

# Дигитално процесирање на слика

Тема:

Детекција и анонимизација на лице

## Содржина

Вовед .....	3
Детекција на лице .....	3
Haar Cascade алгоритмот .....	3
DNN Face Detector from DNN module in OpenCV .....	7
Анонимизација на лице .....	8
Gaussian Blur .....	9
Box blur (Average blur) .....	12
Споредба помеѓу двата модела за детекција на лице .....	13
Користена литература .....	18

## Вовед

Во денешниот дигитален свет, **детекцијата на лице** игра важна улога во различни области како што се надзорот и безбедноста, биометриска автентикација, како и приватноста на корисниците. **Машинското учење** и **длабокото учење** развија моќни алатки за автоматска детекција на лица во различни услови и амбиенти. Овие технологии овозможуваат создавање на модели кои можат да препознаат лица со висока прецизност, како и да ги процесираат за различни цели, меѓу кои е и **анонимизацијата на лице**, која е од клучно значење за заштита на приватноста на луѓето.

## Детекција на лице

**Детекцијата на лице** претставува процес на откривање на човечки лица во дигитални слики или видеа, без оглед на нивната позициј и осветлување. За оваа цел, користат два главни пристапа:

1. **Машинско учење:** Класични методи, како што е **Haar Cascade**, користат претходно дефинирани карактеристики за препознавање на лица. Овие методи се брзи, но имаат ограничена прецизност во слики со комплексна содржина.
2. **Длабоко учење:** Со појавата на **длабоки невронски мрежи**, особено со модели како **Res10\_300x300\_SSD**, детекцијата на лице стана значително поефикасна, точна и поотпорна на варијабилности, како што се промени во осветлувањето, аголот на лицето, и други фактори.

## Haar Cascade алгоритмот

Haar Cascade е алгоритам заснован на машинско учење кој се користи за откривање објекти на слики или видеа. Алгоритмот е развиен од Пол Виола и Мајкл Џонс во 2001 година и се заснова на каскаден пристап кој комбинира едноставни шаблони. За побрзо време на процесирање сликата се прикажува во црно-бела репрезентација, бидејќи вредноста на секој пиксел е цел број. Haar Cascades е дел од OpenCV библиотеката во Python.

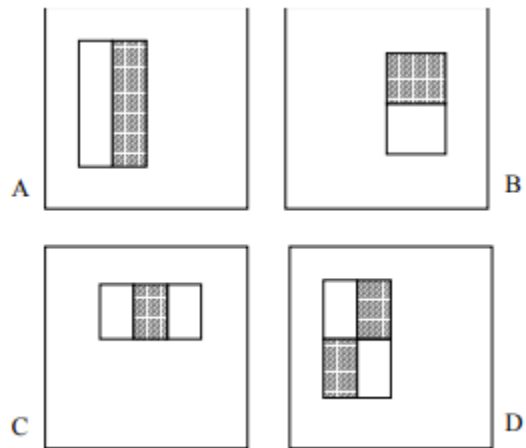
### Haar карактеристики

Алгоритмот користи множество од **Haar карактеристики (шаблони)** кои се едноставни форми (правоаголници) за да ја анализира вредноста на интензитетот на пикселите во дадена област на сликата. Овие карактеристики претставуваат разлика помеѓу сумата на пикселите во осветлени и темни делови од правоаголни области, на пример:

*Edge features* – Карактеристика со два правоаголници каде што едниот дел е црн, а другиот бел, за разликување светли и темни региони. На пример, при откривање лице, заедничка карактеристика на рабовите може да го спореди интензитетот на осветленост на челото (посветло) со пределот на очите (потемни).

*Line features* – Карактеристика со три правоаголници со темна линија помеѓу две светли лении за идентификација на хоризонтални и вертикални линии. На пример, тие може да помогнат да се најде носот или веѓите.

*Four-rectangle features* – Тие откриваат посложени обрасци, како агли или спојки. Ова може да помогне да се детектираат области како носот или устата каде што се спојуваат неколку карактеристики.



Карактеристики со два правоаголници се прикажани кај (А) и (Б).

Сликата (В) покажува карактеристика со три правоаголници.

Сликата (D) покажува карактеристика со четири правоаголници.

Овие карактеристики се применуваат на различни делови од сликата и се пресметува разликата во интензитет помеѓу светлите и темните региони.

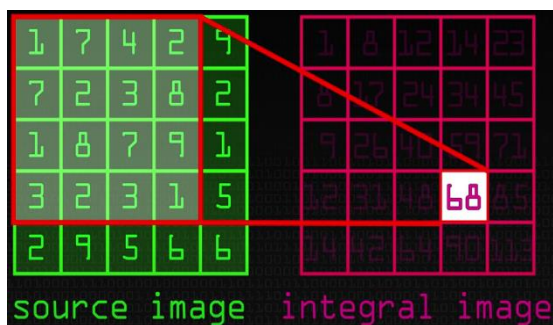
Единствените ориентации на детекција на карактеристиките се хоризонтално, вертикално и дијагонално.

Множеството се состои од околу 180,000 карактеристики (сите можни комбинации од карактеристики со 2, 3 и 4 правоаголници) кои би биле соодветни за дадениот регион од сликата во кој би можело да се наоѓа лицето.

### *Integral Image (интегрална слика)*

За да се забрза процесот на пресметување на Нааг карактеристиките, особено за сликите со висока резолуција, алгоритмот користи **интегрална слика**. Оваа структура овозможува брзо пресметување на сумата на пикселите во било кој правоаголник на сликата. Во интегралната слика, секој пиксел содржи збир на вредноста на самиот себе и сите пиксели над и лево од него.

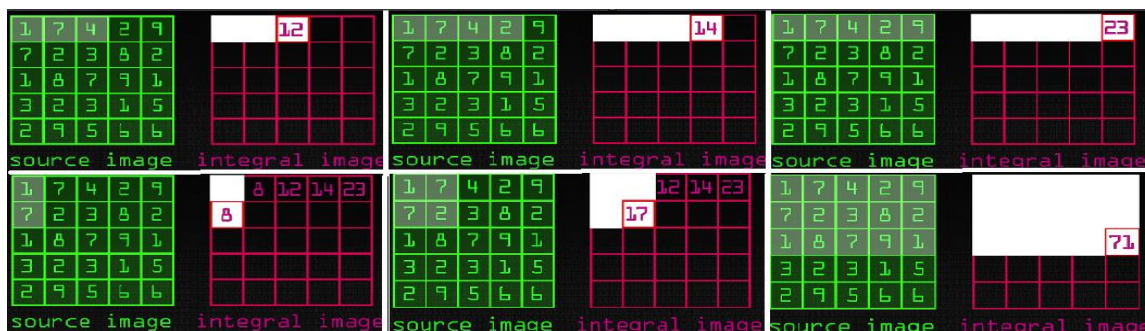
Процес на пресметка:



Резултат:



Процес на пресметка по чекори:



### Adaboost

Алгоритмот е базиран на *Adaboost* за:

- избор на најдобрите (најсоодветните) Хаг карактеристики од множеството
- да го најде најдобриот редослед на карактеристиките по кој ќе се врши детекцијата

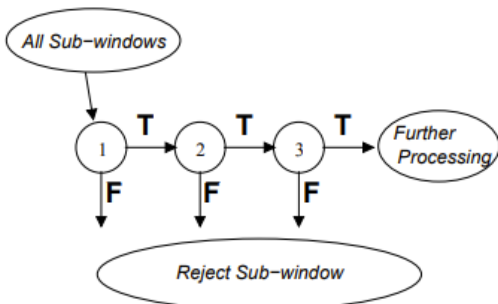
Adaboost е техника за машинско учење која избира мал број „прости класификатори“ (едноставни правила, базирани на една карактеристика) и ги комбинира за да создаде еден „силен класификатор“.

На овој начин се намалува комплексноста на моделот, задржувајќи ја точноста на детекцијата.

### Каскаден класификатор

За да се подобри брзината, Хаг Cascade користи каскаден пристап. Алгоритмот не проверува секој дел од сликата со сите карактеристики. Наместо тоа, тој користи серија класификатори, организирани во каскади, каде секоја каскада (ниво) ги отфрла регионите што не содржат лице.

Регионите што минуваат низ сите каскади на проверка се сметаат за лице.

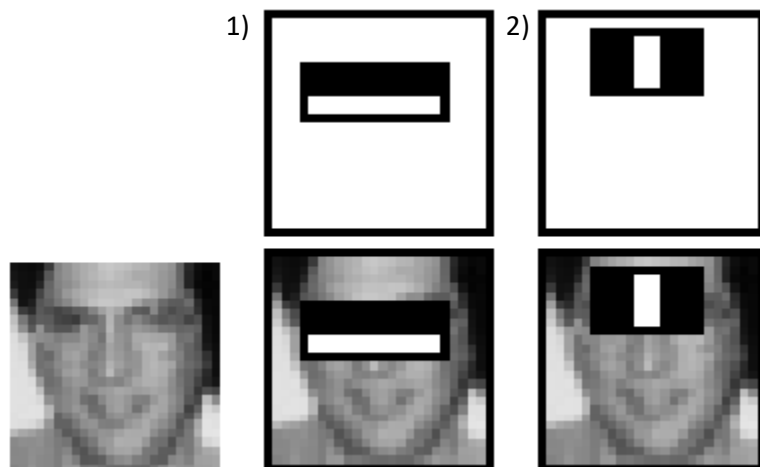


Целиот процес на одлука се прави во форма на дегенерирано дрво на одлуки, наречено каскада.

Првото ниво е многу едноставно и брзо ги елиминира областите кои се со голема веројатност да не се лице. Секој следен слој станува сè покомплексен, користејќи се повеќе карактеристики, но проверува помал број области.

Целосниот каскаден детектор на лице се состои од 38 нивоа со повеќе од 6000 карактеристики.

Оваа техника овозможува брза и ефикасна детекција, бидејќи најголем дел од регионите на сликата се отфрлаат на почетните фази, за таа цел моделот е тестиран на множество слики со и без лица.



Првата и втората карактеристика избрани од AdaBoost.

Двете карактеристики се прикажани во горниот ред. Потоа во долниот ред се обложани на слика која се користи за тренирање на алгоритмот.

- 1) На првата карактеристика ја пресметува разликата во интензитетот помеѓу пределот на очите и пределот на горниот дел од образите.
- 2) Втората карактеристика се споредува интензитетот на регионот од очните со интензитетот на регионот од носот.

### *Multiscale детекција*

Haar Cascade алгоритмот бара лица со различни големини, така што се применува на различни големини на сликата (scale). Ова овозможува да се најдат лица кои можат да бидат на различни растојанија од камерата.

### *DNN module in OpenCV*

Модулот **DNN** во OpenCV (Deep Neural Network module) е специјално дизајниран за обработка и користење на претходно обучени длабоки невронски мрежи за задачи како што се класификација, детекција на објекти, препознавање на лица, сегментација, и многу други задачи поврзани со компјутерски визија (Computer Vision).

DNN модулот поддржува користење на различни формати на модели обучени во популарни библиотеки за длабоко учење како што се:

- Caffe
- TensorFlow
- Torch
- ONNX (Open Neural Network Exchange)
- Darknet (YOLO)

Модулот е оптимизиран за работа на CPU, и ако е достапна поддршка за GPU (преку CUDA или OpenCL), може значително да се забрза извршувањето на невронските мрежи.

Модулот обезбедува лесен начин за вчитување на слики, и за извршување на inference (извршување на моделот за добивање резултати). На пример, `cv2.dnn.readNetFromCaffe()` или `cv2.dnn.readNetFromTensorflow()` овозможуваат лесно вчитување на модели, а `net.forward()` се користи за извршување.

За да се овозможи брза обработка на слики од моделите, DNN модулот користи концепт на **blob** (Binary Large Object). Со функцијата `cv2.dnn.blobFromImage()`, сликата се подготвува за користење во мрежата преку нормализација, скалирање, и менување на димензиите, така што ги одговара на барањата на невронската мрежа.

Модулот е целосно интегриран со останатите функции на OpenCV, што значи дека резултатите од длабоките мрежи можат да се комбинираат со стандардни операции како филтрирање, прагови, и детекција на рабови.

### DNN Face Detector from DNN module in OpenCV

Конкретно во мојата апликација користен е тренираниот модел **res10\_300x300\_ssd\_iter\_140000** во формат на Caffe framework, за детекција на лице.

За имплементација на овој модел потреби се две датотеки кои можат да се симнат од репозиториумот [opencv](#) и [3rdparty](#) на GitHub:

- [res10\\_300x300\\_ssd\\_iter\\_140000.caffemodel](#) - претходно обучен модел (pre-trained model) во форматот на **Caffe framework**. Оваа датотека содржи:
  - **тежини и параметри** на моделот за да се направат точни предвидувања за присутност на лице во слика. Кои се тренирани врз слики од **WIDER FACE** датасетот.
  - 300x300, укажува дека моделот е трениран со влезни слики со големина од **300x300 пиксели**. Затоа, пред процесирање, секоја влезна слика се намалува или зголемува на оваа димензија.
  - `iter_140000` означува дека моделот ја добил својата финална форма по **140.000** итерации на обука. Ова покажува низ колку ажурирања (или чекори за оптимизација) поминал моделот за време на обуката.
  - Моделот е трениран користејќи **Single Shot Multibox Detector (SSD)** со **ResNet-10** како основна архитектура (backbone).
- [deploy.prototxt](#) - Таа претставува текстуална датотека со структура слична на JSON која ги содржи сите дефиниции на слоевите на невронската мрежа.

Residual Networks (ResNets) се конволуциски архитектури за длабоко учење, дизајнирана е за да ги надминат проблемите со длабоките мрежи, бидејќи како се зголемува длабочината на мрежата, така прецизноста се намалува (проблем на деградација). ResNet моделите користат **skip connections (врски на прескокнување)** кои го спречуваат тој проблем.

Постојат повеќе варијанти на Residual Networks кои работат на истиот принцип само што се разликуваат по бројот на слоевите. Во проектот е користена ResNet-10 невронската мрежа, длабока 10 слоја.

Основни елементи на структурата на ResNet-10:

- **Конволуциски слоеви:** Овие слоеви извлекуваат карактеристики од сликата, обработувајќи ги различните визуелни информации низ низата на филтри (kernels).
- **Batch Normalization:** Овој слој ја нормализира излезната вредност на секој слој, што помага да се стабилизира тренинг процесот и да се забрза конвергенцијата.
- **ReLU (Rectified Linear Unit)** активациски функции: Се применуваат по секој конволутивен слој за да се добие нелинеарност во мрежата.
- **Residual (skip) connections:** Овие врски ги заобиколуваат некои слоеви и го додаваат излезот од претходните слоеви директно кон идните, што им овозможува на подлабоките слоеви да учат без да влијаат на перформансите.

ResNet-10 во контекст на SSD е изведен така што служи како **feature extractor**. Овој дел е одговорен за извлекување на релевантните карактеристики од сликата (feature maps).

Откако карактеристиките се извлечени преку ResNet-10, моделот преминува во делот на **SSD**, кој се користи за детекција на објекти. Овој дел користи повеќе конволутивни слоеви за детекција на објекти на различни слоеви:

- **Convolutional Feature Maps:** SSD генерира feature maps од различни скали и резолуции, што му овозможува на моделот да детектира лица на различни позиции и големини во сликата.
- **Multibox слоеви за регресија и класификација:**
  - **Bounding Box Regression:** Овие слоеви предвидуваат координати на рамките (bounding boxes) околу детектираните лица.
  - **Object Class Scores:** Овие слоеви даваат веројатност дека даденото bounding box содржи лице (во овој случај, класификација на лицето).
- **Non-Maximum Suppression (NMS):** Овој механизам се користи за отстранување на дуплирани рамки и задржување на најрелевантните.

Бидејќи SSD извршува детекција во еден чекор и го прави тоа многу брзо, што е идеален избор идеално за апликации во реално време.

### Анонимизација на лице

Со зголемената употреба на системи за детекција на лице, доаѓа потребата за заштита на приватноста на поединците. **Анонимизацијата на лице** претставува процес во кој детектираното лице се обработува на начин што ќе спречи негово идентификување, обично преку методи како **замаглување** или **покривање** на лицето. Овој процес е клучен во домени каде што се процесираат чувствителни податоци, како што се видеа од надзорни камери или слики што се користат за анализа без да се наруши приватноста на луѓето.

Во проектот се користат 2 методи за замаглување на слика:

- Box blur (average blur)
- Gaussian blur



## Gaussian Blur

Гаусовиот алгоритам за замаглување на слика (Gaussian blur) е базиран на примена на Гаусова функција и стандардната девијација, која генерира излезна слика со мазни и замаглени рабови.

### Гаусова функција

Гаусовата функција во две димензии се дефинира со:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Каде:

- $(x, y)$  се координатите на околните пикселите на пикселот (во центарот) што го обработуваме.
- $\sigma$  е стандардната девијација, која контролира колку силно ќе биде замаглувањето.

Функционира на начин :

- Пикселите во сликата се заменуваат со збир на тежинските вредности (weight value) на околните пиксели, каде што тежинската вредност се определуваат според Гаусовата функција.
- Стандардната девијација на Гаусовата функција ја контролира количината на замаглување. Поголемата вредност на  $\sigma$  значи дека се земаат предвид повеќе соседни пиксели при замаглувањето, резултира со поизразено замаглување, додека помалата вредност за  $\sigma$  резултира со послабо замаглување (земени се предвид помалку соседни пиксели). Вредноста на стандардната девијација ја задава корисникот, во зависнот од желбата на јачината на интензитетот на замаглување кое сака да го постигне.
- Се генерира Гаусов филтер (кERNEL) кој претставува квадратна матрица (со непарни димензии, поради централниот пиксел) што содржи пиксели со вредности добиени од Гаусовата функција кои се применуваат врз пикселите од сликата. Центарот на филтерот има најголема вредност, а тежинската вредност на околните пиксели се намалуваат како што се оддалечуваме од центарот.

Пример за Гаусов филтер со 3x3 матрица:

$\frac{1}{16}$	1	2	1
	2	4	2
	1	2	1

 $\Rightarrow$ 

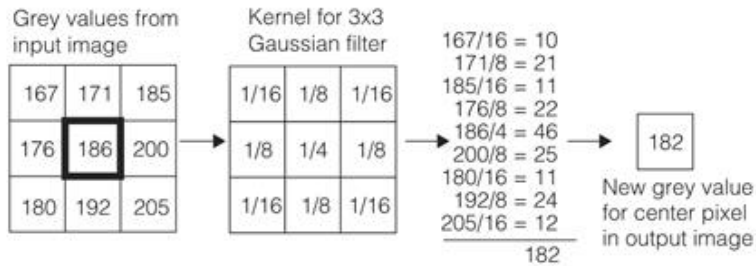
1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

На левата страна, вредностите се однесуваат на распределба на пиксели според Гаусовата функција. Средниот пиксел (со вредност 4) има најголемо значење бидејќи се наоѓа во центарот, а околните пиксели имаат помало значење.

За да се добие тежинска матрица која ја запазува осветленоста на сликата, сите вредности се делат со збирот на оригиналните вредности.

Во овој случај, збирот на сите вредности е:  $1+2+1+2+4+2+1+2+1=16$

Секој елемент во kernel-от се дели со 16 за да се добие резултирачкиот филтер кој е прикажан на десната страна.



При секој пиксел на сликата, се зема 3x3 околина, се множат соодветните вредности со пикселите и се собираат резултатите за да се добие нова вредност на централниот пиксел.

- Се врши конволуција помеѓу Гаусовиот филтер и сликата. Тоа значи дека филтерот се поместува низ секој пиксел од сликата, и на секој пиксел му се пресметува нова вредност така што се земаат предвид соседните пиксели. За секој пиксел се прави сума од вредностите на околните пиксели од филтерот, помножени со соодветната тежинска вредност. Ова резултира во нова вредност за централниот пикселот, што создава ефект на замаглување.
- Доколку сликата е во боја, процесот се применува на секој канал поединечно (црвен, зелен и син за RGB слики).
- Се смета за нископропусен филтер кој задржува ниска просторна фреквенција и го намалува шумот на сликата и занемарливите детали на сликата.

Методот ги прифаќа следните параметри:

`dst=cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])`

**src** – променлива што ја претставува влезната слика (типично NumPy низа)

**ksize** - Го дефинира Гаусовиот кернел [висина ширина]. Висината и ширината мора да бидат непарни (1,3,5,...) и можат да имаат различни вредности. Ако ksize е поставено на [0,0], тогаш ksize се пресметува од сигма вредноста.

**sigmaX**(опционално) - Стандардното отстапување во насоката X. Оваа вредност го контролира степенот на заматување. Ако е поставено на 0, тоа ќе се пресметува од големината на кернелот (ksize).

**dst** (опционално): Излезната слика, која има иста големина и тип како и влезната слика.

**sigmaY**: Стандардното отстапување во насока Y. Ако е поставено на 0, стандардно е на истата вредност како и sigmaX.

**borderType** - Ова се наведените граници на сликата додека кернелот се применува на границите на сликата.

Кога се применува филтер, како што е Gaussian blur, кај пикселите на рабовите на сликата, се јавува проблем бидејќи пикселите на работ немаат доволно соседни пиксели од сите страни за правилно да се изврши конволуцијата. За да се реши овој проблем, се користат неколку различни техники:

### Zero-padding (пополнување со нули)

Оваа техника подразбира додавање на пиксели со вредност 0 околу работ на сликата. Ова создава "рамка" од нули околу сликата која овозможува примена на филтерот на рабовите. Недостаток на оваа техника е што може да создаде вештачки затемнети рабови бидејќи нултите вредности влијаат на конволуцијата.

### 2. Replicate (реплицирање на рабовите cv2.BORDER\_REPLICATE)

Со оваа техника, вредноста на пикселите на работ се реплицира (клонираат) за да се пополни просторот надвор од сликата. Ова ја намалува појавата на вештачки остри рабови.

### 3. Reflect (рефлектирање на сликата cv2.BORDER\_REFLECT)

Во оваа техника, рабовите на сликата се "рефлектираат" (огледуваат) за да се пополни просторот. Тоа значи дека пикселите непосредно до работ се пресликуваат на спротивната страна. Ова помага да се зачува непрекинатоста на текстурата на сликата и често се користи за природни и мазни преоди.

### 4. Wrap (обвивање cv2.BORDER\_WRAP)

Оваа техника е позната и како "циклично пополнување". Во неа, пикселите од спротивниот крај на сликата се користат за да се пополни просторот на работ. Ова значи дека ако се обработува левиот раб на сликата, вредностите од десниот раб се користат за пополнување, и обратно. Ова може да доведе до циклични артефакти и не е секогаш најсоодветно за природни слики.

### 5. Constant Padding (пополнување со константни вредности cv2.BORDER\_CONSTANT)

Тука просторот надвор од сликата се пополнува со константна вредност, како на пример средната вредност на пикселите во сликата или некоја дефинирана вредност (на пример, бела или црна боја). Ова е едноставна техника, но може да создаде неестетски премини на работ на сликата.

#### Пример со 3x3 филтер:

Ако се применува 3x3 Gaussian филтер на пиксел близу аголот на сликата, ќе недостасуваат некои околни пиксели. Со горенаведените техники, овие "празни" места се пополнуваат за да овозможат целосна конволуција.

#### Изборот на техника зависи од:

- Природата на сликата.
- Естетскиот резултат што се сака да се постигне.
- Брзината на алгоритмот (некои техники се побрзи од други).

Овие техники овозможуваат филтерот да се примени на целата слика, вклучително и на пикселите на рабовите, без значителни артефакти.

Кога не се задава експлицитно техника за справување со рабови, се користи **BORDER\_DEFAULT**, кое е всушност **BORDER\_REFLECT\_101** е *рефлектирање на рабовите* (REFLECT). Оваа техника се користи во многу библиотеки за обработка на слики, како OpenCV и други, бидејќи е едноставна и дава добри резултати без видливи артефакти.

Може да се користи и друга техника за справување со рабови, тоа се постигнува со дефинирање на параметарот `borderType`, на пример:

**BORDER\_REPLICATE**: реплицирање на рабовите.

**BORDER\_CONSTANT**: пополнување со константна вредност.

Gaussian blur е ефикасен начин за замаглување на слики и се користи во многу апликации како обработка на слики, компјутерска визија и графички дизајн.

## Box blur (Average blur)

Функцијата **cv2.blur(src, dst, ksize, anchor, borderType)** која е достапна во библиотеката OpenCV го имплементира алгоритмот **box blur** или **average blur**.

Овој метод ги прифаќа следните параметри :

- **src** – објект што го претставува изворот (влезна слика) за оваа операција.
- **dst** - Објект што ја претставува дестинацијата (излезна слика) за оваа операција.
- **ksize** – Size што ја претставува големината на kernel, димензии на кернал (height, weight).
- **anchor** – Променлива од типот цел број што ја претставува точката на прицврстување.
- **borderType** - Променлива од типот цел број што го претставува типот на рамка околу рабовите што треба да се користи за излезот.

Начин на функционирање:

- Едноставен филтер, кој го пресметува просекот на сите пиксели кои се наоѓаат во областа на јадрото и го заменува централниот пиксел со пресметаниот просек. Корисникот задава големина на прозорецот (kernel), кој обично е квадратна матрица од типот  $M \times M$ . Оваа матрица ги дефинира пикселите околу секој пиксел за кои ќе се пресмета средната вредност.

На пример, ако прозорецот е  $3 \times 3$ , тоа значи дека за секој пиксел ќе се разгледаат неговите 8 соседи (горе, долу, лево, десно, и дијагоналните).

- Пресметување на средна вредност:
  - За секој пиксел во влезната сликата, филтерот се поставува околу пикселот во центарот.
  - Вредностите на пикселите во прозорецот се собираат и се пресметува средната вредност.
  - Оваа средна вредност се поставува како новата вредност на централниот пиксел.

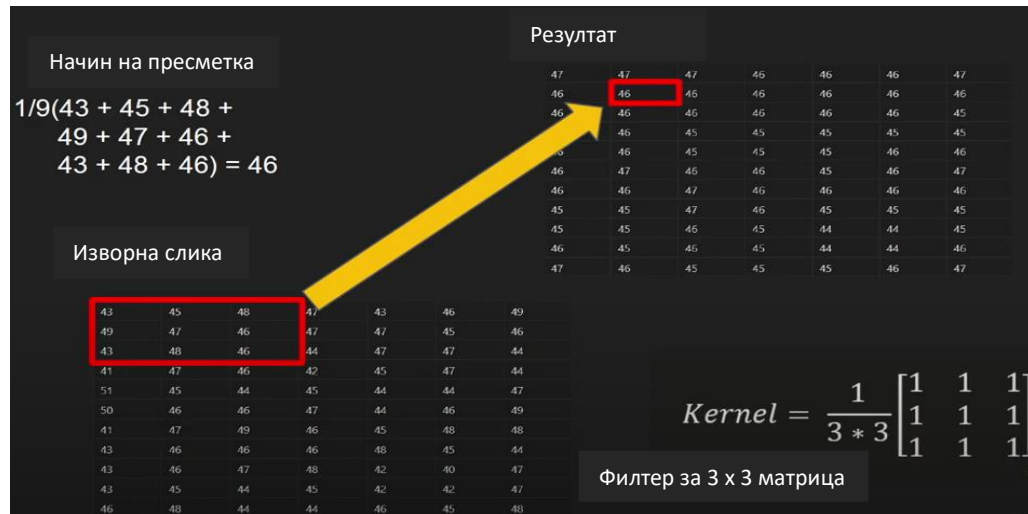
Формално, за пикселот на позиција  $(i, j)$  на излезната слика, новата вредност се пресметува како:

$$I'(i, j) = \frac{1}{M \times M} \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} I(i + k, j + l)$$

каде што  $I'(i, j)$  е новата вредност на пикселот, а пикселот  $I(i, j)$  се оригиналните вредности на пикселите.

- Справување со рабовите: пикселите на рабовите од сликата немаат доволно соседи за да се пополни прозорецот (kernel), па OpenCV користи техники за пополнување на рабовите, како **replicate**, **reflect**, или други методи кои можат да се зададат преку параметарот borderType. Ако не се зададе експлицитен borderType, се користи стандардниот метод **BORDER\_DEFAULT** (обично **BORDER\_REFLECT\_101**).
- Кај cv2.blur сите пиксели во прозорецот имаат еднаква тежина. Ова ја прави имплементацијата и извршувањето на овој алгоритам побрзо, но резултатот е помалку природен и помазен за разлика од Gaussian blur.
- Филтрите се препорачува да бидат со непарни димензии за ширина и должина, поради симетрија околу централниот пиксел, но cv2.blur може да прави калкулации и без точно дефиниран центар. Се справува со недостатокот на центар со мало поместување на регионот над кој работи филтерот. На пример, со кернел  $90 \times 90$ , OpenCV може да го користи горниот лев дел од кернелот наместо централниот пиксел. Ова создава ефект на замаглување сличен

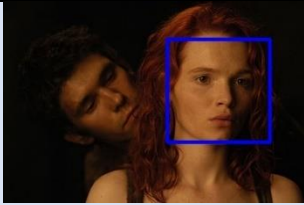
на оној што би го добиле со кернел со непарна големина, но регионот сепак не е совршено симетричен.



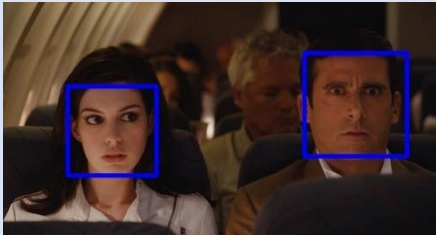
## Споредба помеѓу двата модела за детекција на лице

Двата модели за детекција на лице се тестирали на истиот [dataset](#).

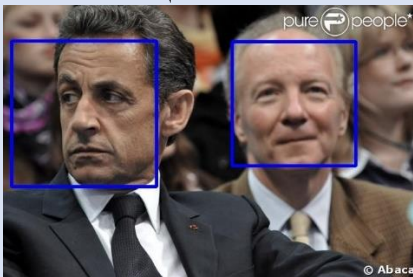
Haar Cascade (Машинско учење):	DNN Face Detector (Длабоко учење):
<p><i>Базична технологија:</i></p> <p>Се базира на класични методи на <b>машинско учење</b>. Користи посебни карактеристики (Haar features) кои се извлекуваат од сликите и се користат за разликување на лица од позадината.</p> <p>Користи <b>Cascaded Classifier</b>, каде што секој класификатор во каскадата кои се едноставни, но голем број од нив се применуваат последователно за да ја зголемат точноста на детекцијата.</p>	<p><i>Базична технологија:</i></p> <p>Овој модел се базира на <b>Convolutional Neural Networks (CNNs)</b>. Специфично, користи <b>Single Shot Multibox Detector (SSD)</b>, кој е модел за детекција на објекти.</p> <p>Наместо да извлекува фиксирани карактеристики, CNN автоматски учи релевантни карактеристики од податоците преку длабоки слоеви.</p>
<p><i>Прецизност и робусност:</i></p> <p><b>Не прецизен</b> за детекција на лица, особено кога лицата се <b>ротирани</b>, <b>замаглени</b> или имаат <b>различно осветлување</b>.</p> <p>Ротирани</p>	<p><i>Прецизност и робусност:</i></p> <p><b>Многу прецизен</b> и робустен дури и во комплексни ситуации како <b>различни агли</b>, <b>осветлување</b>, <b>големини на лице</b>, и во присуство на позадински елементи.</p>



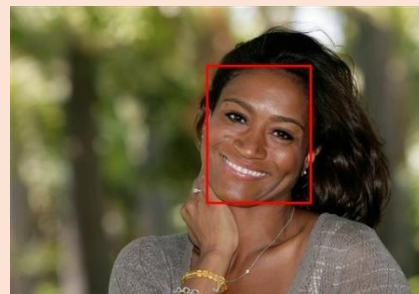
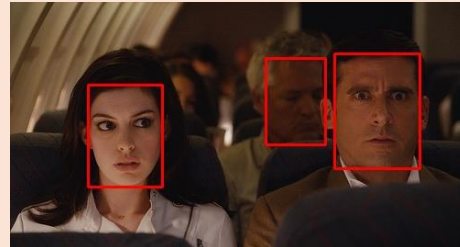
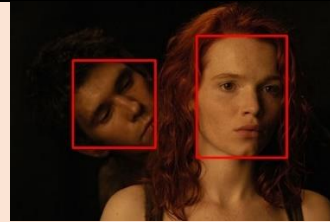
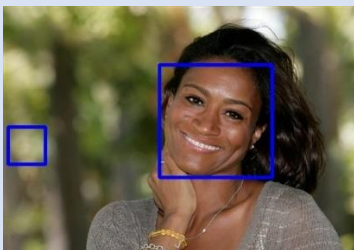
Различно осветлени



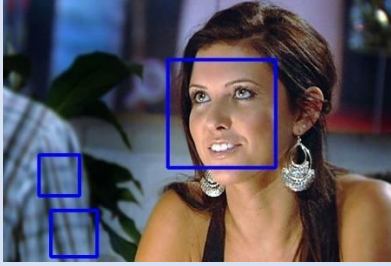
Заматено лице



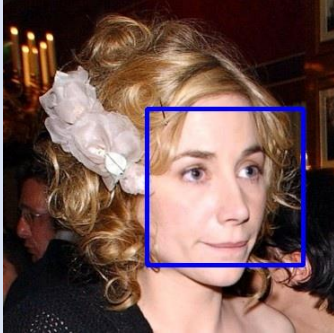
Може да произведе многу **лажни позитивни** резултати, бидејќи е потпира на фиксирани карактеристики кои не се адаптираат на различни ситуации.



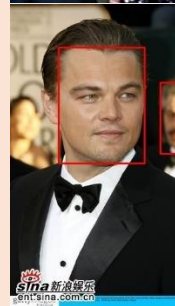
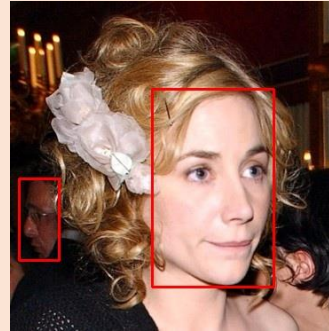




Не успева да детектира лица дури и кога се лицата поставени во необична позиција на сликата.




Успева да детектира лица дури и кога се лицата поставени во необична позиција на сликата.



Многу поретко дава **лажни позитиви**, бидејќи моделот учи директно од податоците и препознава понапредни карактеристики.

Ова се гел од малкуте негативи што излегоа како резултат:

	
<p><i>Брзина и ефикасност:</i>  <b>Брз модел</b>, бидејќи користи каскаден класификатор, кој одбира само релевантни делови од сликата за анализа.</p> <p>Одличен за <b>реално-временска детекција</b> на уреди со помали процесорски капацитети (вградени системи, веб камери).</p>	<p><i>Брзина и ефикасност:</i>  <b>Побавен модел</b> во споредба со Haar Cascade, бидејќи користи длабоки невронски мрежи со повеќе слоеви за обработка на сликата.</p> <p>Бара <b>повисока процесорска моќ</b>, па затоа е поефикасен кога се користи на уреди со моќни графички процесори (GPUs).</p>
<p><i>Тренинг и зависност од податоци:</i>  Користи <b>предтренинг со пред-дефинирани карактеристики</b>, што значи дека нема потреба да се врши дополнителен тренинг за детекција на лице. Моделот е статичен и не учи нови карактеристики.</p> <p>Лесно се имплементира и користи на различни апликации, но со ограничени перформанси во различни ситуации.</p>	<p><i>Тренинг и зависност од податоци:</i>  Овој модел е резултат на <b>длабоко учење</b> и мора да биде трениран со големи количини на податоци. Научен е од <b>огромни слики со етикетирани лица</b>, што овозможува да се подобри со секој нов тренинг сет.</p> <p>Флексибилен е и може да се адаптира на нови апликации со дополнителен тренинг, што го прави поотпорен на нови сценарија.</p>
<p><i>Примена:</i>  Често се користи во <b>поедноставни апликации</b>, како што се веб-камери, системи за надзор, или мобилни апликации, каде што <b>брзината и ниската процесорска потрошувачка</b> се клучни фактори. Не е многу прецизен за комплексни задачи со многу варијации во лицата</p>	<p><i>Примена:</i>  Користен е во <b>поразвиени апликации</b> каде што <b>прецизноста и робусноста</b> се од голема важност, како во <b>автономни возила, системи за безбедност на високо ниво</b>, или апликации кои треба да препознаат лица во големи групи. Многу е применлив во апликации кои бараат <b>анализа на големи количини на податоци</b> (на пример, масовен надзор).</p>



**Заклучок:**

**Haar Cascade** е корисен за едноставни и брзи апликации каде што ресурсите се ограничени, но тој има помала прецизност и не е толку робустен во споредба со **Res10\_300x300 SSD** моделот, кој, иако бара повеќе процесорска моќ, нуди подобра точност, флексибилност и отпорност на варијации во сликите.

## Користена литература

### **Haar Cascade:**

<https://medium.com/@baselanaya/faces-detection-using-haar-cascade-3e175aef84f5>

[https://docs.opencv.org/4.x/d2/d99/tutorial\\_js\\_face\\_detection.html](https://docs.opencv.org/4.x/d2/d99/tutorial_js_face_detection.html)

<https://www.youtube.com/watch?v=uEJ71VIUmMQ>

<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

### **DDN:**

<https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c>

<https://medium.com/@zomev/deep-neural-network-dnn-explained-0f7311a0e869>

[https://www.youtube.com/watch?v=Q1JCrG1bJ-A&list=PLgPbN3wia\\_PeT1\\_c5jiLW3RJdR7853b9&index=15](https://www.youtube.com/watch?v=Q1JCrG1bJ-A&list=PLgPbN3wia_PeT1_c5jiLW3RJdR7853b9&index=15)

### **Box Blur or Average blur (filtering):**

<https://www.javatpoint.com/opencv-blur>

<https://www.youtube.com/watch?v=oXlwWbU8I2o&t=9333s>

[https://www.youtube.com/watch?v=\\_MDV3pBGsfY](https://www.youtube.com/watch?v=_MDV3pBGsfY)

### **Gaussian blur:**

<https://www.sciencedirect.com/topics/engineering/gaussian-blur>