавени трудови:

Krstev, Aleksandar and Beqiri, Lavdim and Krstev, Boris and Krstev, Dejan and Zlatev, Darko and Donev, Aleksandar (2018) *Application of Machine Learning in Software Engineering.* In: IX International Conference of Information Technology and Development of Education 2018 (ITRO 2018), 29 June 2018, Zrenjanin, Republic of Serbia.

Краток извадок

Сликите се дел нашето секојдневие, ги користиме да зачуваме некој момент или да запаметиме некој лик. Тие се покажале како најголем доказ во многу ситуации и сè уште се сметаат за еден од најрелевантните извори на податоци.

Во овој труд, истражувани се методите на длабоко учење кои се покажале како исклучително добри во класификацијата на слики. Разработени се нивните алгоритми и навлезено е во нивната архитектура и начин на работа. Сликите ги донесов до ниво на пиксели и објаснив како методите го користат секој пиксел за да откријат што се случува, што е претставено и како може тоа да се класифицира.

Посветено е особено внимание на човековото лице и изработена е веб апликација која може да детектира лице или лица на слика и како втора функционалност може да го препознае ликот од сликата споредувајќи го со слики од претходно направена база на слики. За потребите на апликацијата истражувана и искористена е библиотеката OpenCV која работи на методи на длабоко учење и е усовршена да работи со слики, да препознава објекти, лица, делови од тело, одредени ситуации и др. Детално се објаснети и применети алгоритмите на OpenCV, како тие работат, врз основа на што ги донесуваат нивните заклучоци и колку се точни во нивната работа.

Библиотеката OpenCV во апликацијата е имплементирана преку програмскиот јазик Python и работната околина Django. Работи на принцип прикачување на слика, а потоа одбирање која функционалност да се изврши врз сликата, дали детекција или препознавање на ликот. Прикажани се ситуации кога апликацијата т.е. алгоритмите се безгрешни, а и ситуации во кои тие не детектираат или не препознаваат добро.

Клучни зборови: *длабоки архитектури, невронски мрежи, нивоа, каскади, белези*

Abstract

Images are part of our everyday life, we use them to save a special moment or to remember a person. Images shown to be the biggest prove in many situations and they are still considered as one of the strongest data sources.

In this study, I made researches about deep learning methods which shown to be exclusively good in image classification. I elaborated their algorithms, dug in their architecture and their principles of work. I brought images to their pixel level and explained how methods use every pixel in order to learn what is going on in the image or what it represents and how can that be classified.

The attention goes on human faces and I made a web application which can detect face or faces on image, and as second functionality it can recognize the person on the image compared with persons from previously build in database of images. For the application's use, I made researches on the OpenCV library which uses deep learning methods and it has been sophisticated to work with images, recognize objects, faces, body parts, scenes and so on. I have explained OpenCV's algorithms and their work, how they make decisions and how precisely are they in their classification.

The OpenCV library has been implemented in my application with the program language Python and Django framework. The work flow starts with uploading an image and then choosing the functionality we want to do, face detection or face recognition. Shown are situations when my application gives flawless results, and situations where faces are not so good detected or recognized.

Key words: *deep architectures, neural networks, levels, cascades, features*

СОДРЖИНА

1. ВОВЕД

Не може, а да не помине ден без околу нас да забележиме како општеството се менува под влијание на новите технологии што дури веќе сме имуни на големото прогресивно, технолошко движење и едноставно го продолжуваме денот во кој неколку пати ќе посакаме да го вклучиме паметниот телефон и тоа ќе го направиме само со негово подигање кон нашето лице, или ќе го погледнеме со нашите очи. Но не секогаш било така.

Зошто денес тоа е возможно и зошто ние ги уживаме тие удобности, е резултат на долгогодишни истражувања, работење, експерти и труд кој се исплатил и овозможил да се развие нова гранка на вештачката интелигенција наречена длабоко учење.

Ако некогаш сликите вределе илјада зборови, денес тие вредат милион. Благодарение на методите на длабоко учење, неверојатно е колку информации можат да се извлечат од една слика и колку значење може да има во еден пиксел. Постојат многу начини и алгоритми кои се направени за таа цел и можат успешно да ги извадат најбитните белези од една слика за да покажат што се “крие” на неа.

Факт е дека за да нешто се дознае детално, мора да се “копа” длабоко затоа длабоките архитектури ја истражуваат сликата до ниво на пиксел и движејќи се прозорец по прозорец ги вадат потребните белези од неа и му укажуваат на алгоритмот каде е тоа што се бара.

2. ЦЕЛ НА ИСТРАЖУВАЊЕТО

Целта и инспирацијата за пишување на овој магистерски труд е да се истражат и осознаат работните техники со кои сликите се класифицираат и се дознава што или кој има на нив. Во моето истражување ќе бидат опфатени повеќе методи на длабоко учење и невронски мрежи меѓу кои: рекурентните невронски мрежи (RNN), долги краткотрајни мемории (LSTM), длабоки невронски мрежи на доверба (DBN), длабоки "натрупувачки" мрежи (DSNS) и особено конволуциските невронски мрежи (CNN). Намерата е да се допре длабоко во длабоките архитектури за да се разбере нивната структура и да се разбере целиот процес низ кој поминува една слика за да на крај успешно се даде нејзината класификација или опис.

Во рамките на практичниот дел ќе биде разработена библиотеката OpenCV која работи на методи на длабоко учење и истата ќе ја употребам во изработката на веб апликација која ќе може да детектира лице од прикачена слика и да го препознае лицето од сликата. Подетално во овој дел ќе бидат опишани некои од алгоритмите кои оваа библиотека ги користи за да ги извади белезите од сликата и да го осознае нејзиното значење. Целта во практичниот дел ќе биде да се тестираат алгоритмите за длабоко учење што ги користи оваа библиотека, да ги видиме јаките и слабите страни на алгоритмите, а од друга страна позитивните и негативните елементи од сликата кои се фактор во секое добивање на резултатот.

3. МЕТОДИ НА ИСТРАЖУВАЧКАТА РАБОТА

3.1. Што е длабоко учење?

Длабоко учење претставува вид на машинско учење кое вклучува алгоритми инспирирани од структурата и функционалноста на човековиот мозок. Моделот на длабоко учење е направен така да може “природно” да учи од примери и да извршува задачи на класификација директно од слики, текст или звук. Длабоко учење е клучната технологија која стои позади автомобилите без возач, контролата на звук во телефоните, таблетите, телевизијата и сл. Моделите на длабоко учење можат да достигнат ниво на точност поголема дури и од човечките сетила.

Зборот „длабоко” во длабоко учење посочува на бројот на нивоа преку кои се трансформираат влезните податоци, се научуваат обележјата, се намалуваат грешките итн.

3.2. Историја на длабокото учење

За татко на длабокото учење се смета Алексеј Иваченко кој е познат Украински математичар и прв го развил методот за индуктивно статистичко учење.

Од него бил напишан првиот функционален алгоритам за надгледувано учење со користење на повеќе нивоа во далечната 1965 година.

Други архитектури за длабоко учење, конкретно оние изградени за компјутерската визија биле претставени петнаесет години подоцна во 1980 година од страна на Кунихико Фукушима, а дури 1986-та година, за прв пат бил спомнат поимот “длабоко учење” пред заедницата вклучена во развојот на машинското учење.

Во 1991-та година почнале да се користат системи за препознавање 2-D рачно напишани цифри, додека препознавањето на 3-D објекти се правело со спарување на 2-D слики со рачно направен 3-D објект. Во 1992-та година бил

објавен методот **Cresceptron**[1] за вршење на 3-D препознавање на објекти во "натрупани" сцени. Овој метод бил способен да учи обележја низ нивоата без надгледување. Во 1994-та година од страна на Андре де Карваљо биле објавени експериментални резултати од ***boolean*** невронска мрежа на повеќе нивоа, исто така позната како бестежинска невронска мрежа, која била составена од невронска мрежа за вадење белези и ниво со модул за класифицирање кои биле тренирани независно едни од други.

Во периодот од 1990 до 2000 година се попопуларни биле поедноставните модели бидејќи цената на компјутерската моќ била доста скапа, а имало и недостаток од познавање на работата на мрежите на човековиот мозок.

Најголемиот подем на длабокото учење започнал во раните 2000 години, сепак биле потребни дури десет години да во 2010-та година за прв пат се искористи длабоко учење во препознавањето на говор. Во 2009-та година Nvidia[2] се вклучила во развојот на длабокото учење вклучувајќи ги нејзините графичко процесирачки единици што навистина го подобрило неговиот развој.

Сепак револуцијата на длабокото учење започнала во 2012 година кога точноста и моќноста на длабокото учење значително помогнала во препознавањето на слики и објекти. Од тогаш методите на длабоко учење победувале на речиси сите натпревари за препознавање на шаблони, сегментација на слики, препознавање на објекти, компјутерска визија, анализа на медицински слики, препознавање на говор итн.

[1] добиено од латинскиот збор *cresco* (расте) и *perception* (перцепција) претставува работна околина која користи повеќе нивоа на неврони

[2] Американска компанија која постои од 1993 година и изработува графички процесорски единици

3.3. Зошто е битно длабокото учење и каде се користи

Длабокото учење овозможува компјутерски модели кои се составени од повеќе слоеви за обработка да можат да научат репрезентации на податоци со повеќе нивоа на апстракција. Овие методи драматично ја подобрија состојбата во препознавањето на говор, препознавање на визуелен објект, откривање на предмети и многу други домени како откривање на дрога и геномика. Длабокото учење открива сложена структура во големи множества на податоци со користење на **backpropagation**[3] алгоритам за да покаже како машината треба да ги промени своите внатрешни параметри кои се користат за да се пресмета претставата во секој слој од претставата во претходниот слој. Длабоките конвертолошки мрежи доведоа до откритија во обработка на слики, видео, говор и аудио, додека рекурентните мрежи ги осветлиле секвенцијалните податоци, како што се текстот и говорот.

Технологијата за **машинско учење** засилува многу аспекти на современото општество: од веб-пребарувања до филтрирање содржини на социјалните мрежи до препораки на веб-страниците за е-трговија и тоа е сè повеќе присутно во потрошувачките производи, како што се камери и паметни телефони. Системите за машинско учење се користат за идентификација на објекти во слики, преведување на говор во текст, совпаѓање со вести, мислења или производи со интереси на корисниците и избирање на релевантни резултати од пребарувањето. Покрај тоа, овие апликации ја користат класата на техники наречени длабоко учење.

Конвенционалните техники за машинско учење беа ограничени во нивната способност да обработуваат природни податоци во нивната сурова форма. Со децении, изградбата на систем за препознавање на моделот или за машинско учење бараше внимателно инженерство и значителна експертиза за доменот за да дизајнира екстрактор на карактеристики кој ги трансформира суровите податоци (како што се вредностите на пиксели на сликата) во соодветна внатрешна

[3] метод кој се користи во вештачката интелигенција за пресметување на градиентот потребен за пресметка на тежините во мрежата

репрезентација или функција вектор од кои систем на учење, често класификатор, може да открие или класифицира модели во влез.

Учењето е збир на методи кои овозможуваат машината да се храни со сурови податоци и автоматски да ги открие претставите потребни за откривање или класификација. Методите за длабоко учење се методи на репрезентативно учење со повеќе нивоа на застапеност, добиени со компонирање на едноставни, но нелинеарни модули кои секој од нив ја трансформира претставата на едно ниво (почнувајќи од суровиот влез) во претстава на повисоко апстрактно ниво. Со составот на доволно такви трансформации може да се научат многу комплексни функции. За задачите на класификација, повисоките слоеви на застапеност ги засилуваат аспектите на влезот кои се важни за дискриминација и ги потиснуваат ирелевантните варијации. Сликата, на пример, доаѓа во форма на низа вредности на пиксел, а научените функции во првиот слој на репрезентација обично го претставуваат присуството или отсуството на рабови во одредени ориентации и локации во сликата. Вториот слој обично ги детектира мотивите со откривање на одредени аранжмани на рабовите, без разлика на мали варијации во работните позиции. Третиот слој може да собере мотиви во поголеми комбинации кои одговараат на делови од познати објекти, а подоцнежните слоеви ќе ги детектираат предметите како комбинации на овие делови. Клучниот аспект на длабокото учење е дека овие слоеви на карактеристики не се дизајнирани од човечки инженери туку тие се научени од податоци користејќи општа намена за учење.

Длабокото учење прави голем напредок во решавањето на проблемите кои се спротивставија на најдобрите обиди на вештачката интелигенција низ многу години. Се покажа дека е многу добро во откривањето на сложени структури во податоци со висока димензија и затоа е прикладно за многу домени на науката, бизнисот и владата. Во предност на добивање на записи во препознавањето на сликите и препознавањето на говор, ги победија другите машински техники за учење при предвидување на активноста на потенцијалните лековити молекули, анализата на податоци за забрзувач на честички, реконструкција на мозочните кола и предвидување на ефектите на мутации во некодирачка ДНК на изразување на ген и болест. Можеби повеќе изненадувачки, длабокото учење создаде крајно

ветувачки резултати за различни задачи во разбирањето на природниот јазик, особено класификација на теми, анализа на чувства, одговарање на прашања и превод на јазик.

Сметаме дека длабокото учење ќе има многу повеќе успеси во блиска иднина, бидејќи бара многу малку инженеринг од рака, така што лесно може да ги искористи предностите на зголемување на количината на расположливи пресметувања и податоци. Новите алгоритми за учење и архитектури кои во моментов се развиваат за длабоки невронски мрежи само ќе го забрзаат овој напредок.

3.4. Поврзаноста на машинското учење со длабокото учење

Во практична смисла, длабокото учење претставува под множество од машинското учење. Технички тоа и е машинско учење кое функционира на многу сличен начин, но има поразлични можности.

Основните модели на машинско учење успеваат да бидат прогресивно многу подобри во секоја нивна функционалност, но сепак мора да бидат предводени од нешто. Ако алгоритмот на машинско учење се случи да предвиди непрецизни резултати, тогаш инженерот мора да пристапи и да направи некои поправки и приспособувања. Но во случајот на длабоко учење, методот сам може да детерминира дали неговата предикција е прецизна или не.

Машинското учење користи алгоритми за да ги парсира податоците, да научи од нив и да прави одлуки базирани на тоа што го има научено. Длабокото учење ги структурира алгоритмите во нивоа за да создаде “вештачка невронска мрежа” која е способна да учи и сама да прави интелигентни одлуки.

Длабокото учење сепак е под множество на машинското учење, додека и двете се во состав на вештачката интелигенција. Длабокото учење е она што најмногу се доближува до човечката моќ во вештачката интелигенција.

3.5. Длабоки архитектури

Вештачките невронски мрежи се структурно и концептуално инспирирани од човековиот биолошки нервен систем. Прецептрон е една од најраните невронски мрежи што е базирана на човечкиот мозок. За да можат да решат покомплексни проблеми, невронските мрежи биле изградени од многу нивоа каде што имаме влезно ниво, ниво за излез и повеќе скриени нивоа помеѓу нив. Невронската мрежа се состои од внатрешно поврзани неврони кои ги преземаат влезните податоци и вршат процеси над нив, а понатаму ги препуштаат кон наредното ниво. Една длабока невронска мрежа денес може да содржи повеќе од илјада нивоа, со таков капацитет тие се способни успешно да меморираат многу шаблони од тренирањето.

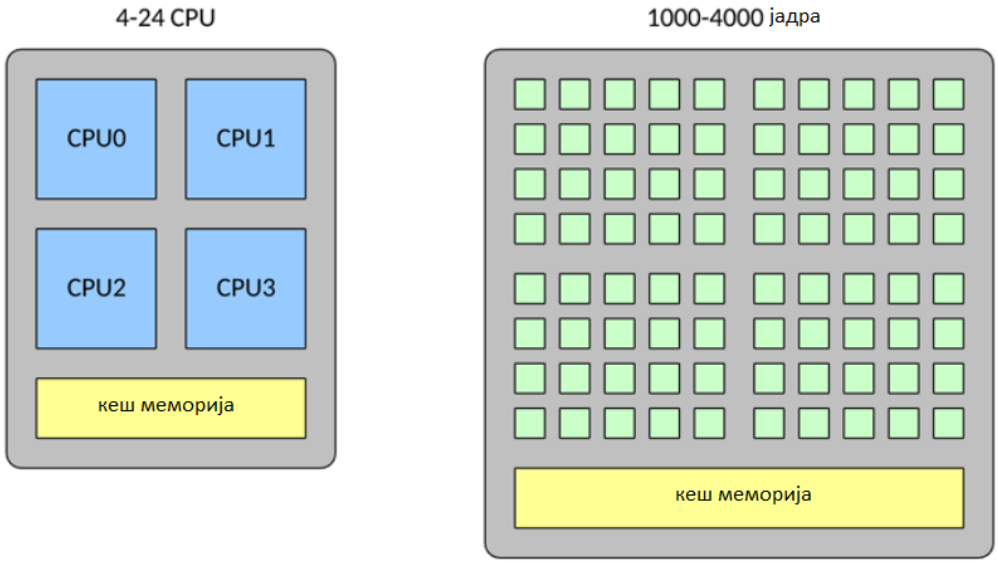
Поврзаните архитектури постојат повеќе од седумдесет години, но новите архитектури заедно со графичко процесирачките единици се она што ги доведе до водечки дел од вештачката интелигенција. Последните две декади ни ги донесоа длабоките архитектури, кои значително го проширија спектарот на проблеми кои можат да ги решат. Постојат пет најпознати архитектури на длабокото учење и тоа се: рекурентни невронски мрежи (RNN), долги краткотрајни мемории (LSTM), конволуциски невронски мрежи (CNN), длабоки невронски мрежи на доверба(DBN), и длабоки натрупувачки мрежи (DSNS).

3.5.1. Длабокото учење и растот на GPU

Невронските мрежи постојат помеѓу нас повеќе години, но развивањето на повеќебројни нивоа на мрежи (каде секоја врши некоја функција) ги направи да бидат многу попрактични за употреба. Додавањето нивоа значи повеќе меѓусебни врски и тежини помеѓу нивоата и во самото ниво. За таа цел најмногу помогнале GPU уредите кои направиле да биде возможно тренирањето и функционирањето на овие длабоки мрежи.

GPU се разликуваат од вообичаените повеќе јадрени процесори во повеќе клучни точки. Најпрво, традиционалниот процесор може да содржи од четири до

дваесет и четири јадра, но GPU може да содржи дури од илјада до четири илјади специјализирани јадра за процесирање. Високата бројка на јадра го прави GPU да биде високо паралелен што значи да може да извршува повеќе пресметки одеднаш. Ова го прави GPU идеален за големи невронски мрежи и овозможува да се пресметаат голем број на неврони одеднаш.

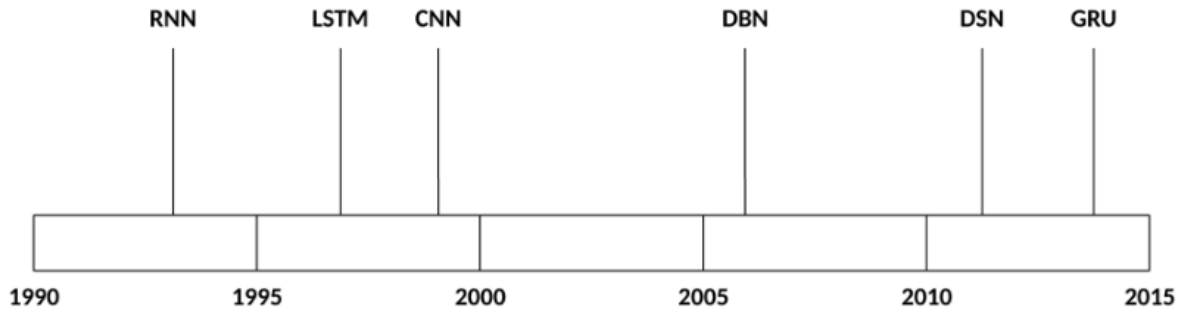


Слика 1: Споредба на CPU со GPU

Figure 1: Comparation of CPU and GPU

3.5.2. Архитектури

Бројот на архитектури и алгоритми што се користат за длабоко учење е голем и разнообразен. Тие се појавувале со текот на времето во различни периоди.



Слика 2: Појава на архитектурите за длабоко учење низ годините

Figure 2: Deep learning architectures through past years

Сите овие методи се употребуваат во голем спектар на сценарија, но нивната најголема функционалност ќе ја претставиме во следната табела:

Архитектура/метод	Употреба
RNN	Препознавање на говор, препознавање на рачно напишан текст
LSTM	Препознавање на природен говор, рачно пишување, препознавање на мимики
CNN	Препознавање на слики, анализа на видеа, процесирање на природен говор
DBN	Препознавање на слики, разбирање на природен говор, предикција на грешки
DSN	Препознавање на говор во континуирано говорење

Табела 1: Архитектури на длабоко учење и нивната употреба

Table 1: Deep learning architectures and their use

3.6. Видови на длабоко учење и нивна употреба

Пред било какво класифицирање и поделба на методите за длабоко учење, мораме да ги запазиме основните постапки и корените од каде што потекнуваат сите посебни архитектури. За таа цел ќе направиме една ретроспектива кон надгледуваното учење и **backpropagation** учењето, а понатаму ќе ги разгледаме повеќето видови на невронски мрежи кои спаѓаат во групата методи за длабоко учење. Можеме да кажеме дека секој метод е различен сам по себе и најдобар во некоја функционалност која ја врши. Затоа низ следните неколку страни подетално ќе ги обопштиме и разгледаме различните методи, нивната архитектура, нивните слаби и јаки страни и нивната најголема употреба.

3.6.1. Надгледувано учење

Најчестата форма на машинско учење, длабоко или не, е надгледувано учење. Замислете дека сакаме да изградиме систем кој може да ги класифицира сликите според нивната содржина, да речеме, куќа, автомобил, личност или миленичe. Ние прво собираме голем збир на податоци од куќи, автомобили, луѓе и домашни миленици, секој од нив е означен со неговата категорија. За време на обуката, на машината и се прикажува слика и таа произведува излез во форма на вектор од резултати, по еден за секоја категорија. Ние сакаме посакуваната категорија да има највисок резултат од сите категории, но ова не е веројатно да се случи пред обуката. Треба да поставиме објективна функција која ја мери грешката (или дистанцата) помеѓу излезните резултати и посакуваната шема на резултати. Машината потоа ги менува своите внатрешни прилагодливи параметри за да ја намали оваа грешка. Овие прилагодливи параметри, честопати наречени тежини, се реални броеви кои може да се гледаат како “јазли” кои ја дефинираат функцијата за влез-излез на машината. Во типичен систем за длабоко учење, може да има стотици милиони од овие прилагодливи тежини, и стотици милиони означени примери со кои ќе се обучи машината.

За правилно прилагодување на векторот на тежината, алгоритмот за учење составува градиент вектор кој за секоја тежина укажува на тоа колку грешката ќе се зголеми или намали, кога тежината би била зголемена за мала количина. Тежинскиот вектор потоа се прилагодува во спротивен правец кон векторот на градиент. Објективната функција е добиена просечно од сите претходни тренирања и може да се гледа како еден вид ридски пејзаж во високодимензионалниот простор на тежински вредности. Негативниот градиент вектор го покажува правецот на најстрмните падови во овој предел, земајќи го поблиску до минимум, каде што излезната грешка е во просек ниска.

Во пракса, повеќето практиканти користат процедура наречена стохастичко градиентско спуштање (Stochastic Gradient Descent) - (SGD). Ова се состои од прикажување на влезниот вектор за неколку примери, пресметување на излезите и грешките, пресметување на просечниот градиент за тие примери и прилагодување на тежините соодветно. Процесот се повторува за многу мали множества на примери од обуката се додека не се намали просекот на објективната функција. Се нарекува стохастички, бидејќи секој мал сет на примери дава бучна проценка на просечниот градиент врз сите примери. Оваа едноставна процедура најчесто наоѓа добар износ на тежини изненадувачки брзо кога се споредува со многу поразвиени техники за оптимизација. По обуката, перформансите на системот се мерат на друг сет на примери наречени тест сет. Ова служи за тестирање на способноста за генерализација на машината, нејзината способност да произведе разумни одговори за нови влезови кои никогаш не ги видела за време на обуката.

Многу од денешните практични апликации за машинско учење користат линеарни класификатори. Линеарниот класификатор пресметува збирна сума од компонентите на векторот. Ако тежинската т.е. збирната сума е над прагот, влезот е класифициран дека припаѓа на одредена категорија. Од 1960-тите години знаеме дека линеарните класификатори можат само да го издвојуваат нивниот влезен простор во поедноставни региони. Но, проблеми како што се слика и препознавање на говор бара функцијата за влез-излез да биде нечувствителна на ирелевантните варијации на влезот, како што се варијации во позиција, ориентација или осветлување на предмет, или варијации во теренот или акцент на говор, додека се

многу чувствителни на одредени минливи варијации. На пример да ја разгледаме разликата помеѓу бел волк и бела боја на куче од расата Самојед, на ниво на пиксели, слики од двајца Самоједи во различни појави и во различни средини може да бидат многу различни едни од други, додека две слики на Самојед и волк во иста позиција и на слични локации може да бидат многу слични една со друга.

3.6.2. Backpropagation

Од најраните денови на препознавање на шаблоните целта на истражувачите е да ги заменат рачно добиените карактеристики со повеќеслојни мрежи кои сами би се тренирале, но и покрај неговата едноставност, решението не било широко разбрано до средината на 1980-тите години. Повеќеслојната архитектурата може да се обучи со едноставно стохастичко градиентско спуштање. Се додека модулите се релативно мазни, едноставните функции на нивните влезови и нивните внатрешни тежини, може да пресметаат градиенти со користење на постапка за задна пропагација (backpropagation). Идејата дека ова може да се направи и дека работи, било откриено независно од неколку различни групи во текот на 1970-тите и 1980-тите години.

Процесот на **backpropagation** за пресметување на објективната функција на градиенти, со земање во предвид на тежините од повеќеслојното складиште на модули, е ништо повеќе од апликација што го исполнува правилото на синџир на деривати. Клучот е во тоа да градиентот на објектот во однос на влезот од модулот може да биде пресметан работејќи наназад со градиентот во одност на излезот на модулот (или влезот на под-модулот). Равенката на **backpropagation** може да биде применета низ сите модули, почнувајќи од излезот на врвот (каде мрежата ја дава прогнозата) низ целиот пат до почетокот (од каде се внесуваат податоците). Откако овие градиенти ќе бидат пресметани, понатаму она што следува е да се пресметаат градиентите во однос со тежините на секој модул посебно.

Многу апликации на длабокото учење користат **feedforward**[4] невронски мрежни архитектури кои учат да мапираат непроменлива големина на влезен податок (пр. слика) со излез со непроменлива големина (пр. веројатност за секоја од последователните категории). За да можеме да се движиме од еден слој кон наредниот, се пресметува тежинска сума од влезните податоци на претходното ниво и се пренесуваат преку нелинеарната функција. Во моментот, најпозната не-линеарна функција е **исправувач на линеарни единици** (ReLU), кој е едноставен **полупериодичен исправител f(z) = max(z,0).** Во минатите декади, невронските мрежи користеле поблаги нелинеарни функции, како ***tanh(z)*** или ***1/(1+exp(-z))***, но **ReLU** типично учи многу побргу во мрежи со повеќе слоеви, дозволувајќи обука на длабоки надгледувани мрежи без да е потребно претходна надгледувана обука. Единиците кои не се во влезот или слојот на излезот се конвенционално наречени скриени единици.

Во доцните 1990-ти години, невронските мрежи и **backpropagation** алгоритмите биле широко заборавени од опкружувањето на машинското учење и игнорирани од развивачите на препознавање на говор. Сеуште било присутно мислењето дека употребата на екстрактори кои сами ќе учат било речиси невозможно. Во принцип било присутно мислењето дека едноставното градиентско спуштање би се заробило во сиромашните локални тежински конфигурации кај кои и мала промена би ја променило просечната грешка.

Во практика, сиромашни локални минимуми се многу редок проблем кај големите мрежи. Без оглед на првичните услови, системот скоро секогаш ги достигнува решенијата со многу сличен квалитет. Неодамнешни теоретски и емпириски резултати силно сугерираат дека во основа, локалните минимуми не се сериозни решенија, наместо тоа, пејзажот претставува комбинација од голем број на екстремни точки каде градиентот е нула и површината се искривува нагоре или надолу. Анализата покажала дека постојат многу екстремни точки со само неколку искривувања во правец надолу, но скоро сите од нив имаат многу слични вредности од објективната функција. Според тоа не е многу важно кај кои од овие **екстреми** алгоритмот ќе застане.

[4] мрежни архитектури кај кои конекциите помеѓу јазлите не формираат круг или циклус

Интересот за длабоки мрежи (**feedforward networks)** беше обновен околу 2006 година од група на истражувачи собрани заедно од Канадскиот Институт за Напредни Истражувања (CIFAR). Истражувачите претставија т.е. воведоа постапки за учење без надзор кои можат да креираат слоеви на детекторски функции без потреба од обележани податоци. Потребата за учење на секој слој била за да може да ги реконструира или моделира активностите на детекторите на обележја во подолните слоеви. Со тоа "претходно изучување" на неколку слоеви т.е. детектори на обележја, длабоките мрежи би можеле да се фокусираат кон осетливите вредности. На крај, на врвот би можел да се додаде последен слој на излезни единици и така би се создал еден цел систем на стандарден **backpropagation**. Ова работело неверојатно добро за препознавање на рачно напишани цифри или за детекција на пешаци, особено кога количината на маркирани податоци била многу ограничена. Првата голема примена на овој приод за обука била во препознавањето на говор, а тоа било овозможено со доаѓањето на брзите графички процесорски единици кои беа погодни за програмирање и им овозможиле на истражувачите да трениратмрежи десет или дваесет пати побрзо.

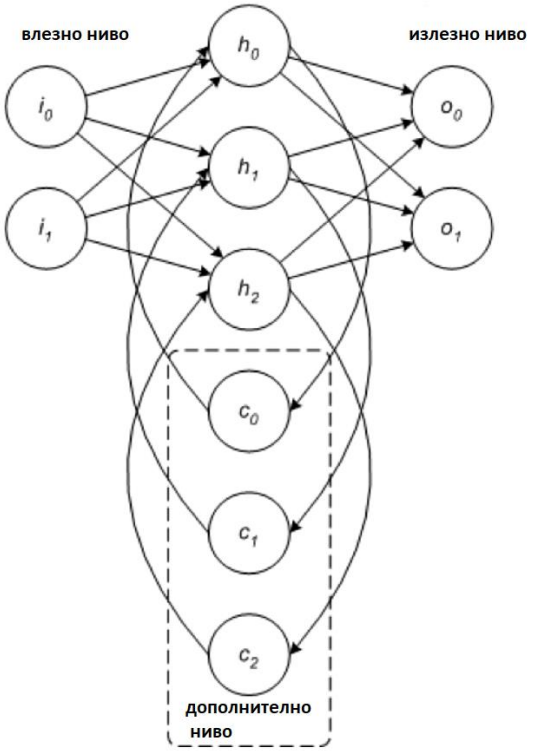
До 2012 година, верзиите на длабоката мрежа од 2009 година беа развиени од многу од главните говорни групи и веќе беа распоредени во Android[5] телефони. За помали множества од податоци, недоволната подготовка помага да се спречи појавата на пренаучување т.е. overfitting[6], што доведува до значително подобра генерализација кога бројот на маркирани примери е мал. Откако веќе еднаш било оживеано длабокото учење, се покажало дека фазата на претходна обука била потребна само за мали множества на податоци.

Постои, сепак, еден посебен вид на длабока, feedforward мрежа која била многу полесно да се обучува и прогнозирала многу подобро од мрежите со целосна поврзаност помеѓу соседните слоеви, тоа е конвулуциската невронска мрежа (ConvNet). Таа постигнала многу практични успеси во текот на периодот кога невронските мрежи биле надвор од користење.

[5] оперативен систем за мобилни телефони
[6] појава во машинското учење последица од пренаучување на податоците за тренирање

3.6.3. Рекурентни невронски мрежи (RNN)

Кога за првпат било воведено учењето со **backpropagation** највозбудливата употреба била за обука на рекурентни невронски мрежи (RNNs). За задачи кои вклучуваат секвенцијални влезови, како што се говорот и јазикот често се користени RNNs. RNN обработува еден елемент од секвенцата во дадено време, одржувајќи во своите скриени единици **state vector** или вектор на состојба кој имплицитно содржи информации за историјата на сите претходни елементи во секвенца. Кога ќе ги погледнеме резултатите од скриените единици на различни дискретни временски рамки, како тие да се излези од различни неврони во длабоката повеќеслојна мрежа, станува јасно како можеме да употребиме backpropagation за обука на RNN.



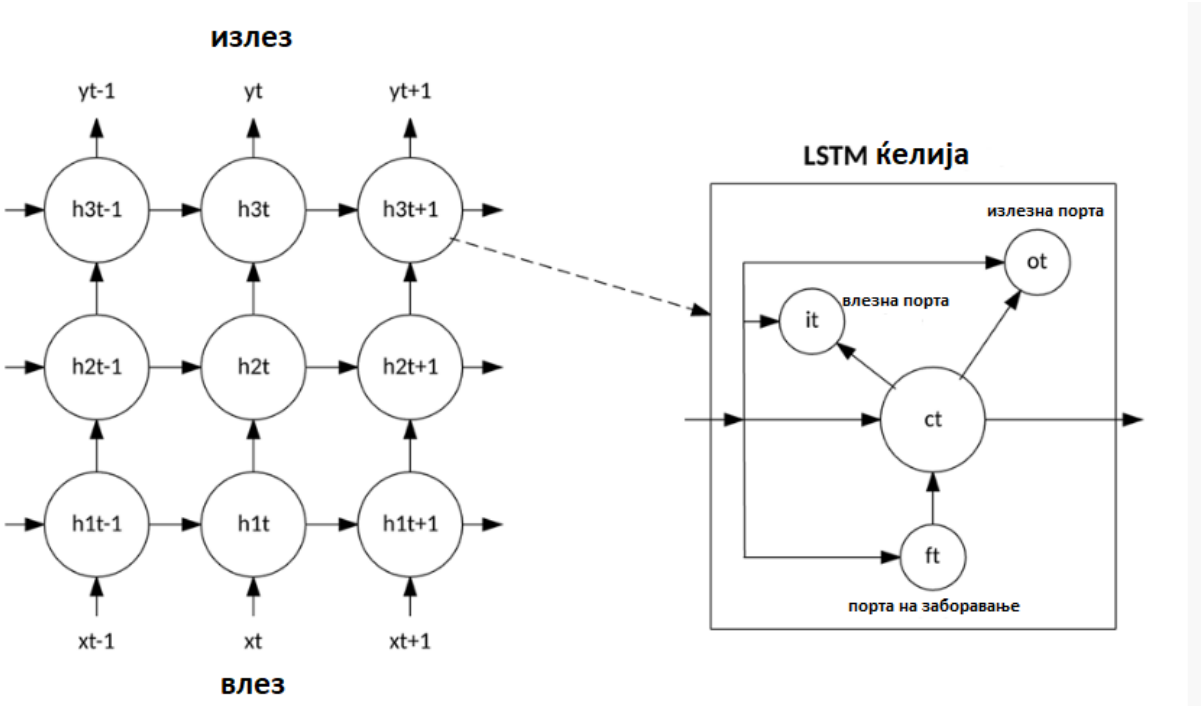
Слика 3: Архитектура на RNN невронски мрежи

Figure 3: RNN architecture

3.6.4. Долги, краткотрајни меморији (LSTM)

Овие мрежи биле креирани во 1997 година, но достигнале поголем напредок во поновото време. Тие денес се користат во многу функционалности на паметните телефони.

Овие мрежи се поразлични од типичните невронско базирани архитектури и наместо тоа се изградени на концепт на мемориска ќелија. Овие ќелии можат да ја меморираат својата вредност за кратко или долго време. Мемориските ќелии содржат три порти кои контролираат како информацијата протекува во и надвор од ќелијата. Последната порта контролира кога информацијата што е во ќелијата може да излезе од неа. Ќелијата исто така има и тежини кои ги контролираат портите.



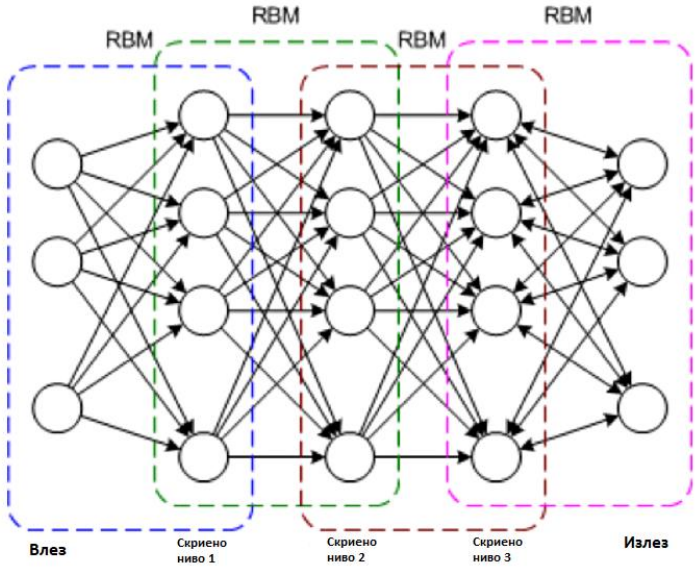
Слика 4: Архитектура на LSTM невронски мрежи

Figure 4: LSTM architecture

3.6.5. Длабоки невронски мрежи на доверба (DBN)

Овие мрежи имаат типична невронска архитектура, но имаат нов алгоритам за тренирање. Кај оваа мрежа секој пар на поврзани нивоа претставува ограничена **Болцманова машина**[7], на овој начин овие мрежи всушност се збир од Болцманови машини. Влезното ниво во овие мрежи го претставуваат суровите влезни податоци, а секое ниво учи апстрактна репрезентација од тие влезови. Излезното ниво се третира поразлично од другите нивоа и тоа ја прави класификацијата. Тренирањето се состои во два чекори: ненадгледувано пред-тренирање и надгледувани завршни процеси.

Во ненадгледуваното пред-тренирање секоја Болцманова машина е тренирана да ја реконструира нејзината влезна информација. Наредната машина се тренира на сличен начин. Процесот продолжува се додека секое ниво не го заврши пред-тренирањето. Кога тој процес ќе заврши започнува завршната фаза. Во оваа фаза, на јазлите им се додаваат лабели кои им даваат до знаење што всушност е контекстот на целата мрежа. На крајот, целиот процес на тренирање се завршува со градиентно спуштање или backpropagation.



Слика 5: Архитектура на DBN невронски мрежи
Figure 5: DBN architecture

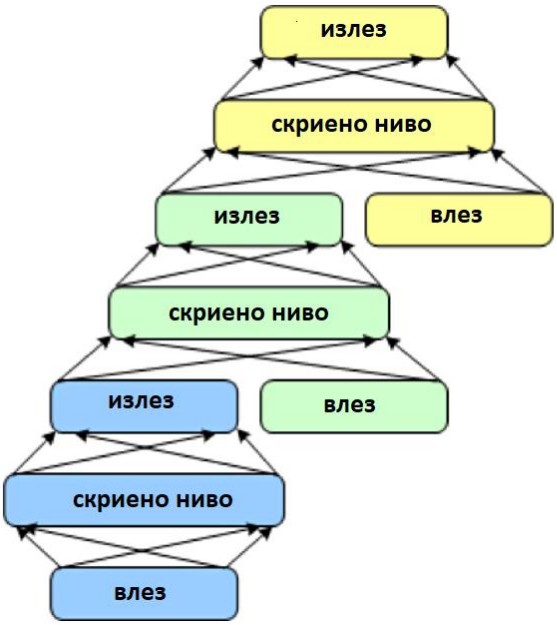
[7] вид на стохастичка рекурентна невронска мрежа

3.6.6. Длабоки “натрупувачки” мрежи (DSNS)

Оваа архитектура се разликува од традиционалните невронски мрежи по тоа што всушност самата мрежа претставува збир од повеќе индивидуални мрежи, сите со свои скриени нивоа. На овој начин тренирањето не е единечен проблем, туку претставува збир од проблеми за секоја под-мрежа.

DSNS се состои од модули каде што секој е за себе подмрежа. За една инстанца од архитектурата се креираат три модули. Секој модул има свое влезно ниво, скриено ниво и излезно ниво. Модулите се поставени еден над друг каде што влезното ниво на еден модул е всушност состав од излезот на претходното ниво и основните влезни податоци. На овој начин се овозможува научување на покомплексни класификации.

DSNS овозможува секој модул да тренира индивидуално, освен тоа што можат да тренираат паралелно. Надгледуваното учење е имплементирано со помош на backpropagation низ целата мрежа.



Слика 6: Архитектура на DSNS невронски мрежи

Figure 6:DSNS architecture

3.7. Конволуциски невронски мрежи (CNN)

Деталното објаснување за овие мрежи го оставив последно бидејќи сакам со поголемо внимание и подлабоко да навлезам во нивната проблематика и функционалност. Тоа го правам со цел најдобро што можам да ја објаснам и доловам нивната работа бидејќи станува збор за мрежи кои најмногу пред сè се користат за обработка на слики што претставува и нашата главна теза во овој труд.

ConvNets[8] се дизајнирани да ги обработуваат податоците кои доаѓаат во форма на повеќекратни низи, на пример слика во боја составена од три 2D низи кои содржат пикселски интензитет во трите канали на боја. Многу од податоците се во форма на повеќекратни низи: 1D за сигнали и секвенци, вклучувајќи го и јазикот; 2D за слики или аудио спектограми; и 3D за видео или волуметриски слики. Постојат четири клучни идеи зад ConvNets кои ги користат предностите на својствата на природните сигнали: локални врски, споделени тежини, здружување и употреба на многу слоеви.

Архитектурата на типичен ConvNet е структурирана како низа од фази. Првите неколку фази се составени од два вида слоеви: конволуциски слоеви и слоеви на здружување. Единиците во конволуцискиот слој се организираат во мапи со карактеристики, во рамките на кои секоја единица е поврзана со локални врски со карактеристиките на претходниот слој преку множество тежини наречени банка за филтрирање. Резултатот од оваа локална тежинска сума потоа се пренесува преку нелинеарност, како што е ReLU. Сите единици во карта со карактеристики ја делат истата банка на филтри. Различни карактеристични мапи во слој користат различни банки на филтри, причината за кое оваа архитектура е двојна. Прво, во низа податоци како што се слики, локалните групи на вредности често се многу поврзани, формирајќи карактеристични локални мотиви кои лесно се откриваат. Второ, локалната статистика на сликите и другите сигнали се инваријантни на локацијата. Со други зборови, ако мотив може да се појави во еден дел од сликата,

[8] кратенка за конволуциски невронски мрежи

може да се појави насекаде, па оттука е идејата за единици на различни локации кои ги делат истите тежини и откриваат иста шема во различни делови на низата.

Иако улогата на конволуцискиот слој е да се откријат локалните сврзани карактеристики со карактеристиките од претходниот слој, улогата на конволуцискиот слој е и да се спојат семантички слични карактеристики во една.

Бидејќи релативните положби на карактеристиките што формираат мотив може да се разликуваат малку, веродостојно детектирање на мотивот може да се направи со грубо гравирање на позицијата на секоја одлика. Една типична единица за собирање го пресметува максимумот од локалниот дел од единици во една мапа на функции (или во неколку мапи со карактеристики). Соседните единици за собирање земаат влез од врски кои се поместени за не повеќе од еден ред или колона, со што се намалува димензијата на претставата и создава инваријантност кон мали смени и нарушувања. Понатаму следуваат две или три фази на составување, здружување на слоеви. Back-пропагацијата преку ConvNet е едноставна исто како и низ обична длабока мрежа, овозможувајќи им на сите тежини во сите банки на филтри да бидат обучени.

Длабоките невронски мрежи го користат својството дека многу природни сигнали претставуваат хиерархија на композиција, во која функции на повисоко ниво се добиваат со компонирање на пониските нивоа. Во сликите, локалните комбинации на рабови формираат мотиви, мотивите се собираат во делови, а деловите формираат објекти. Слични хиерархии постојат во говорот и текстот од звуци на телефони, фонеми, слогови, зборови и реченици. Соединувањето овозможува прикажувањето да се менува многу малку кога елементите во претходниот слој се разликуваат во положбата и изгледот.

Конволуциските и здружувачки слоеви во ConvNets се директно инспирирани од класичните претстави за едноставни клетки и сложени клетки во визуелната неврологија, а целокупната архитектура потсетува на хиерархијата на LGN-V1-V2-V4-IT во виралната патека на визуелниот кортекс.

Примитивна 1D ConvNet наречена временска задоцнета невронска мрежа била користена за препознавање на фонеми и едноставни зборови. Имало бројни апликации на конволуциски мрежи во раните 1990-ти, како што се мрежи за

препознавање говор и читање на документи. Системот за читање документи го користел ConvNet обучен заедно со веројатниот модел кој ги имплементира јазичните ограничувања. До крајот на 1990-тите овој систем читаше над 10% од сите проверки во САД. Голем број на признанија за оптички знаци базирани на ConvNet и системи за препознавање на ракопис подоцна беа распоредени од Microsoft. ConvNets исто така беа тестирани во почетокот на 1990-тите за откривање објект во природни слики, вклучувајќи лица и раце и за препознавање лице.

3.7.1. Успехот на конволуциските мрежи

Во последниве седум години, квалитетот на класификацијата на слики и детектирањето објекти значително се подобрило и усовршило благодарение на методите на длабоко учење. Конволуциските невронски мрежи донеле револуција во областа на компјутерската визија и не само што континуирано ја подобруваат точноста во класификацијата на слики, туку играат и клучна улога во класификацијата на сцени, детекцијата на објекти, семантичка сегментација, текстуален опис на слика итн. Ова биле задачи во кои маркираните податоци се релативно изобилни, како што се препознавање сообраќајни знаци, сегментација на биолошките слики и откривање на лица, текст, пешаци и човечки тела во природни слики. Главен неодамнешен и практичен успех на ConvNets е препознавање на лице (***face recognition)***.

Битно е да се знае дека сликите може да се маркираат т.е. подредуваат до ниво на пиксел и тоа има големо влијание во технологијата за автономни мобилни роботи и авто-возење кај автомобилите. Компаниите како што се **Mobileye**[9] и **NVIDIA** користат методи базирани на ConvNet во нивните претстојни системи за визија кај автомобилите. Другите апликации кои се исто така битни вклучуваат природни јазични разбирања и препознавање на говор.

И покрај овие успеси, ConvNets во голема мера биле “заборавени” од страна на “традиционалните” заедници кои се занимаваат со компјутерска визија и

[9] компанија која дизајнира системи за асистенција во возењето базирани на компјутерска визија

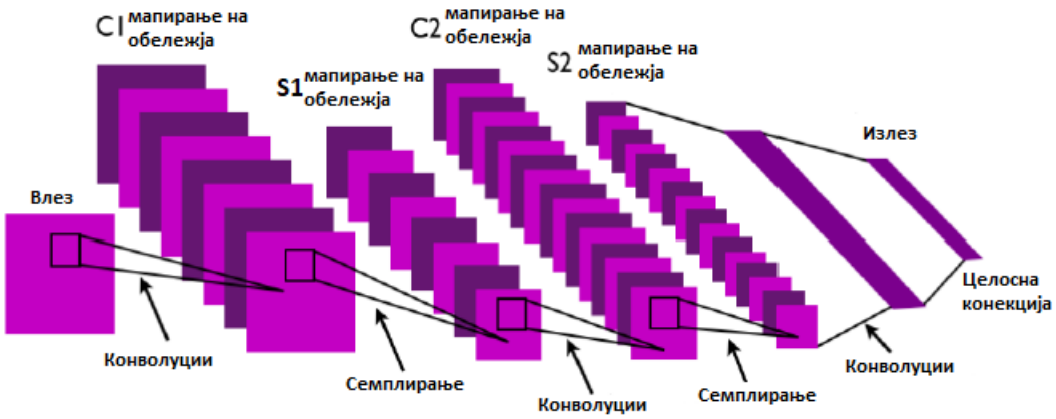
машинско учење, се до натпреварот **ImageNet**[10] во 2012 година. Тогаш, длабоките конволуциски мрежи биле применети на збир на податоци од околу милион слики од интернет кои содржеле 1.000 различни класи и постигнале спектакуларни резултати со речиси преполовување на стапката на грешка споредбено со нивните конкуренти. Овој успех произлегува од ефикасната употреба на графичките процесори, ReLUs, нова техника за регулација, наречена ***dropout*** и техники за креирање повеќе примери за обука(тренирање) со деформирање на веќе постоечките податоци. Овој успех придонел за револуција во компјутерската визија, така ConvNets сега претставуваат доминантен пристап во скоро сите задачи за препознавање и детектирање или доближување до човечките перформанси за некои задачи.

Најновите ConvNet архитектури имаат од десет до дваесет слоеви на ReLUs, стотици милиони тежини и милијарди врски помеѓу единиците. Додека обуката на такви големи мрежи можеше да потрае неколку недели пред само две години, напредокот во хардверот, софтверот и паралелизацијата на алгоритмите го намали времето на обука на неколку часа. Изработката на системи за визија базирани на ConvNet предизвика повеќето големи технолошки компании, вклучувајќи ги и Гугл, Фејсбук, Мајкрософт, IBM, Yahoo, Твитер и Adobe, како и брзо растечките start-up компании, да иницираат истражувања и проекти за поголем развој на ConvNet мрежи базирани на производи за разбирање на слики и услуги. Неколку компании како што се NVIDIA, Mobileye, Intel, Qualcom и Samsung развиваат ConvNet чипови за да овозможат визија на апликации во реално време кај паметните телефони, камери, роботи и авто-возење т.е. управување на автомобили.

[10] натпревар за компјутерска визија во кој тимовите се натпреварууваат со нивните алгоритми врз база на податоци

3.7.2. Конволуциските мрежи и процесирањето на слики

Овие невронски мрежи се едни од најмоќните класи на длабоките невронски мрежи во процесирањето на слики. Конволуциските мрежи содржат три вида на нивоа: конволуциски нивоа, ниво за подсемплирање и ниво за целосна конекција. Во продолжение накратко ќе зборуваме за секое ниво.

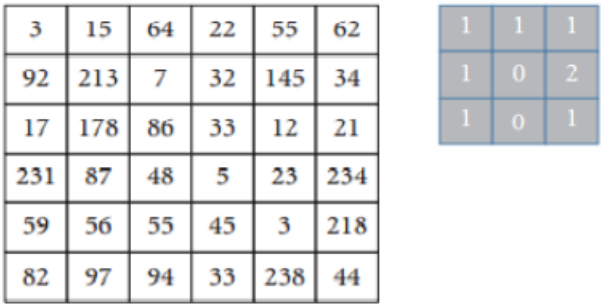


Слика 7: Архитектура на CNN невронски мрежи

Figure 7: CNN architecture

- Конволуциско ниво

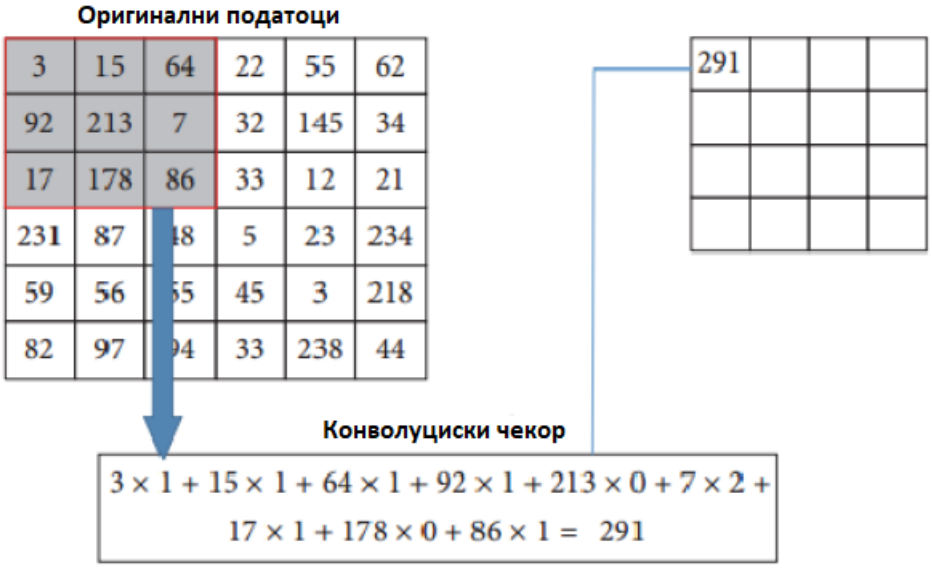
На следната слика е прикажано конволуциското ниво каде што левата матрица е дигитална репрезентација на слики и тоа претставува влезот, додека матрицата десно е конволуциската матрица. Конволуциското ниво го зема конволуцискиот приказ на сликата од конволуциската матрица и ја генерира сликата на излез.



Слика 8: Дигитална репрезентација на слика и матрица на конволуција

Figure 8: Digital image representation and convolution matrix

Најчесто, конволуциската матрица е наречена филтер, а излезната слика е наречена одговор на филтерот или мапа на филтерот.



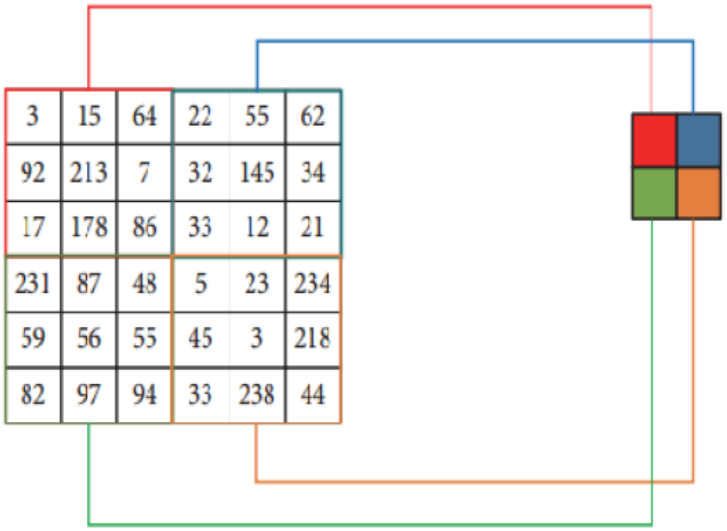
Слика 9: Пример за конволуциско пресметување

Figure 9: An example of convolution calculation

На следнава слика е прикажан пример за конволуциска калкулација каде што блок од пиксели преку даден чекор т.е. филтер генерира резултат за пиксел во новата слика.

- Ниво за подсемплирање

Ова ниво е многу битно во една конволуциска невронска мрежа бидејќи во главно се користи за намалување на големината на влезната слика со цел на невронската мрежа да и се даде инваријатност и робустност. Најупотребуваниот модул за подсемплирање во процесирањето на слики е **максимално здружување (max pooling)**. Со него најпрво сликата се дели на блокови каде што пикселот со највисоката вредност се зема како вредност на пикселот во излезната слика. Ова се прави бидејќи самото ниво на подсемплирање користи помалку параметри и така полесно учи.



Слика 10: Пример за нивото на подсемплирање

Figure 10: An example of subsampling layer

- Ниво за целосна конекција

Ова ниво овозможува невронската мрежа да се однесува како feed-forward мрежа со вектори кои имаат преддефинирана должина и да се користат за понатамошно процесирање.

3.7.3. Апликации

Длабокото учење е широко распространето низ многу различни полиња како што се компјутерската визија, процесирањето на сигнали и препознавањето на говор.

- CNN базирани апликации за компјутерска визија

Како што веќе напоменавме, конволуциските невронски мрежи се многу моќна алатка за препознавање на слики и нивна класификација. Овие различни видови на CNN обично се тестираат на познатата ImageNet Large Scale Visual Recognition Challenge (ILSVRC) база на податоци. Во 2012 година CNN мрежите победиле на ImageNet натпреварот и од тогаш нивните методи донеле револуција во компјутерската визија. Тие постигнале многу добри резултати во детектирање објекти, сегментација на објекти и препознавање на објекти и региони во слика.

- CNN за препознавање на лица

Препознавањето на лица е една од најбитната задача во компјутерската визија уште од 1970 година. Системите за препознавање на лица обично се состојат од четири чекори. Најпрво со помош на детектор се локализираат и изолираат сите лица. Понатаму, секое лице посебно се претпроцесира со користење на 2D или 3D методи. Следно, се пронаоѓаат белези и се формира нова репрезентација со помали димензии, за на крај класификаторот да направи предикција базирана на помалата репрезентација. Клучот до добри перформанси во системите за препознавање на лица е одржување на добра и ефективна репрезентација со помали димензии. Денес, едни од помоќните и најпознатите системи за препознавање на лица како што се Facebook Deep Face, Google FaceNet се базирани на CNN мрежи.

Наместо да се користат рачно “извадени” обележја, CNN директно се вклучува врз RGB вредностите на пикселите и се користи детектор на обележја за

да создаде репрезентација на влезната слика од карактеристиките на лицето. Со цел да се нормализира влезната слика и лицето да се направи робустно гледано од повеќе агли, Deep Face го моделира лицето во 3D. Веќе нормализираната слика се дава на филтерот за конекција каде на крај имаме три локално поврзани нивоа и две нивоа целосно поврзани за да ја направат крајната прогноза.

Слика 11: Тек на процесот при CNN препознавање на лице

Figure 11: Logic flow of CNN based face recognition

Архитектурата на Deep Face е прикажана на оваа слика. Иако Deep Face успева најефикасно да направи проценка и препознавање, неговата репрезентација е малку потешко да се интерпретира и употреби бидејќи лицата на една иста личност не се кластерирани во текот на процесот на тренирање. Спротивно на тоа, Face Net дефинира триплет на функција за загуба директно од репрезентацијата со што овозможува процесот на тренирање да ја кластира репрезентацијата на лицето од една личност. Исто така треба да се напомене и дека Open Face користи едноставни 2D афини трансформации за влезната слика со лицето.

3.8. OpenCV

OpenCV е библиотека која содржи програмски функции наменети за целите на компјутерската визија и машинското учење. Првото појавување на оваа библиотека датира уште од 1999 година кога за прв пат бил направен проект вклучувајќи OpenCV од страна на Интел.

Библиотеката содржи повеќе од 2500 оптимизирани алгоритми кои вклучуваат машинско учење и компјутерска визија. Со овие алгоритми може да се изврши детекција и препознавање на лица, идентификација на предмети и објекти, класификација на човечки движења во видеа како и следење на објекти, наоѓање на слични слики во база од слики, следење на движење на око, препознавање на сценарија итн.

OpenCV е напишана со програмскиот јазик C++, но може да се употребува и да работи заедно со повеќе работни околини и јазици како Python, Ruby или Java како и на повеќе платформи како Windows, Linux, macOS и др.

Официјалните документи и изворни програми може да се превземат бесплатно од репозиториумот на GitHub[11].

[11] интернет платформа на која можат да се најдат изворни кодови од многу компјутерски библиотеки, алгоритми, програми и сл.

3.8.1. Детектирање и препознавање на лица со OpenCV

Помеѓу многуте причини кои ја прават компјутерската визија фасцинантна тема е фактот дека таа успева да ги направи реални оние задачи кои некогаш ни изгледале футуристички и недостижни. Една таква способност е детекцијата и препознавањето на лица која е вградена функционалност на OpenCV.

Во понатамошното разлагање ќе зборуваме за некои од функционалностите и можностите на OpenCV за детекцијата на лица, конкретно ќе зборуваме **за Haar каскадната** класификација, комбинирање на повеќе такви класификации во хиерархија итн. Исто така ќе зборуваме и за правоаголните области во рамките на една слика и како со нивно преместување или промена може да манипулираме со регионите на сликата кои ги следиме.

Целта е на крај да успееме сами да заклучиме како работи целосната имплементација на оваа библиотека и сето тоа да го докажеме со пример при што како дел од овој труд ќе вклучам изработување на веб апликација којашто има две главни функционалности детекција и препознавање на лице и истата е изработена со користење на OpenCV библиотеката и програмскиот јазик Python.

Апликацијата е изработена со помош на работната околина Django и користи неколку импортирани библиотеки. Целосната работа и функции ќе ги објаснам во понатамошниот тек на трудот.

3.8.2. Концептот на Haar каскадната класификација

Кога зборуваме за класификација на слика, мораме да бидеме конкретни и одлучни во она што сакаме да го видиме и постигнеме. Фотографиите сами во себе кријат многу разлики и детали кои нам луѓето на прв поглед со голо око ни изгледаат исти или слични и не обрнуваме значително внимание на аголот, осветленоста, оддалеченоста, дигиталниот шум итн. Всушност понекогаш таквите детали иако се

битни не влијаат на суштината на сликата и тоа што таа е. Пример за тоа е “фактот” дека секоја снегулка е уникатна и посебна сама за себе и тие особености можат да се забележат кога би ги гледале снегулките под микроскоп, но таквите детални белези не влијаат на фактот дека тоа се снегулки иако сите се различни помеѓу себе.

Токму поради таа причина, постои фазата т.е. процесот на екстракција на белези односно обележја од слика. Во една слика мора да постојат многу помалку белези од бројот на пиксели и покрај тоа што во еден пиксел може да се скријат повеќе белези. Нивото на сличност помеѓу две слики се добива врз основа на **Евклидовото растојание** помеѓу соодветните обележја на сликите. Растојанието може да биде дефинирано преку повеќе услови како што се просторни координати или вредности на боја. Обележјата што се користат во Haar класификацијата се едни од тие што се користат во следење на лица во реално време. Секое обележје е избрано така да прикажува контраст помеѓу акцентираните региони на сликата како што се рабови, тенки линии, темиња и сл.

За многу различни слики, обележјата можат да вариираат зависно од големината на регионот кој се зема од сликата и кој уште се нарекува **window size**. За две слики кои се разликуваат само во големината, би требало да дадат слични обележја иако се од прозорци со различна големина. Понекогаш е корисно да можеме да генерираме обележја за повеќе големини на еден прозорец. Таквата колекција од обележја е наречена **каскада**. Од тука можеме да кажеме дека каскадите на Haar се робустни и приспобливи за промена во големината.

Каскадите на Haar за првпат биле искористени од страна на *Пол Виола и Мајкл Џонс* во 2001 година. Секоја карактеристика определена со Haar го опишува контрастот меѓу соседните региони на сликата. На пример, рабови, темиња и тенки линии генерираат посебни карактеристики. Haar каскадите во OpenCV не се робустни на ротација и на позначителни промени, како што на пример, надолу превртено лице не се смета за слично со правилна слика на лице и лицето гледано од профил не се смета за исто како лицето гледано од напред. Можеби треба да се

поработи на оваа област која би ги направила каскадите да можат да работат со ротирани и превртени лица.

3.8.2.1. Како работат каскадите на Haar?

Познато е дека Haar каскадите се способни да детектираат лица и делови од телото, но исто така можат да бидат тренирани и да го идентификуваат било кој предмет или објект. За да алгоритмот може да даде точна детекција, неговиот класификатор мора да биде трениран на “позитивни” и “негативни” слики. Тоа значи дека мора да биде трениран на слики кои на пример содржат лице и слики кои не содржат лице.

Првиот чекор е да се одберат и извадат сите Haar обележја. Обележјата во Haar се правоаголни региони на конкретна локација во кои Haar ги собира сите интензитети на пикселите во рамките на правоаголниците и ја калкулира разликата помеѓу тие суми. Меѓутоа, помеѓу сите тие обележја како правоаголници секогаш ќе постојат некои кои се ирелевантни и едноставно не помагаат во детекцијата. Тогаш се поставува прашањето како од толку многу обележја да се издвојат само најдобрите и најпотребните? Тоа се решава со концептот на **Adaboost**[12] кој ги селектира најдобрите обележја и го тренира класификатор на нив.

Во процесот на детекција, прозорец со одредена големина се движи низ сликата и за секој дел се пресметуваат сите Haar обележја. При тоа се пресметуваат сите разлики. Разликата се споредува со веќе научен влез кој знае да ги препознае објектите од оние кои не се објекти. Потребно е голем број на Haar обележја за да се опише еден објект со доволно голема точност бидејќи Haar се смета за слаб класификатор, затоа повеќе од нив се организирани како каскадни класификатори.

Каскадниот класификатор содржи повеќе етапи и сите тие претставуваат слаби клафикатори како Haar. Слабите класификатори се научуваат преку boosting методи кои допринесуваат да се тренира каскадниот класификатор до голема точност.

[12] машинско учење кое се користи за подобрување на перформансите

Низ етепите, секој регион се класифицира или како позитивен или како негативен. Позитивен означува дека во тој регион бил пронајден објект и во тој случај регионот оди кон следната етапа, а негативен означува дека не е пронајден објект и прозорецот оди кон наредниот регион итн. Детекторот на крај дава извештај во кој регион бил пронајден објектот ако и последната етапа го класифицира тој регион како позитивен.

Етапите се создадени да ги отфрлаат негативните региони најбрзо што можат. Резултатите на крај се дека поголемиот број на региони не го содржат објектот кој ни треба, а позитивните региони се ретки, но сепак вреди да се изгуби време барајќи ги.

Слика 12: Визуелен пример на етапи во каскадната класификација и позитивен регион

Figure 12: Visual representation of stage in cascade classification and positive region

- Точно позитивно е случај кога позитивен податок е точно класифициран;
- Неточно позитивно е случај кога негативен податок по грешка е класифициран како позитивен;
- Неточно негативно е случај кога позитивен податок е грешно класифициран како негативен.

За да работи добро, секоја етапа на класификаторот мора да има ниска **“лажна негативна стапка”** т.е. случаите кога позитивниот примерок е третиран како негативен да бидат многу ретки или воопшто да не постојат. Ако етапата неточно го класифицира објектот како негативен, тогаш класификацијата запира и тие грешки се фатални. Спротивно на тоа е случај кога етапите имаат висока **“лажна позитивна стапка”** т.е. кога регионите ги класифицираат како позитивни, а реално тие се негативни. Сепак тие грешки полесно се поправаат со додавање на меѓу етапи и се намалува бројот на лажно позитивни региони.

3.8.3. Употреба на OpenCV во детекција на лице

Иако звучи различно, всушност детекцијата на лице во слика и во видео е многу сличен процес и операција. Суштинската разлика е во тоа што детекцијата на лице во видео претставува употребување на истиот процес на детекција, но на повеќе рамки што се вчитуваат.

3.8.3.1. Превземање на Haar каскадите

Како дел од нашата конфигурација на OpenCV, пожелно е да имаме директориум наречен **Haarcascades**. Во него треба да ги зачуваме каскадите кои се обучени за одредени задачи користејќи алатки кои доаѓаат со OpenCV. Целосната патека на директориумот зависи од нашиот систем и начинот на поставување на OpenCV. Зависно од тоа каква цел сакаме да постигнеме, можеме да ги снимаме оние каскади кои ни требаат. Во овој проект бидејќи ќе работиме на детекција и препознавање на лице, потребни ни се две каскади и тоа:

- haarcascade_frontalface_alt
- haarcascade_frontalface_default

3.8.3.2. Детекција на лице на слика

Првата функционалност на апликацијата е да може да детектира лице или повеќе лица во слика. Постапката започнува со прикачување на било која слика од компјутерот за која би сакале да ја видиме детекцијата. За овој процес апликацијата може да прима слика со едно лице или слика со повеќе лица на неа.

Откако сликата ќе биде прикачена, со кликање на копчето Detect ја повикуваме скриптата ***detect.py*** која содржи функции и алгоритми за вршење на процесот на детекција. Секоја скрипта мора да започнува со ***import cv2*** редот бидејќи така ја вчитуваме библиотеката OpenCV со која понатаму ќе ја процесираме сликата. Освен тоа, потребно е да вчитаме и други библиотеки како што се ***numpy*** која служи за вршење математички функции врз сликата, ***matplotlib*** која има функции со кои може да се менуваат фигури и многу други.

Освен потребните библиотеки, во скриптата се определува Haar каскадата која треба да се користи во детекцијата на лице. Во овој случај, каскадата ја вчитувме преку патека од фолдерот во рамките на проектот каде претходно ги снимавме сите каскади кои ќе ни требаат во понатамошната работа.

Линијата код изгледа вака:

```
face_detector =
"D:/WORK/DPOpenCV/HaarCascades/haarcascade_frontalface_alt.xml"
```

Понатаму скриптата продолжува со функцијата ***detect_face*** која ќе биде објаснета подолу:

```
def detect_face(request):

    face_detector = "D:/WORK/DPOpenCV/HaarCascades/haarcascade_frontalface_alt.xml"

    if request.method == "POST":

        if request.FILES.get("image", None) is None:
            return render(request, 'uploadtest.html')

        if request.FILES.get("image", None) is not None:
            image_to_read = read_image(stream = request.FILES["image"])
            original_image=image_to_read

        image_to_read = cv2.cvtColor(image_to_read, cv2.COLOR_BGR2GRAY)

        detector_value = cv2.CascadeClassifier(face_detector)

        values = detector_value.detectMultiScale(image_to_read,
                                        scaleFactor=1.1,
                                        minNeighbors = 5,
                                        minSize=(30,30),
                                        flags = cv2.CASCADE_SCALE_IMAGE)

        values=[(int(a), int(b), int(a+c), int(b+d)) for (a,b,c,d) in values]

        default.update({"#of_faces": len(values),
                    "faces":values,
                    "safely_executed": True })

    for (w,x,y,z) in default["faces"]:
        cv2.rectangle(original_image,(w,x), (y,z), (51, 51, 255), 6)

    cv2.imwrite('D:/WORK/DPOpenCV/faces/static/detectedfaces/doneimage.png',
original_image)
    return render(request, 'detected.html')
```

Функцијата нема аргументи бидејќи прима ***request*** преку ***html*** формата за прикачување на слика, каде со помош на методот **POST** сликата се испраќа кон скриптата за понатамошна обработка.

Проверуваме дали методот е POST и проверуваме дали воопшто постои слика испратена во барањето, бидејќи ако не постои не би требало да ги

извршуваме понатамошните функции поради избегнување на појава на грешки. Ако сликата не е испратена во барањето ништо понатаму нема да се изврши.

```
if request.FILES.get("image", None) is not None:
        image_to_read = read_image(stream =request.FILES["image"])
        original_image=image_to_read
```

Во случај кога сликата е прикачена и испратена во барањето, во променливата *image_to_read* се повикува функцијата *read_image* извршена над прикачената слика.

Следниот код ни ја покажува функцијата *read_image*:

```
def read_image(stream=None):

    if stream is not None:
        data_temp = stream.read()

        image = np.asarray(bytearray(data_temp), dtype="uint8")

        image = cv2.imdecode(image, cv2.IMREAD_COLOR)

    return image
```

Ако постои слика, ја вчитуваме и ја зачувуваме во привремена променлива врз која извршуваме неколку математички функции потребни за сликата да ја форматираме во посакуваната форма.

- **bytearray** ни враќа низа од битови за сликата;
- **np.asarray** во комбинација со uint8 ја форматира сликата во низа од осум битни цели броеви;

- **cv2.imdecode** ја чита сликата и каналите ги поставува во BGR редослед

Откако сликата помина низ функцијата ***read_image***, се враќаме во функцијата ***detect_face*** и продолжуваме со постапка на претворање на сликата во црно бела слика бидејќи OpenCV може да препознава лица од таков формат на слики и го повикуваме објектот за лица од класата **CascadeClassifier** кој е одговорен за препознавањето на лицата. Наредниот чекор е всушност делот каде се случува детекцијата на лица преку функцијата ***detector_value.detectMultiScale*** што претставува вградена функција која објектот ја користи за детекција.

Параметрите кои се доделуваат на оваа функција се најпрво сликата во нејзината црнобела форма, втор параметар е факторот на скалирање што ја означува процентуалната поделба на сликата при секоја итерација, третиот параметар е минималниот број на соседи што го определува минималниот број на соседи задржани од еден правоаголник при секоја итерација, minSize означува колкава е минималната големина што може да ја има објектот и flags е параметар на каскадата.

```
detector_value.detectMultiScale(image_to_read,
                                scaleFactor=1.1,
                                minNeighbors = 5,
                                minSize=(30,30),
                                flags = cv2.CASCADE_SCALE_IMAGE)
```

Во променливата values се зачуваат димензиите на сите четириаголници кои треба да се појават на сликата со оглед на тоа дали ќе се детектира едно или повеќе лица. Тие се чуваат како координати со кои се определува левото долно теме и десното горно теме на правоаголникот. Со помош на методата *cv2.rectangle* исцртуваме правоаголници на сликата со дадени координати *x* и *y* кои ги

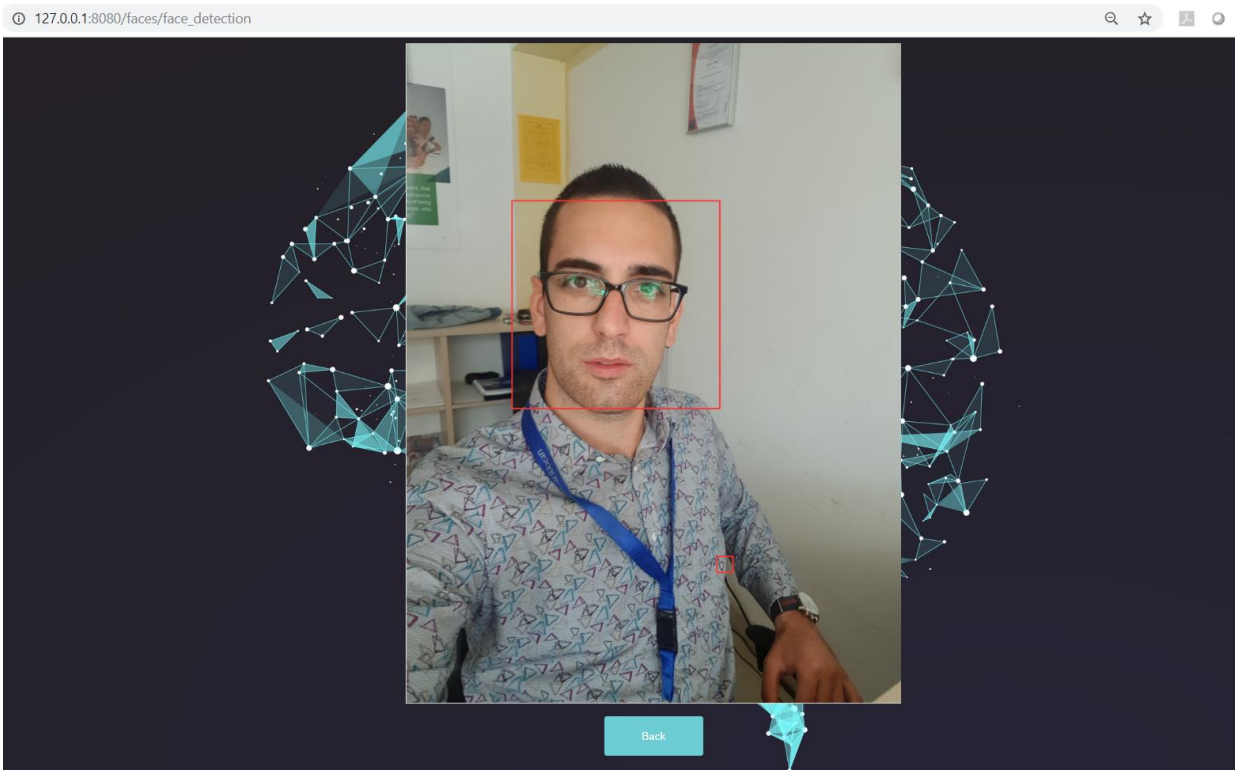
претставуваат најлевата и највисоката точка, а *w* и *h* ја претставуваат ширината и висината на правоаголникот.

Со наредните редови код ќе исцртаме црвени правоаголници околу лицата кои ќе ги најдеме вртејќи низ низата *faces*, а овде ќе ја користиме оригиналната слика, а не нејзината црнобела верзија.

```
for (w,x,y,z) in default["faces"]:
    cv2.rectangle(original_image,(w,x), (y,z), (51, 51, 255), 6)
```

На крај, во нашиот случај сликата со детектираното лице ја рендерираме повторно во **html** страна, но освен тоа можеме да креираме *namedWindow* инстанца за да ја прикажеме сликата по целиот процес, а за да спречиме брзо затворање на сликата користиме *waitKey* функција која го затвора прозорецот кога ќе претиснеме на некое копче.

```
cv2.namedWindow('Faces Detected !! ')
cv2.imshow('Faces Detected !! , original_image)
cv2.imwrite('./faces.jpg, original_image)
cv2.waitKey(0)
```



Слика 13: Приказ на сликата по извршување на функцијата за детекција на лице

Figure 13: Image representation after running function for face detection

3.8.4. Употреба на OpenCV во препознавање на лице

Препознавањето на лица претставува лесна задача кога станува збор за човекот. Експериментите покажале дека бебињата кои се стари само неколку дена можат да разликуваат познато од непознато лице. Тогаш зошто би било тешко компјутерот да може да ги препознава лицата?

Докажано е дека човечкиот мозок има посебни ќелии кои се специјализирани да одговараат на специфични белези од некоја сцена, како што се линии, рабови, агли и движења. Според тоа ние сме создадени околината да ја гледаме во целина, а не дел по дел бидејќи нашиот визуелен кортекс успева некако да ги комбинира сите различни извори на слики и информации околу нас создавајќи "успешна" слика.

Препознавањето на лице е уште една фантастична функционалност на OpenCV која ја нуди можноста на дадена програма, слика или видео да ја идентификува личноста на неа. Еден од начините да се постигне препознавањето на лице е преку тренирање на програмата со нејзино преполнување со класифицирани слики или база од лица на личности. За да започнеме со испробување на оваа функционалност најпрво ни треба извор на слики. Можеме да користиме веќе постоечки бази достапни на интернет или можеме сами да ја креираме базата од лица. За да го тестираме препознавањето на лице, понатаму треба како примерок да користиме една од сликите кои се зачувани во базата, а потоа можеме да користиме и други слики од личностите кои ги имаме во базата.

3.8.4.1. Процес на препознавање

OpenCV доаѓа со три основни методи за препознавање на лица базирани на три различни алгоритми и тоа: Eigenfaces, Fisherfaces и Local Binary Pattern Histograms(LBPH). Сите тие методи следат сличен процес на работа, сите користат множество од класифицирани слики (базата на податоци што содржи многубројни примероци за една личност), понатаму сите "тренираат" на тие податоци, прават

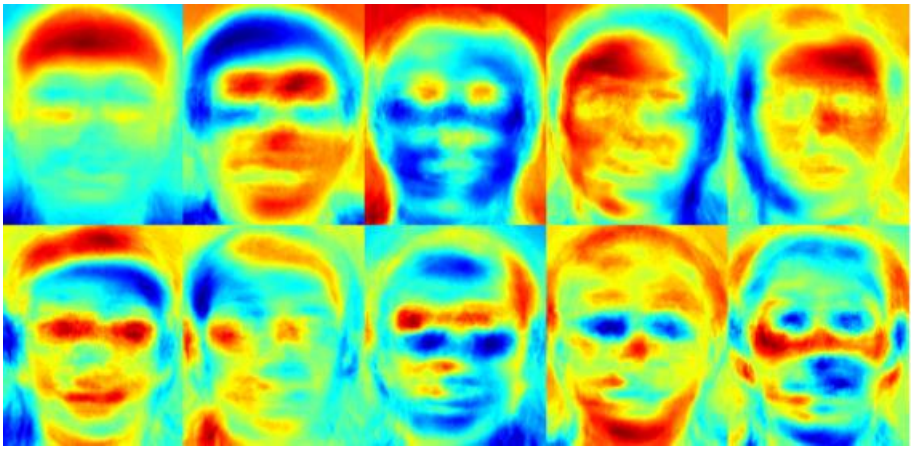
анализи и одредуваат два битни елементи: дали објектот е идентификуван и со колку голема веродостојност. Сепак тие се разликуваат во начинот на којшто ја гледаат сликата и како ја обработуваат. Во наредниот дел ќе подетално ќе ги објаснам секој од методите поединечно.

- **Eigenfaces алгоритам**

Пристапот на овој алгоритам кон сликите бил поттикнат од хипотезата дека не сите делови од лицето се еднакво битни и корисни во процесот на препознавање. Всушност тоа било поттикнато од фактот дека кога гледаме во некоја личност ние неа ја препознаваме по одредени обележја како што се очите, челото или пак образите и како тие се однесуваат еден кон друг. Затоа овој алгоритам се фокусирал на местата каде што има максимална варијација. Како на пример кога ги гледаме очите до носот или пак носот до устата, забележуваме дека постојат многу впечатливи разлики помеѓу овие области.

Кога гледаме повеќе лица, нив ги споредуваме според воочливите разлики токму во тие области и така ја наоѓаме максималната варијација низ лицата и можеме да ги разликуваме едно од друго. На тој начин работи и Eigenfaces алгоритмот. Сите слики кои ги има за тренирање од сите лица ги гледа како една целина и се обидува да извади компоненти кои се релевантни и можат да се употребат, а другите ги отфрла. Тие компоненти се нарекуваат главни компоненти.

Освен што овој алгоритам ги детектира корисните обележја во сликите, тој има способност да памети кои главни компоненти припаѓале на кое лице.



Слика 14: Изваден обележја од повеќе лица

Figure 14: Extracted variance from a list of faces

Така кога на алгоритмот ќе му се даде нова слика за препознавање тој врши неколку математички концепти така што најпрво ги идентификува основните компоненти во сликата, ги идентификува основните елементи од множеството на примероци (базата на податоци), ја пресметува дивергенцијата на дадениот примерок во споредба со базата и одредува вредност. Колку е помала вредноста, толку е помала разликата на лицето од базата со лицето од сликата што ја разгледуваме. Она лице кое добило најниска разлика е всушност она кое покажало најголемо совпаѓање и тоа се одредува како препознаено лице.

- **Fisherfaces алгоритам**

Fisherfaces се базира на истите принципи како и Eigenfaces, но користи малку покомплексна логика и се стреми кон добивање поточни резултати. Разликата е во тоа што овој алгоритам се стреми кон идентификување на сличностите, а не на разликите помеѓу лицата. Тоа се покажало како успешно бидејќи многу често разликите во лицата се појавуваат поради надворешни фактори како што се на пример светлината на самата слика. Тргнувајќи од тоа ако се гледаат разликите може да се направи грешка во препознавањето, но ако се гледаат сличностите би дошле до вистинскиот резултат. Овој алгоритам, исто така, зема два параметри и ги отфрла оние лице кои имаат помала вредност на точност од бараната.

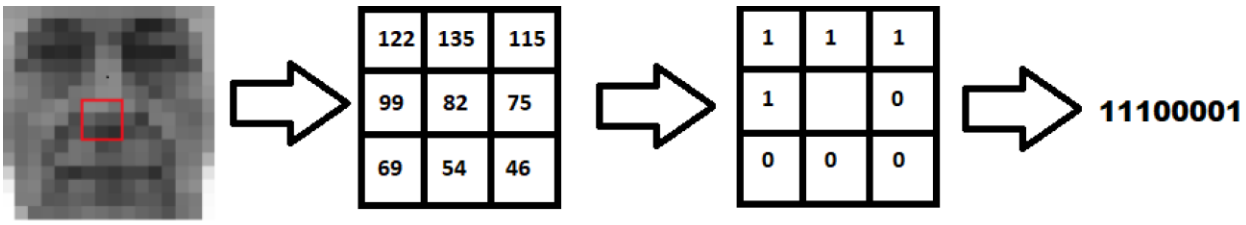
Слика 15:Реконструкција на слика со FisherFace

Figure 15: Fisherface reconstruction

- **Препознавање на лице со LBPH**

LBPH алгоритмот е направен со цел да може да ги занемари можните разлики кај сликите направени од промена на светлината или други влијателни услови. Идејата за овој алгоритам бил да не ја гледа сликата како целина, туку да најде локална структура споредувајќи го секој пиксел со неговиот соседен пиксел.

Алгоритмот зема 3x3 прозорец од пиксели на сликата и го споредува централниот пиксел со сите пиксели околу него. Пикселите што имаат поголема вредност од централниот се земаат како 1, а оние кои имаат помала вредност од централниот се земаат како 0. За да се добие бинарниот број броевите се читаат во насока на стрелката на часовникот почнувајќи од горниот лев број.



Слика 16: LBP конверзија на пикселите во бинарен број

Figure 16: LBP conversion to binary

Откако ќе се добие бинарниот број, тој се претвора во децимален кој во овој случај би бил еднаков со 225. По добивањето на сите децимални броеви се добива една сосема поразлична претстава на сликата и користејќи ја таа слика може да се креира хистограм за секој регион. Хистограмот ја претставува фреквенцијата на секој интензитет од пикселите, за да потоа се прави споен хистограм од сите добиени хистограми.

Така за да се најде која слика е соодветна со нашата прикачена слика, треба само да се споредат двата хистограми преку Евклидово растојание или сл.

Алгоритмот ќе го врати пресметаното растојание кое се толкува како степен на “доверба”. Овој метод е многу флексибилен и е единствен алгоритам кој овозможува дадениот примерок да биде различен по форма или големина од примероците во базата.

3.8.4.2. Како да се генерираат потребните податоци за препознавање на лица?

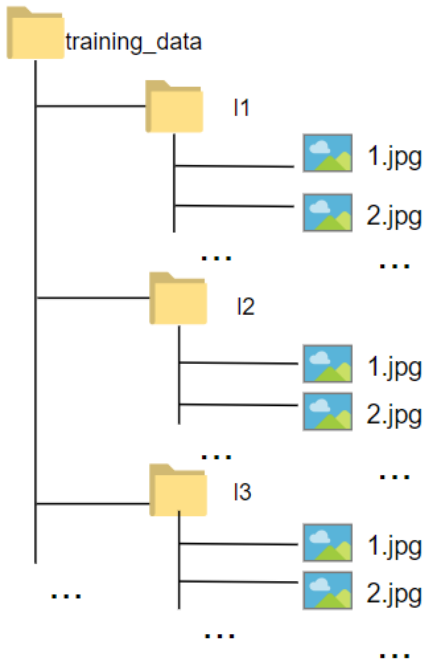
Сликите во базата на податоци мора да задоволуваат неколку критериуми и тоа да бидат црнобели во *.pgm* формат, да имаат иста димензија и форма на квадрат. Тоа е најдобро така бидејќи на тој начин би имале најголема точност и прецизност на нашиот модел. Меѓутоа во мојот случај ќе користиме поразличен пристап кон генерирањето на потребните податоци.

3.8.4.3. Претпроцесирање на податоците за тренирање

Базата на слики за тренирање која ќе ја користам во проектот се наоѓа во рамките на самиот проект и претставува директориум од фолдери кадешто во секој фолдер посебно има различни слики од една личност. Фолдерите во директориумот се именувани со *I1, I2, I3 … In* соодветно, а сликите од личностите се именувани со броевите од *1 … n*. Во директориумот можеме да креираме број на фолдери и број на слики по наша желба.

При моето истражување внимателно ги одбирав сликите за тренирање, одбирав тие да бидат со добар квалитет, светлина, да имаат исправен лик, а од друга страна внимавав да вклучам повеќе различни, карактеристични слики како што се на пример: слики со очила за гледање, насмеано лице, озбилно лице, налутено лице, без и со шминка, со пуштена или собрана коса итн. Разновидноста на сликите му помага на моделот за тренирање да научи поопширно од она што е многу вообичаено за едно лице и на тој начин го правиме моделот поспособен да може да го препознае лицето. Она што е сепак најбитно, а го споменавме и

претходно е дека со користење на оваа библиотека и Haar каскадите не можеме да очекуваме добри или воопшто било какви резултати ако лицето на сликата е искривено, ротирано, во неправилна положба, половично или во профил. Сето тоа понатаму ќе го покажам и докажам во рамките на апликацијата.



Слика 17: Организација на податоците за тренирање

Figure 17: Dataset for training

Сите алгоритми за препознавање на лице земаат два параметри во нивниот *train()* метод: низа од слики и низа од лабели. Лабелите претставуваат ознаки за лицата како уникатно ID и со тоа знаеме кое точно лице е препознаено од нашата база.

3.8.4.4. Препознавање на лице на слика

Втората функционалност во рамките на апликацијата е препознавање на лицето од сликата. За таа цел откако ќе ја прикачиме сликата треба да кликнеме на копчето **Recognize** за да ја повикаме скриптата ***recognize.py*** која го извршува препознавањето. Во оваа скрипта исто така мора да ги има потребните библиотеки кои секогаш се вчитуваат на почетокот на скриптата.

```
subjects = ["", "Darko", "Angela", ”Luka”, ”Aleksandar”]

def prepare_training_data(data_folder_path):
    dirs = os.listdir(data_folder_path)

    faces = []
    labels = []
    for dir_name in dirs:
        if not dir_name.startswith("l"):
            continue
        label = int(dir_name.replace("l", ""))
        subject_dir_path = data_folder_path + "/" + dir_name
        subject_images_names = os.listdir(subject_dir_path)
        for image_name in subject_images_names:
            if image_name.startswith("."):
                continue
            image_path = subject_dir_path + "/" + image_name
            image = cv2.imread(image_path)
            face, rect = detect_facerec(image)

             if face is not None:
                faces.append(face)
                labels.append(label)

    return faces, labels
```

Препознавањето бара извршување на повеќе функции врз сликата, затоа за да ни биде појасно, детално ќе разгледаме некои линии код од оваа скрипта.

Најпрво, забележуваме дека имаме декларирано низа со имиња како членови и тоа претставуваат имињата на личностите чии слики имаме зачувано во нашата база. Имињата ќе ги искористиме за да кога ќе се препознае личноста, името да се испечати дека станува збор за тој човек. Така секогаш кога скриптата ќе препознае некое ID, ние ќе го испечатиме соодветното име во низата *subjects*, наместо ID-то.

Наредно во овој код е преставена функцијата ***prepare_training_data*** која служи за тренирање на моделот врз сликите од нашиот фолдер. Кога ја повикуваме оваа функција ние во *data_folder_path* ја запишуваме патеката на фолдерот **training_data**. Функцијата има за задача да ги вчита сликите од директориумот прегледувајќи ги фолдер по фолдер, а понатаму за секоја слика врши детекција на лицето. На излез враќа низа од *faces* и *labels* т.е. број на лицата детектирани од фолдерите.

```
def face_rec(request) :

   if request.FILES.get("image", None) is None:
       return render(request, 'uploadrec.html')



   test_img1 = cv2.imdecode(np.fromstring(request.FILES['image'].read(), np.uint8), cv2.IMREAD_UNCHANGED)

   predicted_img1 = predict(test_img1)

   cv2.imwrite('D:/WORK/DPOpenCV/faces/static/recognizedfaces/doneimage.png',predicted_img1)
   return render(request, 'recognized.html')
```

Кога сликата ќе се испрати од формата на почетната страна преку методот **POST** таа се декодира и се претвора во низа од осум битни цели броеви

форматирано преку каналите BGR и потоа се вршат потребните функции кои ќе ги објаснам во понатамошното излагање.

Ја имаме функцијата ***predict*** која ја повикуваме на нашата прикачена слика од **html** formata. Можеме да забележиме дека од оваа функција се отповикува функцијата за тренирање бидејќи пред моделот да даде прогноза тој мора да ги прегледа сите слики од базата. Понатаму се креира променливата *face_recognizer* и таа е објект од класата **cv2.face.LBPHFaceRecognizer_create()** кој се повикува да тренира над *faces* и низата *labels*.

```
def predict(test_img):

    img=test_img

    faces, labels = prepare_training_data('D:/WORK/DPOpenCV/faces/training_data/')

    face_recognizer = cv2.face.LBPHFaceRecognizer_create()

    face_recognizer.train(faces, np.array(labels))

    face, rect = detect_facerec(img)

    label, confidence = face_recognizer.predict(face)
    label_text = subjects[label]

    draw_rectangle(img, rect)
    draw_text(img, label_text, rect[0], rect[1]-5)

    return img
```

Во *face* се зачувуваат координатите на правоаголникот што треба да се исцрта околу детектираното лице, во *label* се добива индексот на лицето кое е

препознато, а во *confidence* се зачувува број што всушност покажува со колкава грешка е добиен резултатот од препознавањето. Во најдобар случај вредноста на атрибутот *confidence* треба да изнесува 0.0, а било која вредност над 50.0 би значело дека препознавањето е извршено со голема грешка и тој резултат најверојатно не е точен. Ако тоа лице носи лабела со вредност 1 тогаш ќе го добие првото име од низата, ако носи лабела со вредност 2 ќе го добие второто име итн. На крај се исцртува правоаголникот со ***draw_rectangle*** и се испишува името на личноста со ***draw_text***.

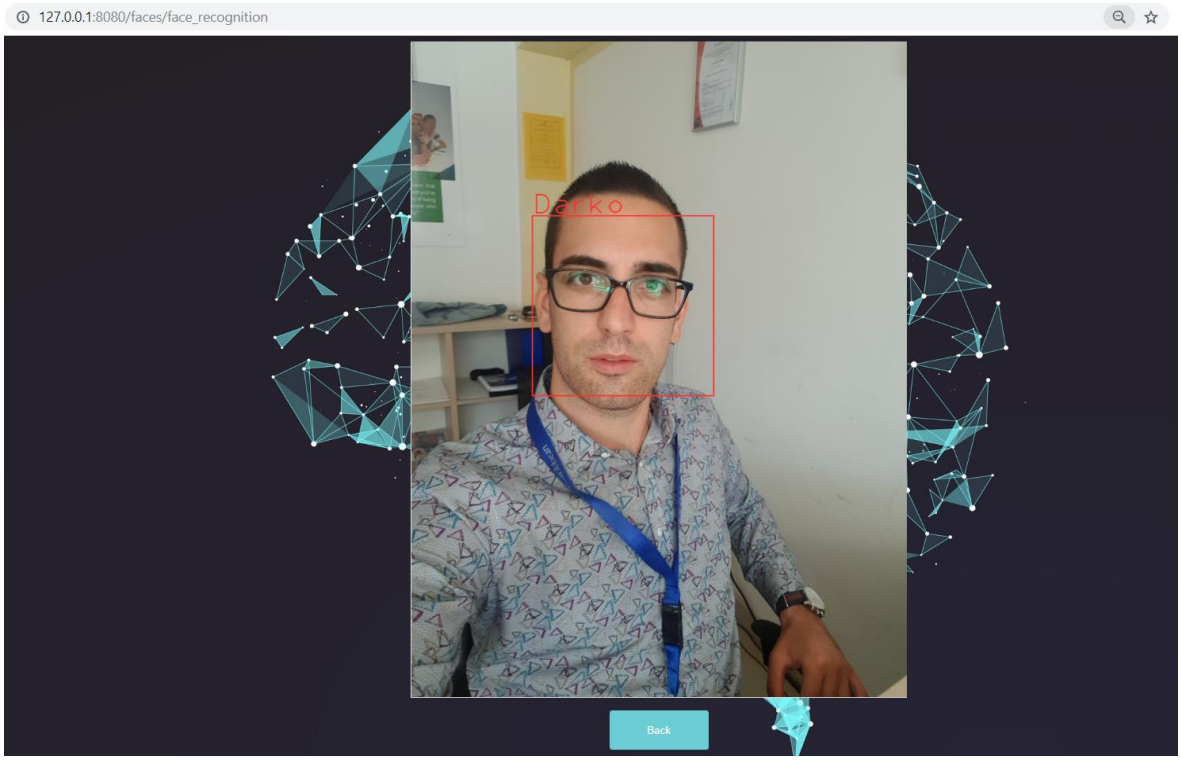
```
def draw_rectangle(img, rect):
    (x, y, w, h) = rect
    cv2.rectangle(img, (x, y), (x+w, y+h), (51, 51, 255), 6)

def draw_text(img, text, x, y):
    cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_PLAIN, 10, (51, 51, 255), 6)
```

На крај, сликата се покажува во рамките на апликацијата и на неа се гледа детектираното лице со црвен правоаголник, и напишаното име со црвени букви.

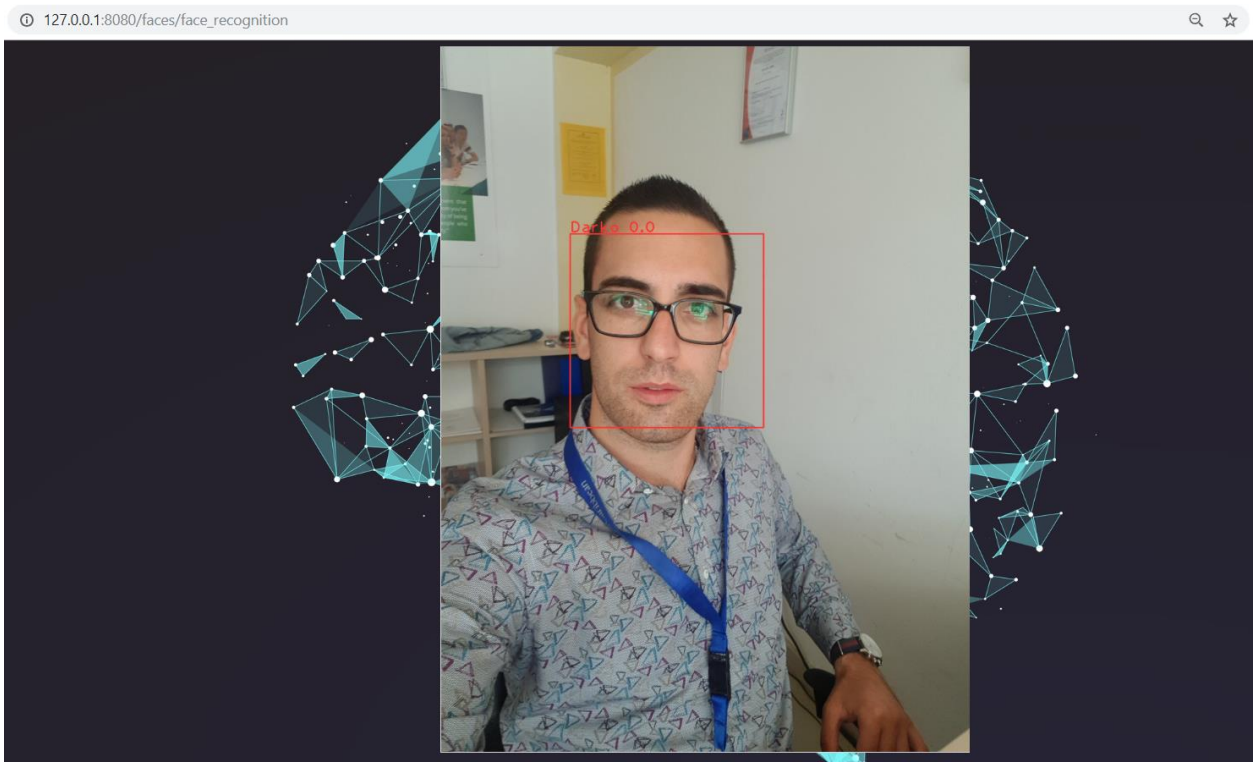
По наша желба би можеле веднаш до името да ја испечатиме и вредноста на грешката за да имаме претстава дали добиеното име е точно со голема точност или се случило грешка препознавање. Тоа би го направиле со повторно повикување на функцијата ***draw_text*** на следниот начин:

```
draw_text(img, str(confidence), rect[1], rect[1]-5)
```



Слика 18: Приказ на сликата по извршување на функцијата за препознавање на лице

Figure 18: Image representation after running function for face recognition



Слика 19: Лицето “Darko” е препознато со 0.0 грешка на оваа слика

Figure 19: Person “Darko” is recognized with 0.0 failure on this picture

4. РЕЗУЛТАТИ

Моето истражување во областа на длабокото учење и неговите методи за препознавање на лице ми даде до заклучок дека денешните архитектури и методи и те како се успешни во извршувањето на нивната функционалност и намена. Благодарение на архитектурите и библиотеките направени за тие цели, јас успешно создадов своја апликација за детекција и препознавање на лице која ми даде добри резултати, а од друга страна добивме резултати кои нѐ подучуваат какви слики е пожелно да се избегнуваат во овие функции со цел да се намали нивото на грешка и да се извади максималното од методите.

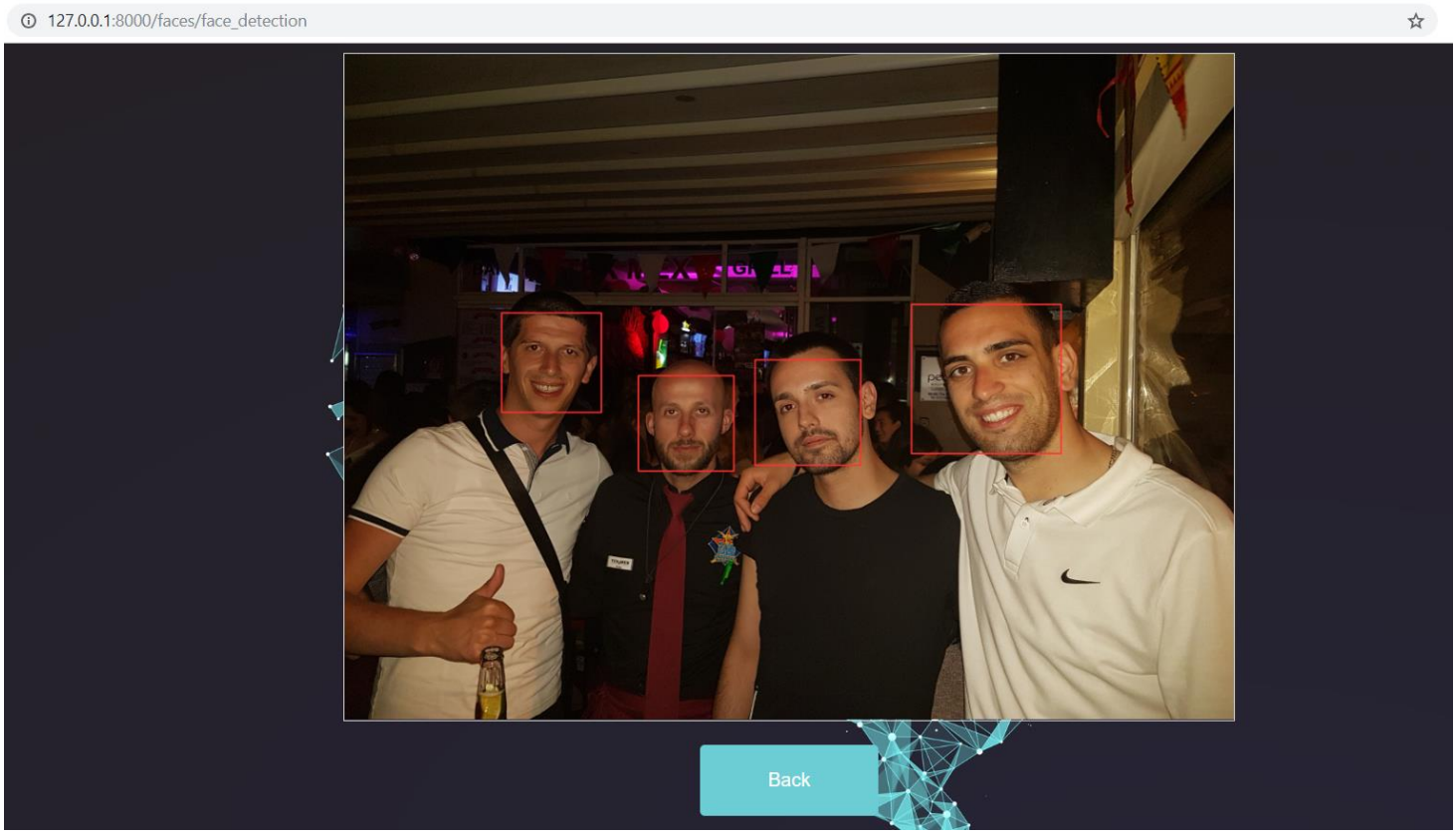
Библиотеката OpenCV која ја користев во моето истражување и апликација нуди навистина лесен пристап кон разбирање на методите на длабоко учење. Преку нејзините Haar и LBP каскадни класификации успеав да допрам во длабочините на обработката на слики, да дознаам кои елементи од сликата се битни за нејзиното процесирање, како се врши целиот процес на форматирање за да се дојде до слика на која може да се работи, кои надворешни фактори можат да влијаат врз резултатите, како да се избегнат несаканите грешки и како да се подобри функционалноста на апликацијата.

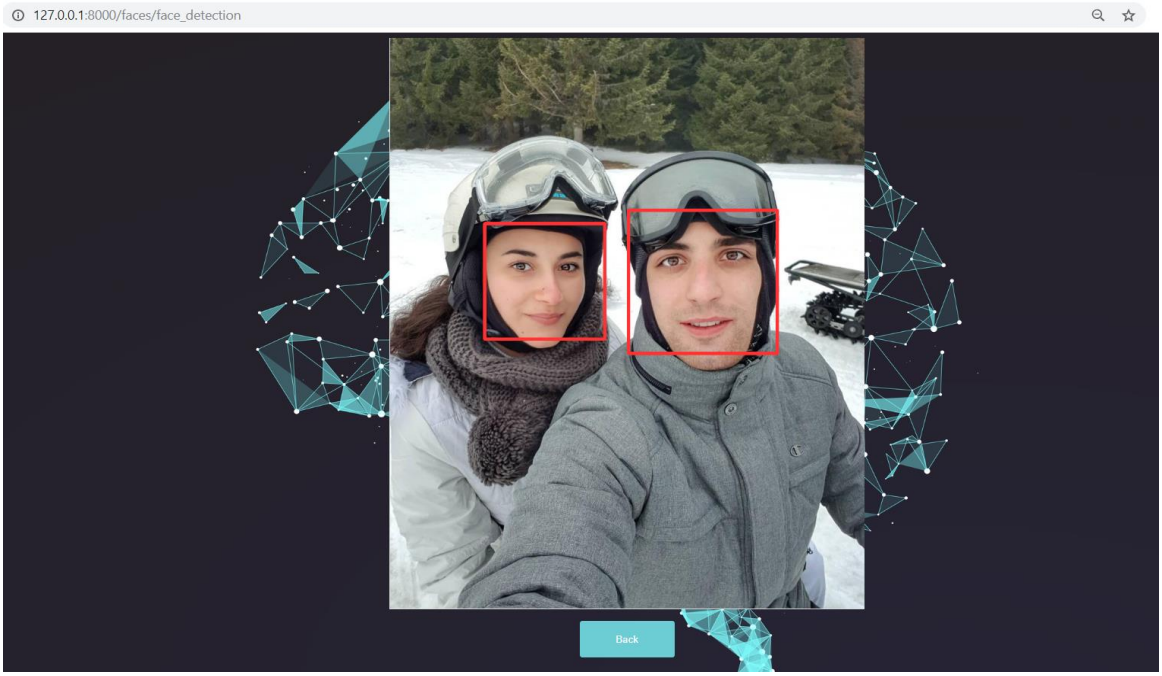
Во наредниот дел од трудот ќе ги покажам моите тестирања на апликацијата со разновиден избор на слики врз двете основни функционалности:

1. Детекција на лице или повеќе лица во слика
2. Препознавање на лицето од сликата

Детекцијата на лицата во сликите е речиси безгрешна со тоа што алгоритмот секогаш успева да го детектира лицето или лицата од сликата. Сепак, како што споменавме и претходно, алгоримот на Haar каскадите не е имун на ротирани лица, лица кои се во профил, накривено поставени и сл. Лицата кои беа така поставени во сликите не беа детектирани, додека во секој друг случај добив позитивни резултати.

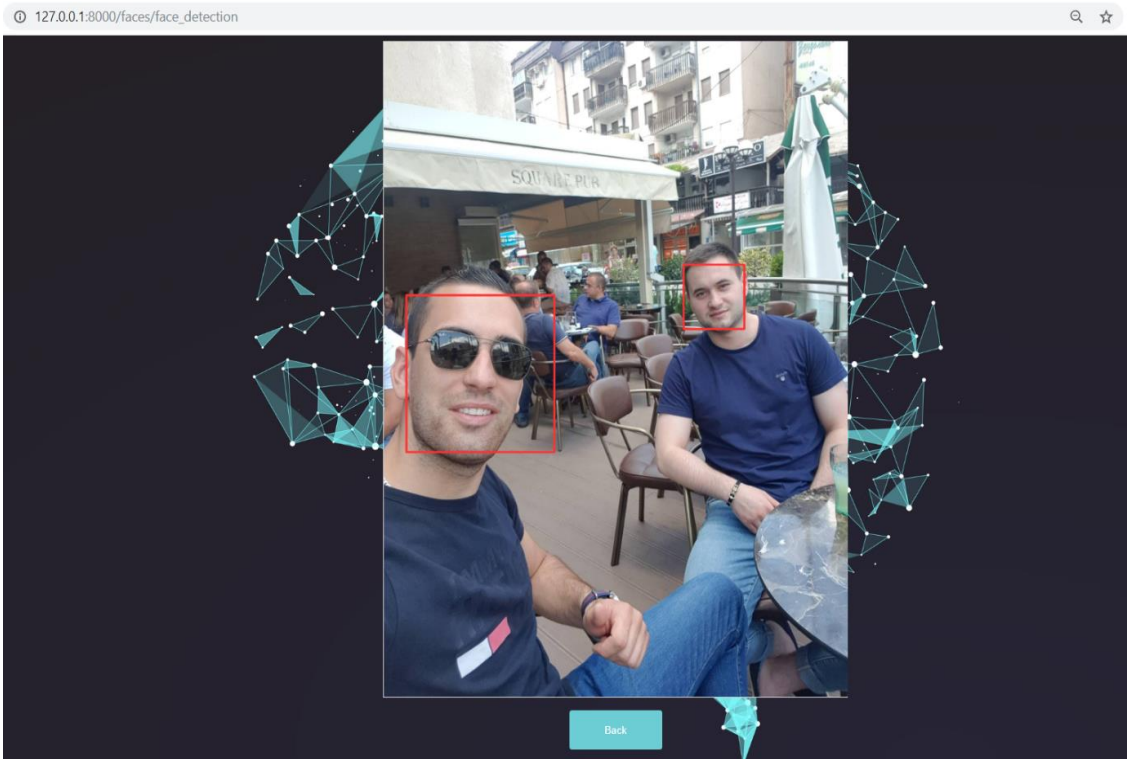
4.1. Визуелни резултати од детекцијата на лице





Слика 20 и 21: Апликацијата детектира повеќе лица

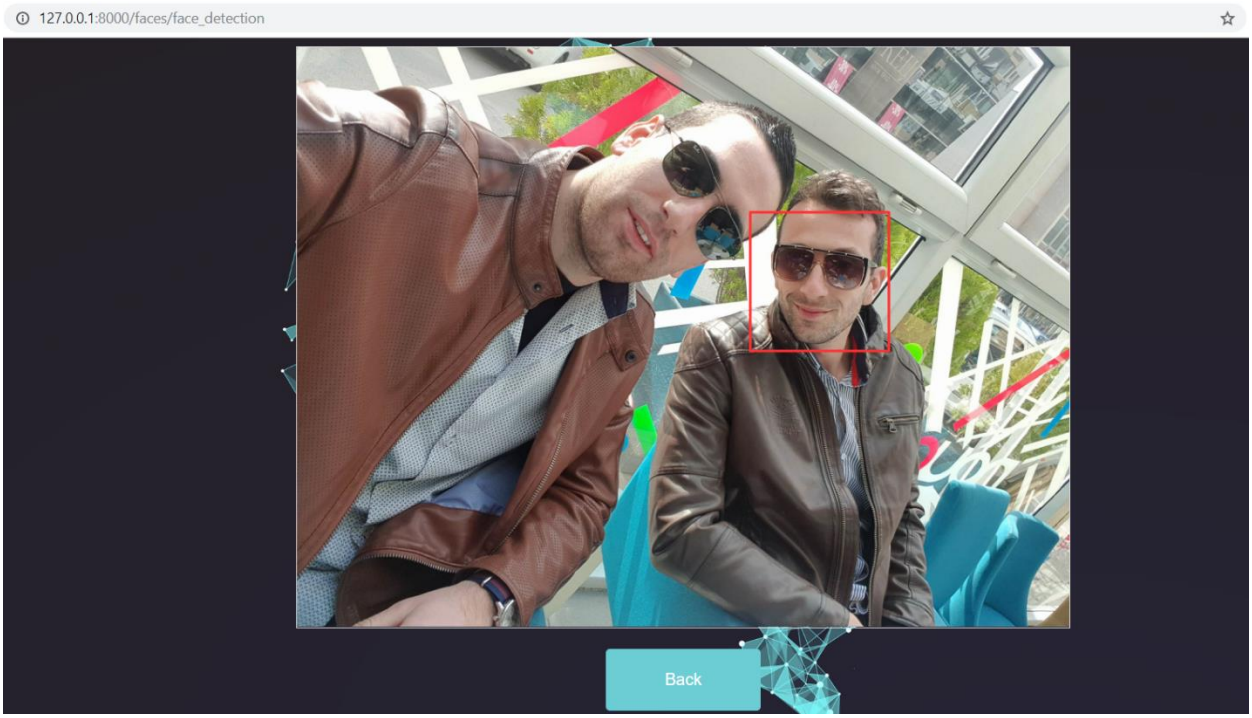
Figure 20 and 21: Multiple faces detected with the application



Слика 22: Апликацијата детектира лице со очила за сонце

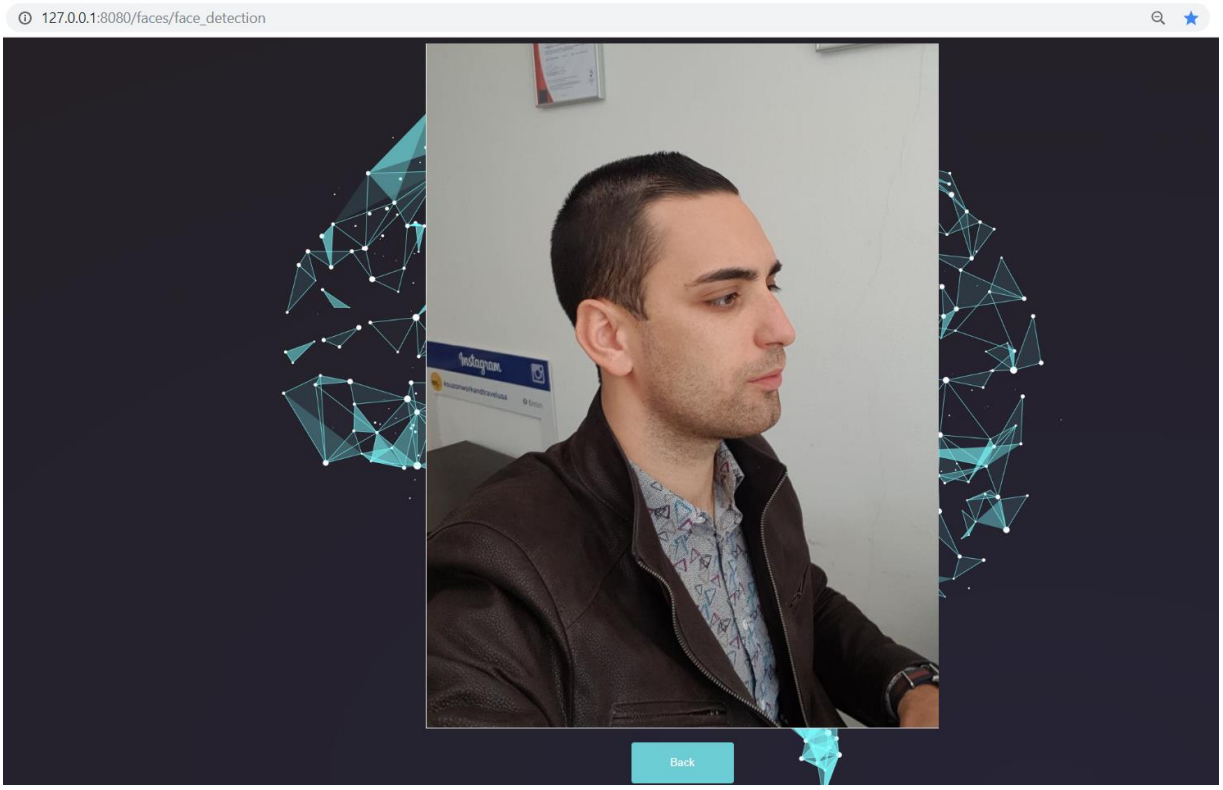
Figure 22: Application detects face wearing sunglasses

Haar каскадите не можат да го детектираат лицето кое е извртено под поголем агол, додека другото лице иако носи очила за сонце успешно е детектирано.



Слика 23: Апликацијата не го детектира ротираното лице

Figure 23: Application does not detect rotated face

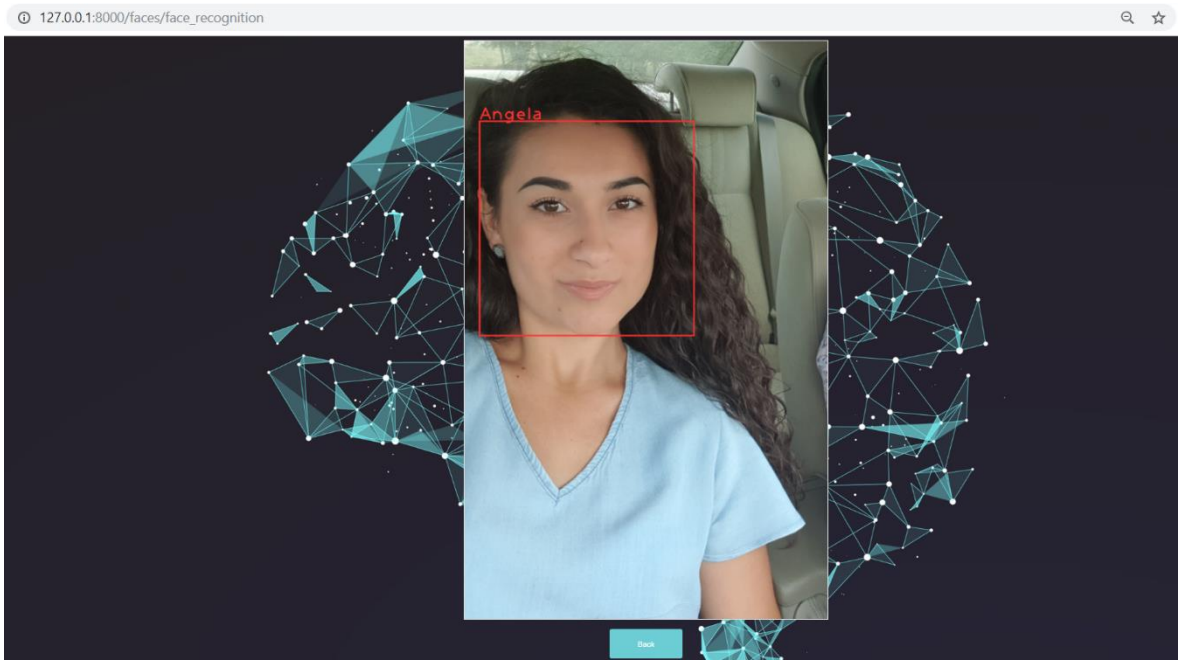


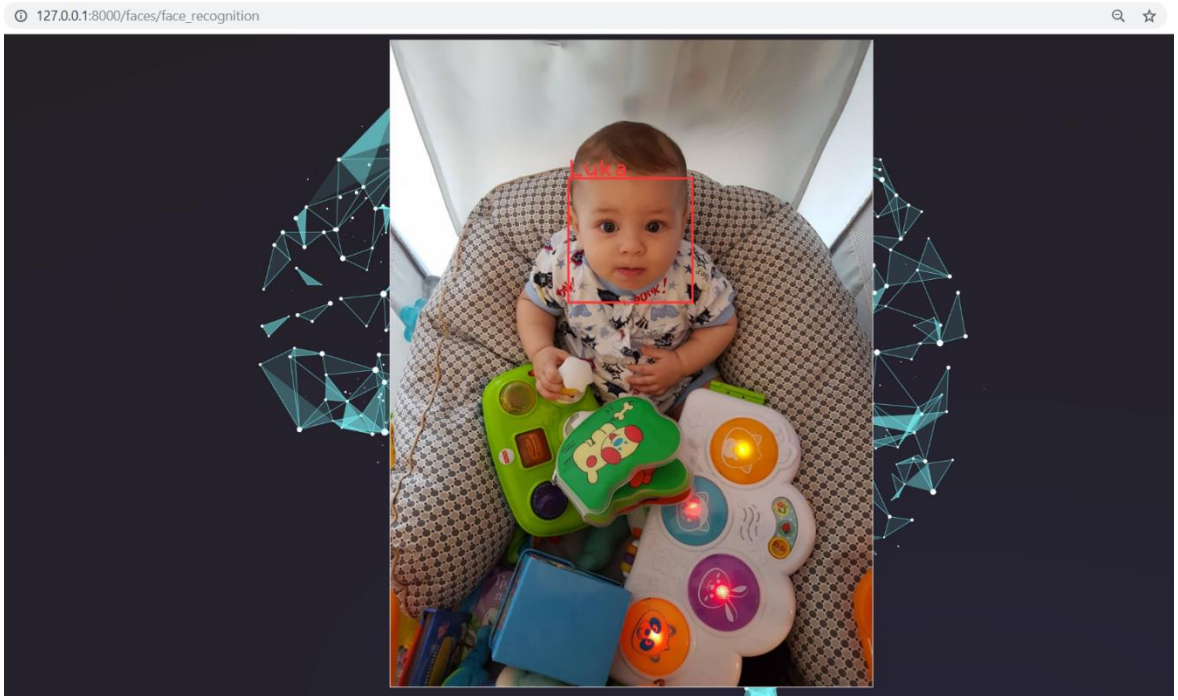
Слика 24: Пример за недетектирано лице

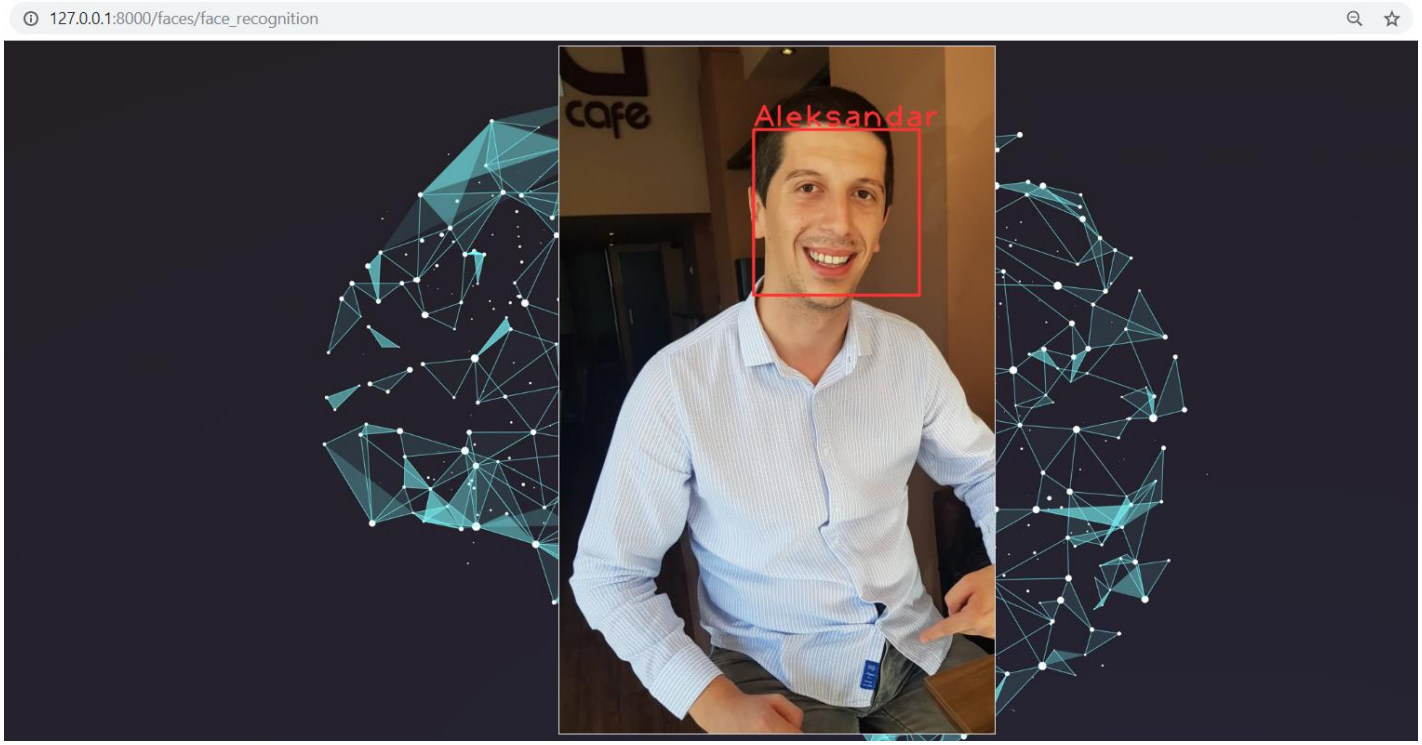
Figure 24: Example for undetected face

Лицето на оваа слика не беше детектирано бидејќи тоа е завртено во профил и каскадите не можат да детектираат дека има лице на сликата.

4.2. Визуелни резултати од препознавање на лице

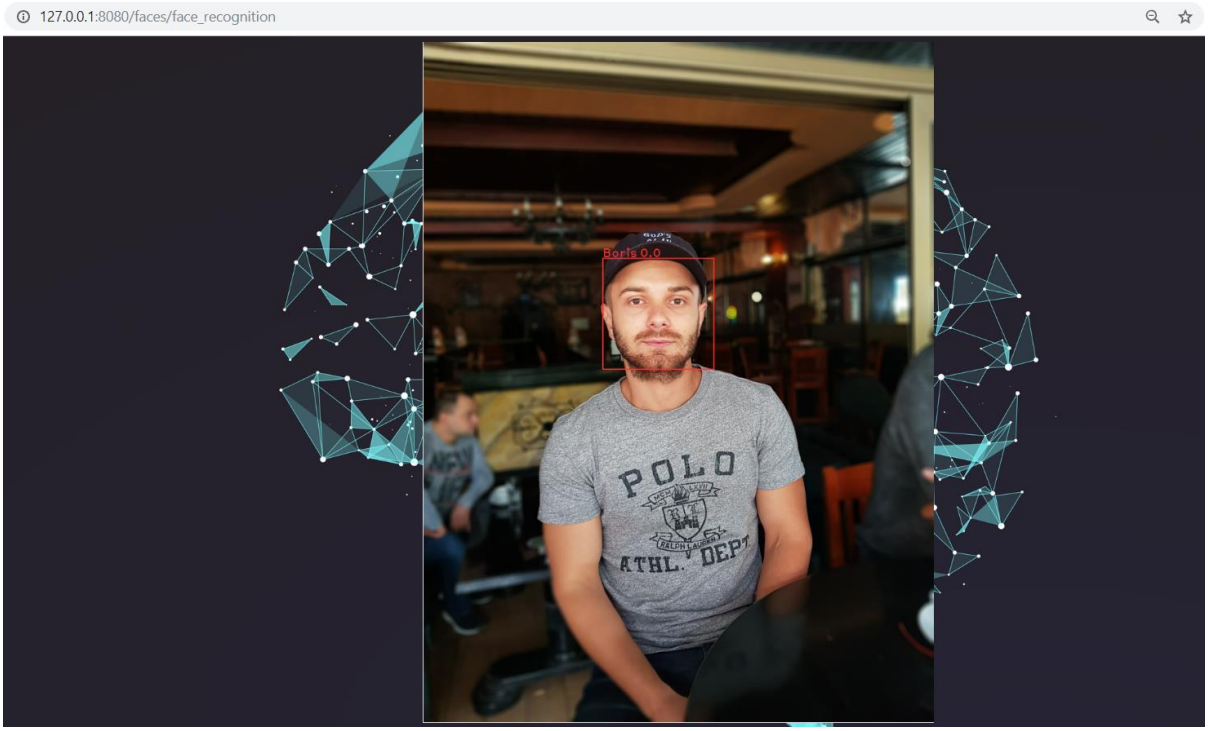






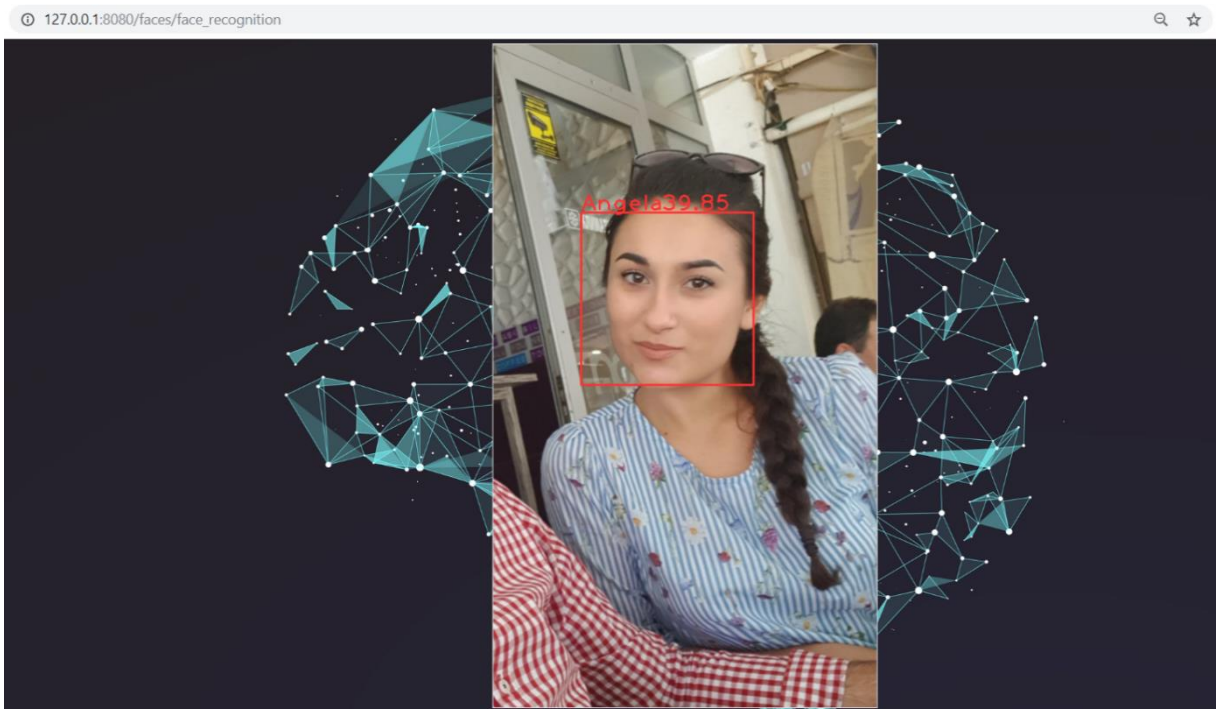
Слика 25, 26 и 27: Апликацијата препознава дека ликот на сликата е најсоодветен со лицата во базата кои се водат под тоа име

Figure 25, 26 and 27: Application recognizes that the face from the picture is most compatible with the person from the database named with that name



Слика 28: Лицето „Boris” е препознато со 0.0 грешка на оваа слика

Figure 28: Person “Boris” is recognized with 0.0 failure on this picture



Слика 29: Лицето “Angela” е препознато со 39,85 грешка на оваа слика

Figure 29: Person “Angela” is recognized with 39.85 failure on this picture

5. ЗАКЛУЧОК

Сликите не секогаш се гледале како огромен извор на податоци како што претставуваат сега. Некогашните проблематики поврзани со препознавањето на објекти или лица долги години биле невозможни, тешки за решавање и секогаш барале присуство од човечки фактор да го каже крајниот резултат.

Иако постоела идејата, потребен бил и технолошко технички развој во хардверски решенија за да се овозможи брзо процесирање на податоците бидејќи станувало збор за високо слојни архитектури кои бараат голема процесирачка моќ и време. Тука дошле на сцена GPU уредите со нивните моќни јадра, па така тој развојот на технологиите и техниките успеал да го оживее тоа поле на предизвици овозможувајќи методи на длабоко учење кои сликата ја упростуваат до ниво на пиксел и каскадно ги вадат сите обележја кои се “кријат” во неа.

Можеме да заклучиме дека конволуциските мрежи (CNN) како метод за длабоко учење се покажале како најдобри во употребата за класификација на слики и денес се најупотребувани за тие цели. Нивните длабоки архитектури успеваат најдобро да ги извадат обележјата на сликата.

Водејќи се по мојата инспирација, ги истражував начините на кои се детектира или препознава лице во слика задржувајќи се на познатата библиотека OpenCV и нејзините алгоритми и методи. Имплементирањето на библиотеката OpenCV во мојата апликација ни помага да заклучиме како работат каскадните класификации и како квалитетот на сликите и светлината влијаат врз добивањето на резултатот. Преку вршење на тестирање со различни слики добив резултати за успешна и неуспешна детекција, како и успешно и неуспешно препознавње. Слабата страна на каскадните класификации е извртеното или ротирано лице, а кај методите за препознавање заклучивме дека најбитна е разновидноста и квалитетот на сликите кои ќе ги приложиме во нашата база со слики, како и сликата која ја приложуваме кон алгоритмот да работи со неа.

Ваквата апликација би имала голема примена во сектори како што се јавната администрација, амбасади, гранични премини, аеродроми и сл. Би можело да се искористи и во помали и поголеми фирми за логирање, отклучување или најавување.

6. КОРИСТЕНА ЛИТЕРАТУРА (REFERENCES)

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Convolutional Neural Networsk University of Toronto

[2] Baba Abedallatif (2017), A literature study of Deep Learning and its application in Digital Image Processing

[3] Elza John (2017), MATLAB EXPO 2017 Simplifying Image Processing and Computer Vision Application Development

[4] Gary Bradski, Adrian Kaehler (2008), Learning OpenCV

[5] Greg Schoeninger, Image Classification with Deep Neural Networks

[6] Jan Erik Solem (2012), Programming Computer Vision with Python

[7] Jiajun Wu, Yinan Yu, Chang Huang, Kai Yu, Deep Multiple Istance Learning for Image Classification and Auto-Annotation

[8] Joseph Howse (2013), OpenCV Computer Vision with Python

[9] Joe Minichino, Joseph Howse (2015), Learning OpenCV 3 Computer Vision with Python Second Edition

[10] Larry Brown (2015), Deep Learning for Image Classification

[11] Marc Aurelio Ranzato (2015), Image Classification with Deep Learning

[12] Muhammad Imran Razzak, Saeeda Naz, Ahmad Zaib, Deep Learning for Medical Image Processing: Overview, Chalenges and Future

[13] Rolan Memisevic (2016), Deep Learning in Image Processing

[14] Tariq Rashid – Make your own Neural Network

[15] University of Montreal, LISA LAB(2015), Deep Learning Tutorial Release 0.1

[16] The OpenCV Tutorials Release 2.4.13.7, July 19, 2018

Дарко Златев

Дигитална обработка на слики со користење на методи базирани на длабоко учење

Универзитет ,,Гоце Делчев'' - Штип