Универзитет Св. Кирил и Методиј - Скопје

Факултет за информатички науки и компјутерско инженерство

Документација за семинарска работа

Предмет: Дигитално процесирање на слика

Имплементација на Инстаграм филтри

Ментор: проф. д-р Ивица Димитровски

Студент: Елена Кецкароска

Индекс: 201036

Скопје, јуни 2022

Содржина

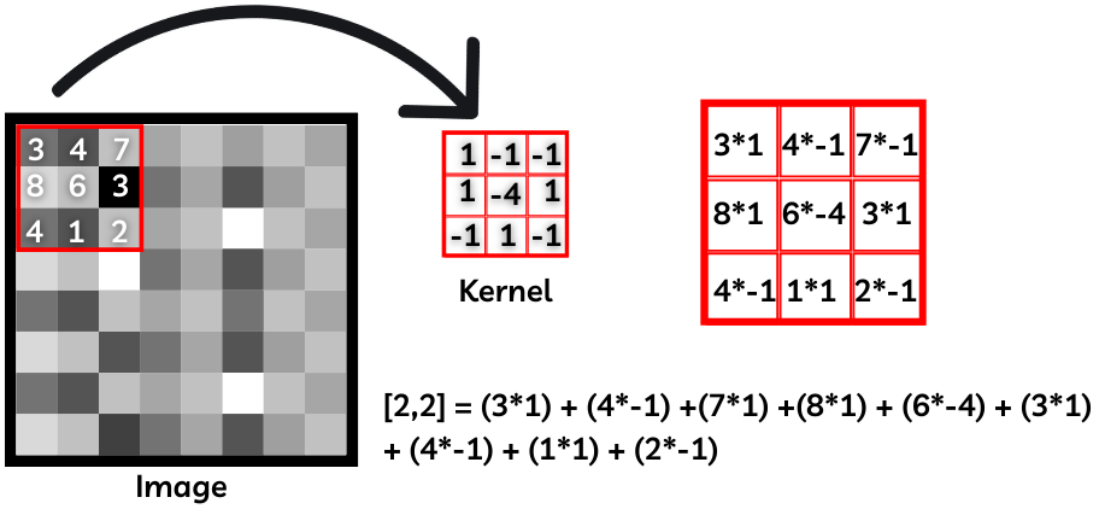
Апстракт

Инстаграм апликацијата се разви во една од најпознатите социјални мрежи. Една од главните одлики на истата е тоа што овозможува лесно едитирање на сликите и сликање со филтри. Темата на овој проект е всушност имплементација на неколку од тие филтри со помош на програмскиот јазик Python и OpenCV библиотеката, кои нудат голем број на готови функции со чија помош се постигнува изглед на филтрите идентичен како оној на Инстаграм апликацијата. Бројот на ваквите филтри е доста голем, но логиката за имплементација претставува игра со боите, каналите, осветлувањето, контрастот и други карактеристики на сликата. Па така, намерата на овој проект е да се претстават неколку имплементации и да се открие како всушност Инстаграм апликацијата ни нуди вакви поволности.

Sharpen filter

Конвулуција на слики

Конвулуцијата на слики претставува обработување на сите пиксели од сликата. Истата се базира на идејата да се користи еден kernel и со итерација на влезната слика за сите нејзини пиксели, да се генерира нова излезна слика. Во текот на една итерација се зема матрица со вредности од изворната слика со иста големина како на кернелот, се множи секој елемент од матрицата со кернелот соодветно, потоа се собираат вредностите и добиените податоци се нормализираат за да се претстават како пиксели во добиената слика.



Слика 1: множење матрица од слика со кернел;

Како проблем се појавува тоа што добиената слика е со помала големина од изворната, бидејќи пикселите на самиот раб на сликата не можат да бидат опфатени при изборот зошто не соодветствуваат со големина на кернелот. Решенија за истото се: да се изостават пикселите на работ, да се направи пресликување (mirroring), нивните места да се пополнат со боја и слично.

Имплементација на Sharpen филтер

Овој ефект лесно се добива со помош на функцијата filter2D од OpenCv. Како аргументи оваа функција ги прима влезната слика, кернелот и длабочина, која означува тип на податоци во сликата.

За точниот ефект потребно е вредностите на кернелот да бидат избрани како на слика 2. Постојат повеќе однапред дефинирани кернели кои придонесуваат за имплементацијата на добро познати ефекти на слики.

-1	-1	-1
-1	9	-1
-1	-1	-1

Слика 2: кернел за Sharpen филтер;

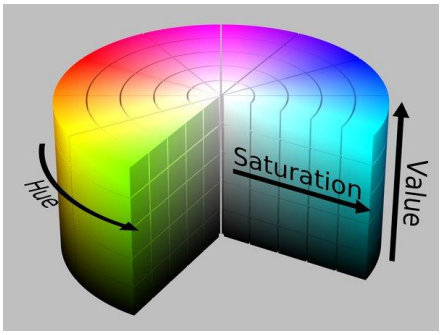
Слика 3: резултат од Sharpen филтер;

Gingham филтер

HSV – color space

HSV како простор на бои означува три вредности кои ја формираат бојата: hue, saturation, value. HSV – просторот на бои го моделира начинот на кој различни бои се комбинираат заедно за да создадат нова боја во реалниот свет, со димензија на вредноста (Value), која наликува на различни количини на црна или бела боја во смесата (на пр. за да се создаде „светло црвена“, црвениот пигмент може да се меша со бела боја). Целосно заситените бои се претставени со вредност од 0 или 1 што одговара на целосно црно или бело, соодветно.

Hue кореспорендира со позицијата на бојата на тркалото на слика 4 и прима вредности од 0 до 1, со тоа што како бројот се ближи кон 1, боите се менуваат од црвена кон портокалова, потоа зелена, сина па се до црна. Често вредноста за бојата се претставува со степен за да се симулира кругот како на сликата подолу. Сатурацијата се однесува на количеството на боја, односно со голема сатурација бојата нема да содржи бела компонента во себе, додека со мала сатурација ќе биде во сив тон. Исто како и бојата, прима вредности од 0 до 1. Конечно вредноста – value ја опишува колку бојата ќе биде темна или светла.



Слика 4: приказ на HSV просторот на бои;

Премин од BGR – color space во HSV – color space

Функцијата cvtColor од библиотеката OpenCv нуди лесен премин од еден во друг простор на бои, но тоа што се случува во позадина со сликата е најпрво одделување на трите компоненти или т.н. канали на пикселот, blue, green и red и потоа според формулите се добиваат компонентите на hue, saturation и value.

$R = red, G = green, B = blue$

$R' = R/255$

$G' = G/255$

$B' = B/255$

$C_{max} = \max(R', G', B')$

$C_{min} = \min(R', G', B')$

$\Delta = C_{max} - C_{min}$

Секоја од вредностите се дели со 255 поради тоа што компонентите на HSV примаат вредности помеѓу 0 и 1. По оваа операција на ниво на пиксел, според формула се пресметува соодветниот број за бојата, сатурацијата и вредноста.

$$\boldsymbol{H} = \begin{cases} 0°, & \Delta = 0 \\ 60° * (\frac{G' - B'}{\Delta} mod 6), & C_{max} = R' \\ 60° * (\frac{B' - R'}{\Delta} + 2), & C_{max} = G' \\ 60° * (\frac{R' - G'}{\Delta} + 4), & C_{max} = B' \end{cases}$$

$$\boldsymbol{S} = \begin{cases} 0, & C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & C_{max} \neq 0 \end{cases}$$

$\boldsymbol{V} = C_{max}$

$S = saturation$ (сатурација)

$H = hue$(боја)

$V = value$ (вредност)

Пример за конверзија помеѓу просторите на бои: (128, 0, 128) се вредностите R, G и B за еден пиксел.

$$R' = \frac{128}{255} = 0.5, G' = \frac{0}{255} = 0, B' = \frac{128}{255} = 0.5$$

$$C_{max} = 0.5, C_{min} = 0, \Delta = 0.5 - 0 = 0.5$$

$$\boldsymbol{H} = \left(\frac{0.5-0}{0.5} + 4\right) * 60° = 300° \quad \boldsymbol{S} = \frac{0.5}{0.5} = 1 * 100\% = 100\% \quad \boldsymbol{V} = 0.5 * 100\% = 50\%$$

Имплементација на Gingham филтер

Убавината на овој филтер е тоа што ги истакнува и засилува боите на сликата, што се постигнува со игра на компонентите во HSV просторот на бои. Имено влезната слика се претвара од BGR (blue, green, red) во овој простор на бои и се одделуваат компонентите за да може да се манипулира со истите. Потоа сатурацијата се намалува за 90 %, hue се зголемува за 110 %, за истата вредност се зголемува и контрастот, додека светлината се намалува за 80 %. Овие вредности се точно избрани за да се добие ефектот на сликата и се применуваат врз секој пиксел на влезната слика, со што треба да се внимава на опсегот да остане во соодветната граница.

Слика 5: резултат од Gignham Filter;

Sepia филтер

Конвулуција на слики

Sepia филтерот исто така претсавува обработување на влезна слика пиксел по пиксел односно конволуција на слики. Наместо со готовата функција filter2D од

OpenCV, истиот ефект може да се постигне и со рачно изминување на пикселите од сликите.

BGR и RGB – color spaces

OpenCV библиотеката, која ја користиме за вчитување на сликите, го користи BGR (Blue, Green, Red) како простор на бои, односно секој пиксел од сликата е составен од трите канали: син, зелен и црвен со соодветни вредности од 0 до 255 за секој од истите. Со нивна комбинација е претставена бојата која човекот ја гледа. Sepia филтерот всушност е добиен со игра на вредностите за секој од каналите и затоа е важно тие да бидат правилно одвоени. Имено кога ќе имаме за обработка еден пиксел тој ќе биде претставен преку три вредности, на пр.: [0 94 144] и тука значајно е првиот канал да го третираме како син, додека последниот како црвен и да им ги промениме местата при имплементацијата.

RGB просторот на бои се користи кај matplotlib библиотеката која ја користам за претставување на добиената слика. Овој простор на бои е ист како BGR, но со таа разлика што тука првиот канал кај пикселот е црвениот, а псоледниот е син. Пред да се предаде сликата за приказ, истата треба да се конвертира од BGR во RGB просторот на бои со помош на готовата функција од OpenCV: cvtColor(image, cv2.COLOR_BGR2RGB).

Имплементација на Sepia филтер

Овој филтер се постигнува со два циклуси и со чија помош се добива вредноста за пикселот. Потоа за секој пиксел ги одделуваме трите канали на боја и секој од нив го обработуваме на следниот начин:

- новодобиени син канал: 0.272 * red + 0.534 * green + 0.131 * blue;
- новодобиен црвен канал: 0.393 * red + 0.769 * green + 0.189 * blue;
- новодобиени зелен канал: 0.349 * red + 0.686 * green + 0.168 * blue.

Потоа се врши проверка за секој каналите да не ја надмине максималната вредност од 255 и се поставува истата на 255 доколку се случило истото. Потоа формираме пиксел со трите нови вредности за каналите и ја повторуваме постапката одново додека не се изминат сите пиксели од влезната слика. Крајниот резултат е слика која има црвенкаст и потемен изглед во споредба со влезната, како на слика 7.

0.272	0.534	0.131
0.349	0.686	0.168
0.393	0.769	0.189

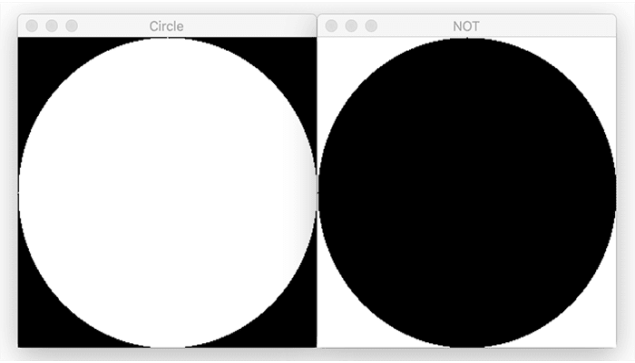
Слика 6: матрица со вредности за пресметување на каналите на секој пиксел;

Слика 7: резултат од Sepia Filter;

Invert филтер

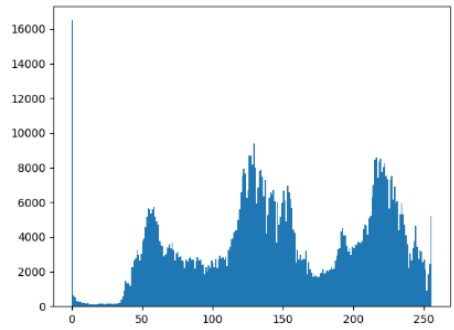
Bitwise not

Bitwise not во дигиталното процесирање на слика има исто значење како и во математиката, односно негација, се што има вредност точно (1 или true) по оваа операција станува неточно (0 или false) и обратно. Оваа операција се извршува над влезната низа од пиксели на сликата претворајќи ги белите пиксели во црни и обратно.

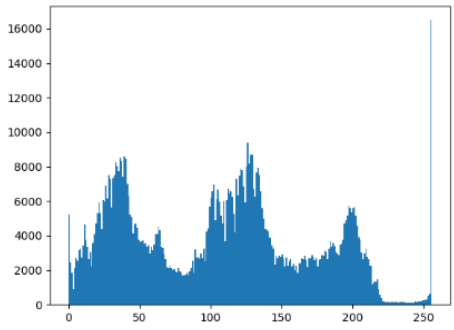


Слика 8: резултат од bitwise not операцијата над црно - бела слика;

Bitwise not операцијата е посложена кај сликите во боја. Тогаш пресметувањето се сведува на одземање на вредноста на пикселот од 255 за секој од трите канали. За појасно гледиште кон тоа што навистина се случува е претставено на хистограмите подолу, слика 9 и слика 10. Претставени се два хистпграми, еден за влезната слика пред и втиориот за состојбата после негацијата. Јасно се забележува дека и двата графици се симетрични според вертикалната оска, со што се докажа негацијата.



Слика 9: хистограм на влезна слика во боја;



Слика 10: хистограм на излезна слика во боја;

Имплементација на Invert филтер

Фотографиите врз кои е применет овој филтер добиваат доста интересен изглед и често се користи за добивање слика каков би бил човекот со бела коса. Имплементацијата е доста едноставна бидејќи OpenCV библиотеката има готова функција (bitwise_not(image)), која прима само еден аргумент, а тоа е сликата. Ја претвора во влезна низа од пиксели и изминувајќи ги, за секој еден од нив се врши операцијата.

Слика 11: резултат од Invert Filter;

Детекција на објекти во слика и примена во филтри

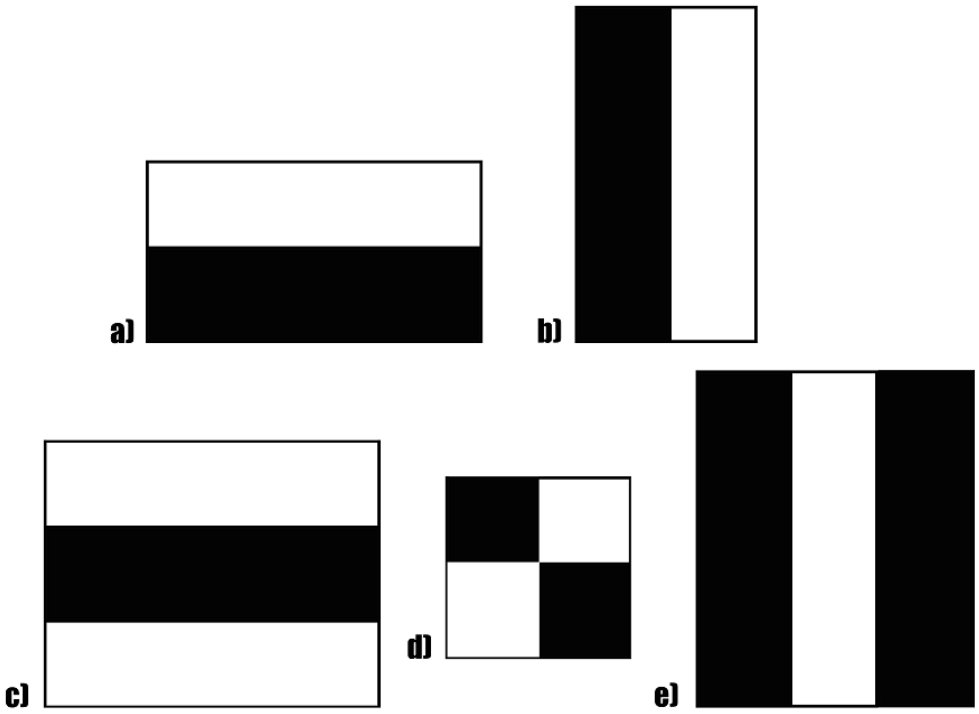
Детекција на лице е широко популарен предмет на дискусија со огромен опсег на можности. Современите паметни телефони и лаптопи доаѓаат со вградени софтвери за детекција на лице, кои можат да го автентицираат идентитетот на корисникот. Постојат бројни апликации кои можат да снимаат, детектираат и обработуваат лице во реално време, можат да ја идентификуваат возраста и полот на корисникот, а исто така можат да применат некои навистина интересни филтри. Списокот не е ограничен само на овие мобилни апликации, бидејќи Face Detection исто така има широк опсег на апликации за надзор, безбедност и биометрика. Но, потеклото на неговите успешни приказни датира од 2001 година, кога Виола и Џонс ја предложија првата рамка за откривање објекти за откривање лица во реално време во видео снимки.

Техниката за откривање лица на Виола и Џонс, популарно позната како Haar Cascade, е предложена долго пред да започне ерата на длабоко учење (deep learning). Но, иако е постара, сепак претставува одлична трхника во споредба со моќните модели што можат да се изградат со современите техники за длабоко учење. Алгоритмот сè уште се користи речиси насекаде. Има целосно имплементирани модели достапни на GitHub, кои се доста брзи и прецизни.

Вуди Бледшу, Хелен Чан Волф и Чарлс Бисон беа првите кои го направија детекција на лице на компјутер уште во 1960-тите. За едно лице мораше рачно да се одредат координатите на цртите на лицето, како што се центрите на зеницата, внатрешниот и надворешниот агол на очите. Координатите беа искористени за пресметување на 20 растојанија, вклучувајќи ја и ширината на устата и на очите. Човекот

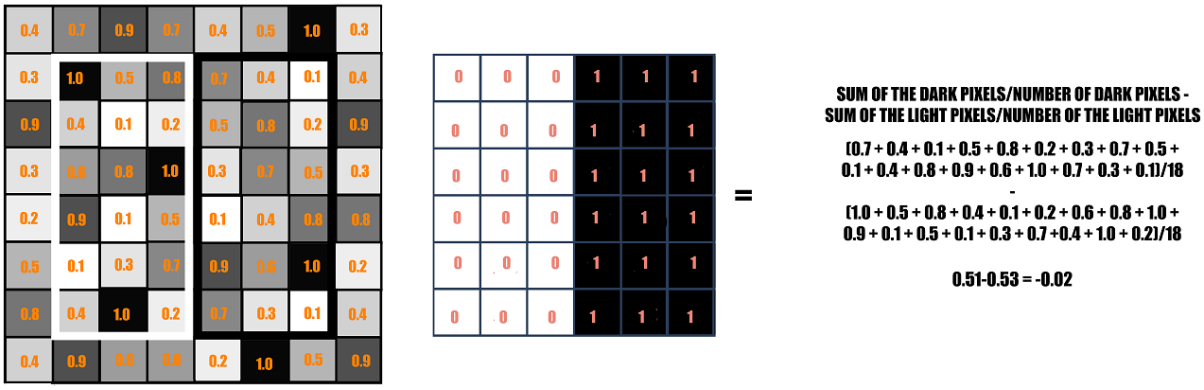
може да обработи околу 40 слики на час на овој начин и така да изгради база на податоци за пресметаните растојанија. Компјутерот потоа автоматски ќе ги спореди растојанијата за секоја фотографија, ќе ја пресмета разликата помеѓу растојанијата и ќе ги врати затворените записи како можна совпаѓање.

Haar Cascade е алгоритам за откривање објекти што се користи за идентификување лица во слика или видео во реално време. Алгоритмот користи карактеристики за откривање раб или линија предложени од Виола и Џонс во нивниот истражувачки труд „Брзо откривање објекти со помош на засилена каскада од едноставни карактеристики“ објавен во 2001 година. На алгоритмот му се дадени многу позитивни слики кои се состојат од лица, и многу негативни слики кои не се состојат од никакво лице за тренирање на нив. Како што споменав претходно, овој модел е достапен во складиштето на GitHub.



Слика 12: Haar карактеристики;

Складиштето ги има моделите складирани во XML-датотеки и може да се читаат со методите OpenCV. Тие вклучуваат модели за откривање лице, детекција на очи, откривање на горниот или долниот дел од телото, откривање на регистарски таблички итн. Првиот придонес во истражувањето беше воведувањето на Haar карактеристиките. Овие карактеристики на слика 12 го олеснуваат откривањето на рабовите или линиите на сликата или бирањето области каде што има ненадејна промена во интензитетот на пикселите.



Слика 13: пример за пресметка на Haar карактеристика;

На слика 13 е прикажан пример за пресметка на вредноста на Haar од правоаголна слика. Потемните области во карактеристиката Haar се пиксели со вредности 1, а посветлите области се пиксели со вредности 0. Секоја од нив е одговорен за откривање на една одредена карактеристика на сликата. Како раб, линија или која било структура на сликата каде што има ненадејна промена на интензитетот. На пр. на сликата погоре, карактеристиката Haar може да открие вертикален раб со потемни пиксели на десната страна и полесни пиксели на левата страна.

Целта овде е да се дознае збирот на сите пиксели на сликата што лежат во потемната област на карактеристиката Haar и збирот на сите пиксели на сликата што лежат во посветлата област на карактеристиката Haar, а потоа да се пресмета нивната разлика. Детекцијата се врши така што ако сликата има раб што ги одвојува темните пиксели десно и светлите пиксели лево, тогаш вредноста на хаар ќе биде поблиску до 1. Тоа значи, дека е откриен раб ако вредноста на Haar е поблиску до 1. Во примерот погоре, нема раб бидејќи вредноста на хаар е далеку од 1.

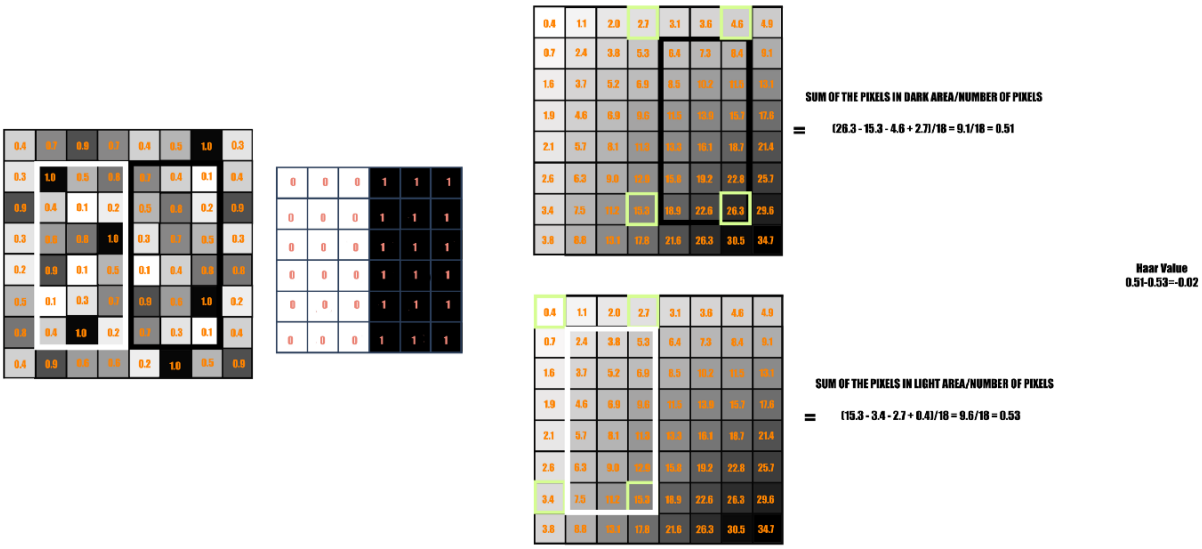
Ова е само една претстава на одредена Haar карактеристика што одвојува вертикален раб. Останиот голем број на Haar карактеристики детектираат рабови во други насоки и какви било други структури на сликата. Важно е да се спомене дека за да открие раб било каде на сликата, карактеристиката Haar треба да ја измине целата слика.

Избраната Haar карактеристика постојано изминува од горниот лев агол на сликата до долниот десен агол, за да ја бара одредената карактеристика. Но за сето ова да биде прецизно, изминувањето на Haar карактеристиката се врши за секој пиксел на сликата и со различни големини на карактеристиката.

Во зависност од карактеристиката што ја бара секој од нив, тие се широко класифицирани во три категории. Првиот сет од две правоаголни карактеристики се одговорни за откривање на рабовите во хоризонтална или вертикална насока. Вториот сет од три правоаголни карактеристики се одговорни за откривање дали има полесен регион опкружен со потемни области од двете страни или обратно. Третиот сет од четири правоаголни карактеристики се одговорни за откривање на промената на интензитетот на пикселите низ дијагоналите.

Бидејќи оваа операција побарува големи пресметки и перформанси на машината на која се изведува, предложен е нов концепт познат како Интегрална слика. Интегралната слика се пресметува од оригиналната слика на таков начин што секој

пиксел во интегралната е збирот на сите пиксели што лежат лево и десно, и горе и долу во оригиналната слика. Пресметката на пиксел во интегралната слика е прикажан на сликата подолу. Последниот пиксел во долниот десен агол на интегралната слика ќе биде збир од сите пиксели во оригиналната слика.



Слика 14: пресметување на пиксел во интегралната слика;

AdaBoost

Во основа, тоа што се случува е дека има збир на карактеристики што би доловувале одредени структури на лицето, како што се веѓите или мостот помеѓу двете очи, или усните итн. Но, првично сетот на функции не бил ограничен на ова. Сетот на функции броел приближно 180.000, кои се намалија на 6000 со помош на AdaBoost техниката.

Повеќето од овие карактеристики не работат добро или ќе бидат ирелевантни за цртите на лицето, бидејќи ќе бидат премногу случајни за да се најде нешто што би било корисно. Така, овде беше потребна техника за селекција на карактеристики, за да се избере подмножество карактеристики од огромниот сет, кој не само што би ги одбрал карактеристиките што имаат подобри перформанси од другите, туку и ќе ги елиминираат ирелевантните. Техника за зајакнување наречена AdaBoost, во која секоја од овие 180.000 карактеристики беа применети на сликите посебно за да се создадат Weak Learners. Некои од нив произведоа ниски стапки на грешки бидејќи ги одвојуваа позитивните од негативните слики подобро од другите, додека некои не. Овие Weak Learners се дизајнирани на таков начин што погрешно би класифицирале само минимален број слики. Тие можат да работат подобро од само случајно погодување. Со оваа техника, нивниот последен сет на функции се намали на вкупно 6000 од нив.

Attentional Cascade

Подмножеството од сите 6000 карактеристики повторно ќе се активира на сликите – модели за да открие дали има црта на лицето или не. Сега авторите земаа

стандардна големина на прозорец од 24x24 во која ќе работи откривањето на карактеристиките. Но ова повторно побарува големи перформанси.

За да го поедностават ова, тие предложија друга техника наречена „Attentional Cascade“. Идејата зад ова е дека не треба сите функции да се извршуваат на секој прозорец. Ако некоја карактеристика не успее на одреден прозорец, тогаш можеме да кажеме дека цртите на лицето не се присутни таму. Оттука, можеме да преминеме на следните прозорци каде што може да има црти на лицето.

Карактеристиките се применуваат на сликите во фази. Фазите на почетокот содржат поедноставни карактеристики, во споредба со карактеристиките во подоцнежните фази, кои се доволно сложени за да се пронајдат и брчките на лицето. Ако почетната фаза не открие ништо на прозорецот, тогаш се отфрла самиот прозорец од преостанатиот процес и се преминува на следниот прозорец. На овој начин ќе се заштеди многу време за обработка, бидејќи ирелевантните прозорци нема да се обработуваат во повеќето фази.

Обработката во втората фаза би започнала, само кога карактеристиките во првата фаза ќе бидат откриени на сликата. Процесот продолжува вака, т.е. ако помине една фаза, прозорецот се пренесува на следната фаза, ако не успее тогаш прозорецот се отфрла.

Целеата теорија е доста комплексна, но самото користење на истата во OpenCV библиотеката е многу едноставно. Имено со помош на функцијата CascadeClassifier, која како аргуметн го прима xml Haar cascade фајлот и функцијата detectMultiScale со која се добиваат сите детектирани лица или насмевки во зависност од потребата според предадената слика во gray формат. Исто така како аргумент прима и scale factor, параметар што означува колку се намалува сликата во итерациите за детектирање на потребниот објект во сликата и minNeighbors, параметар кој означува колку минимум пиксели соседи треба да има секој од правоаголниците (Haar карактеристиката) за да биде задржан.

Имплементација на Halloween филтер

Овој филтер е имплементиран во видео во реално време и за таа цел видео-камерата издвојува слики на кои се имплементира филтерот. Сето ова е овозможено со циклус за повторување и фунцкијата webcam.read(), која лови слики од видеото, па затоа поради брзината се добива чувство дека е во реално време.

Имплементацијата на овој филтер започнува со креирање на маска од сликата – слика 15 која ја користиме за крајниот изглед на филтерот. Имено со cv2.threshold() се добива слика 16 и со bitwise_not операцијата ја добиваме слика 17.

Слика 15: маска за филтер;

Слика 16: резултат на threshold;

Слика 17: резултат на bitwise_not операцијата;

Претходно споменатите детектирани објекти со detectMultiScale функцијата, сега се изминуваат и секој таков објект (лице) посебно се обработува за Halloween маската да се постави над лицето. Поконкретно функцијата враќа четири вредности, од кои две се: координата горе лево на правоаголникот во кој се наоѓа лицето и координата долу лево на истиот правоаголник, ширината и висината на правоаголникот. Потоа се прилагодува висината и ширината на Halloween маската според лицето, односно се употребуваат следниве формули:

$$Halloween_{mask_{width}} = 1.5 * Halloween_{OriginalMask_{width}};$$

$$Halloween_{mask_{height}} = Halloween_{mask_{width}} * \frac{Halloween_{OriginalMask_{height}}}{Halloween_{OriginalMask_{width}}}.$$

Следно се пресметуваат координатите на Halloween маската според лицето кое е детектирано:

$$Halloween_{mask_{x1}} = face_{x2} - \left(\frac{face_{width}}{2}\right) - \left(\frac{Halloween_{mask_{width}}}{2}\right);$$

$$Halloween_{mask_{x2}} = Halloween_{mask_{x1}} + Halloween_{mask_{width}};$$

$$Halloween_{mask_{y1}} = face_{y1} - (face_{height} * 1.25);$$

$$Halloween_{mask_{y2}} = Halloween_{mask_{y1}} + Halloween_{height}.$$

каде *x1* е координатата во левиот горен агол, *x2* е координата во десниот горен агол, а *y1* и *y2* се координататите лево и десно долу соодветно. Потоа се врши проверка дали добиените координати сеуште соодвествуваат со големината на сликата која ја обработуваме.

Имплементацијата понатака продолжува со промена на големината на Halloween маската и нејзините варијанти threshold и bitwise_not според потребите на лицето, со цел да се добие подобар и поприроден изглед.

Според горенаведените координати се издвојува делот од сликата за обработка и со bitwise_and(roi,roi,mask=mask), каде roi е делот од сликата и mask е Halloween маската (слика 16) се добива слика 18. Со истата функција применета на оригиналната слика за Halloween маската (слика 15) и слика 17 се издвојува само делот со капата, кој го добиваме во боја како оригиналот но сега е поставен над црна позадина како на слика 19. И на крај со обичен збир на овие две слики со помош на cv2.add() се добива слика 20, која ја поставуваме на соодветните координати на лицето во оригиналната слика. Оваа постапка се повторува за сите детектирани лица во една слика и се добива краен резултат како на слика 21 и слика 22.

Слика 18: резултат на bitwise_and над слика од лице и слика 16;

Слика 19: резултат на bitwise_and над слика 15 и слика 17;

Слика 20: резултат од add функцијата на слика 18 и слика 19;

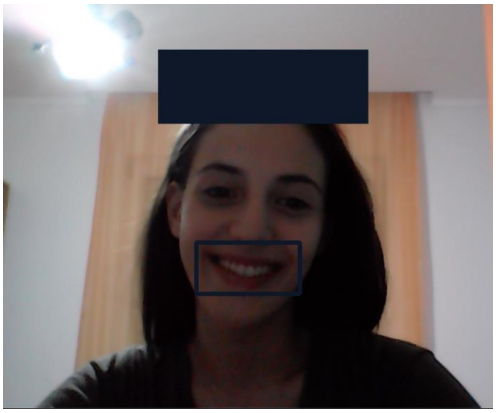
Слика 21: поставување на слика 20 во излезната слика – резултат со едно детектирано лице;

Слика 22: резултат со две детектирани лица;

Имплементација на филтер за оценка на насмевката

Овој филтер има цел да даде оценка за детектирана насмевка во видео во реално време. Детекцијата се постигнува со detectMultiScale функцијата, но овој пат се користи два пати. Прво се детектираат лица, а потоа насмевки на лицата.

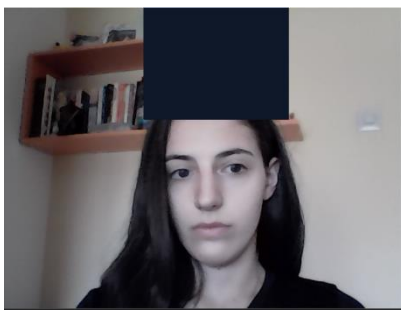
За секое пронајдено лице се користат координатите со тоа што се исцртува правоаголник над лицето со функцијата cv2.rectangle(im, (x, y), (x + (w + 20), y + (h - 300)), (40, 24, 14), -1), каде *im* е сликата на која се црта, *x,y,w,h* се вредностите, а останатите параметри се боја и начин на исцртување. Потоа се бара насмевка на лицето и идентично ги користиме координатите на насмевката за да нацртаме правоаголник околу насмевката. Како резултат до тука се добива слика 23.



Слика 23: резултат на детекција на лице и насмевка и исцртување;

Преостанатиот дел од имплементацијата е да се испише вредност за насмевката. random.randint(0, 100) функцијата избира случаен број од 0 до 100, потоа избираме фонт и боја за текстот и со функцијата cv2.putText() на која ги предаваме: сликата на која испишуваме, слика 23, текстот, односно бројот, координатите од правоагилникот исцртан над лицето и параметри за изгледот на текстот.

Но за да постигне ефект дека бројот се избира неколку секунди, користиме глобална променлива која на почеток е поставена на 0 и ја зголемуваме за некоја вредност, на пр. 5 и постојано се испишува текст за оценката. Кога оваа променлива ќе ја надмине вредноста 100, тогаш повторно испишуваме текст и ја повикуваме функцијата time.sleep(7) за да се добие ефект дека насмевката добила оценка по изборот.



Слика 24: резултат кога не е детектирана насмевка;

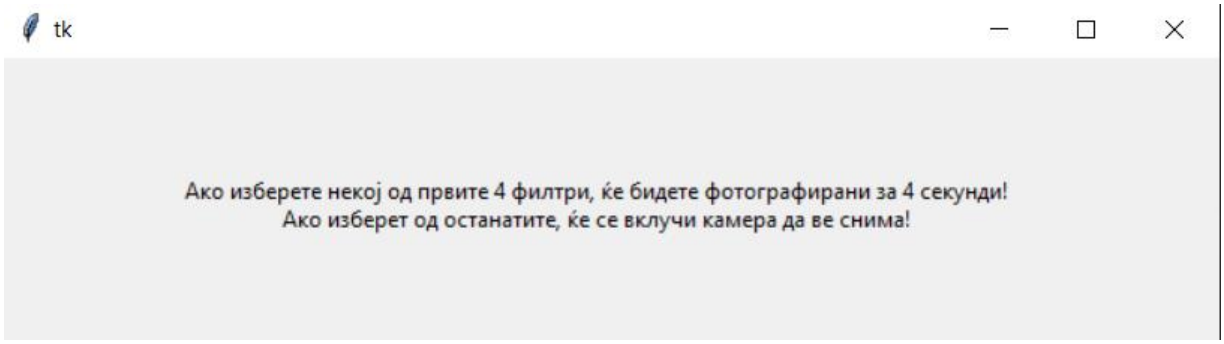
Слика 25: резултат од детектирана и оценета насмевка;

Слика 26: резултат од детектирана и оценета насмевка;

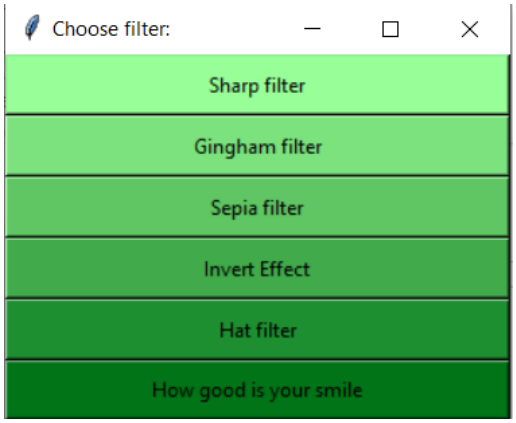
Слика 24 претставува тест за алгоритамот за детекција, односно се детектира лице, се исцртува правоаголникот над главата, но бидејќи не детектира насмевка не се испишува ништо за оценката. Останатите: слика 25 и слика 26 се резултат на детектирана насмевка.

Интерфејс на демо апликацијата

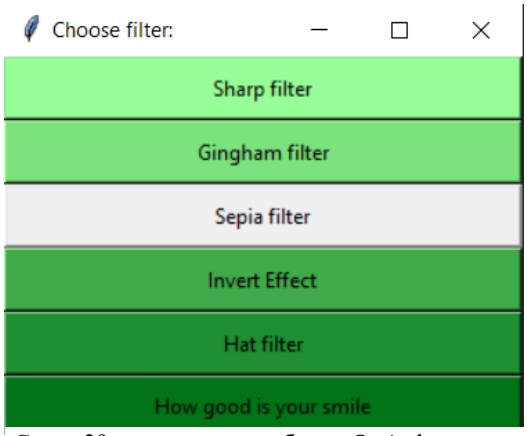
Интерфејсот на апликацијата е како на слика 27, што ја претставува замислата дека ако се избере некој од следните филтри: Sharp, Gingham, Sepia или Invert, ќе се вклучи камера да фотографира и потоа на таа фотографија се имплементира филтерот избран од понудени опции како на слика 28.



Слика 27: известување за корисникот;



Слика 28: избор на филтри;



Слика 29: изглед кога е избран е Sepia филтерот;

При избор од останатите две можности се вклучува камерата и таа снима, односно испраќа многу фотогтафии во краток временски период на кои се имплементира филтерот и веднаш се прикажува, со што добиваме чувство дека сето тоа се случува во реално време.

Имплементацијата за известувањето е овозможена од tkinter библиотеката. *Label(top, text="Ако изберете некој од првите 4 филтри, ќе бидете фотографирани за*

4 секунди!" "\nАко изберет од останатите, ќе се вклучи камера да ве снима!") – функцијата прима прозорец *top* на кој се испишува текстот означен во *text.*

Слично е овозможен и изборот за филтри со функцијата *Radiobutton()*, која освен прозорец, текст и аргументи за изглед, прима и *command=selection* аргумент. *Selection* е функција која има централна улога во апликацијата, односно раководи кој филтер да се примени според вредноста на копчето кое е избрано од корисникот како на слика 29.

Користена литература

1. Borcan Marius, „Python OpenCV: building Instagram - like Image filters “, Towards Data Science, march, 2020. <https://towardsdatascience.com/python-opencv-building-instagram-like-image-filters-5c482c1c5079>
2. Brownlee Jason, „How to Perform Face Detection with Deep Learning“, Machine Learning Mastery, august, 2020. <https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/>
3. Mikan Jelena, „Remake of some filters on Instagram made with Python, OpenCV and Matplotlib“, Big Meter, april, 2019. <https://www.big-meter.com/opensource/en/5fec9fc796d0a3317c75c0f3.html>
4. Bhavsar Kushal, „Insta_flters_with_python”, Github, february, 2019. <https://github.com/Spidy20/Insta_flters_with_python/blob/master/video_detection.py>
5. Roy Harshit, How to Create Instagram filter using Python, Youtube, june, 2021.<https://www.youtube.com/watch?v=1lfnpKRI8q4>