Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Семинарска работа по предметот

Дигитално процесирање на слика

Тема:

Замаглување и анонимизација на лица во слики

Изработи

Јасна Јованова

223103

Скопје

Септември, 2024

Вовед

Во денешно време се почесто и почесто секој од нас наидува на слики од личности чии лица не можеме да ги познаеме затоа што се замаглени и анонимизирани.

Анонимизацијата се користи за да се скрие идентитетот на луѓето, најчесто за:

- **Заштита на приватноста**: На пример, при објавување на фотографии на социјалните мрежи каде што луѓето не сакаат да бидат јавно видливи.
- **Правни цели**: Во судски случаи или полициски истраги, кога е потребно да се заштити идентитетот на сведоци или жртви.
- **Медиумско известување**: Новинарите често користат замаглување кога известуваат за чувствителни случаи за да не го откријат идентитетот на вклучените лица.
- **Контрола на содржина**: Во видеа и фотографии кои содржат непожелни или несоодветни содржини, лицата или делови од сликите може да се замаглат за да не бидат видливи.

За анонимизација на лицата во слики се користи замаглување.

Замаглувањето и анонимизацијата имаат за цел да заштитат нечија приватност, но анонимизацијата може да вклучува поширок спектар на техники и податоци, а не само визуелни информации.

Сето од горенаведеното е изводливо токму преку демо апликацијата која ја направив, чиј тек на изработка и користење во целост е опишан во продолжене. Демо апликацијата за замаглување и анонимација на лица ја изработив со помош на OpenCV и програмскиот јазик Python.

Што е замаглување на лица и како може да се искористи за анонимација на лица?

Замаглувањето на лицето е метод на компјутерска визија што се користи за анонимизирање лица во слики и видеа, т.е. да ги направи луѓето во сликите непрепознатливи.

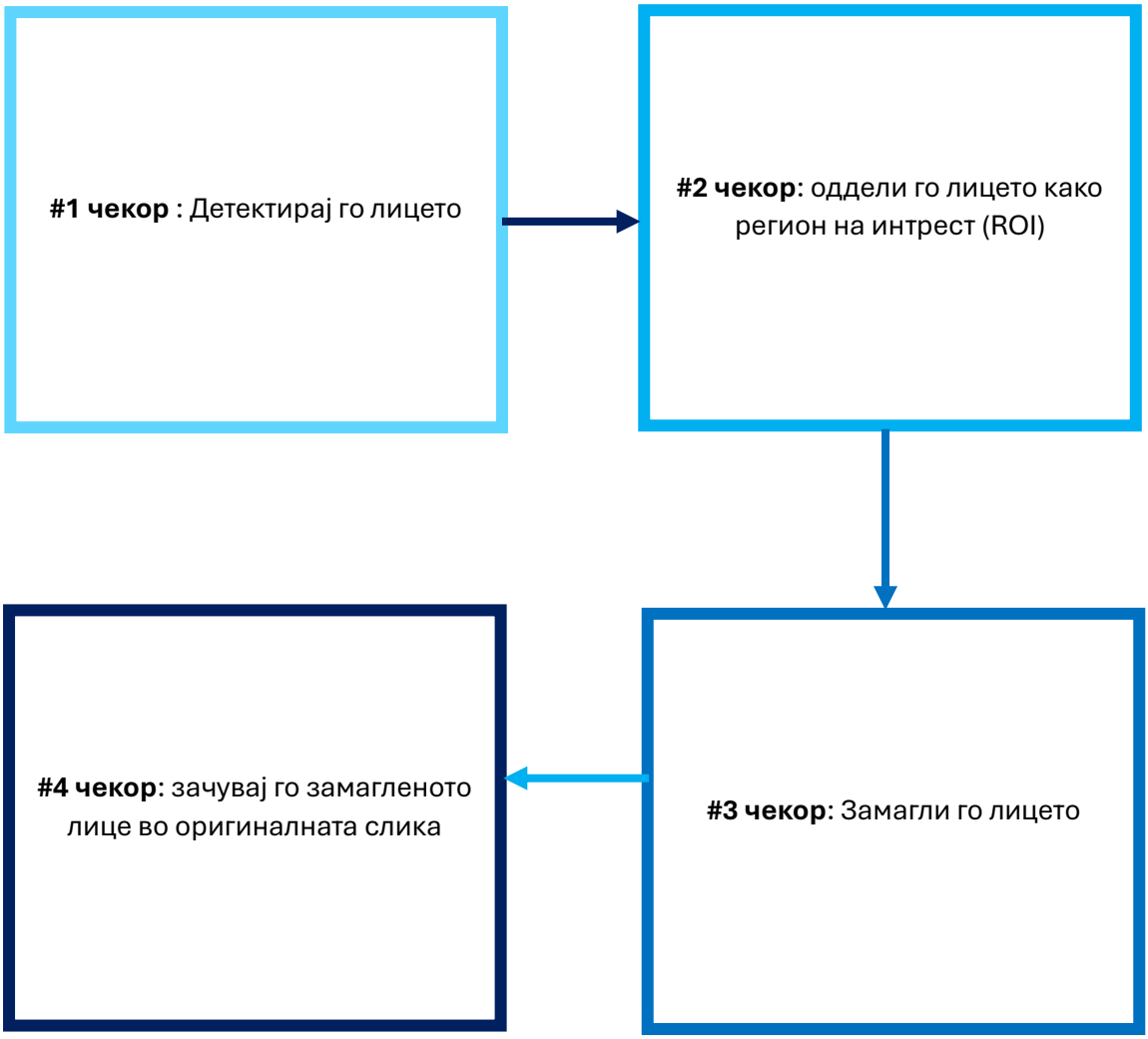
Во оваа семинарска работа имплементирани се два методи за замаглување на лица:

-**Гаусово замаглување** за анонимизирање на лица

-Примена на ефектот “Замаглување со пиксели” (**пикселизација**)

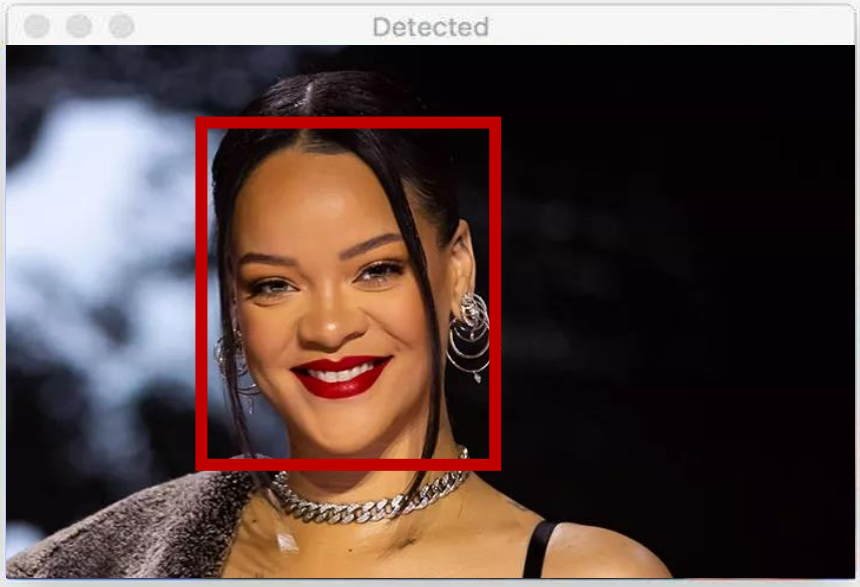
Слика 1: пример за замаглување на лицето и анонимизација – забележете како ликот е замаглен и човекот на сликата е непрепознатлив

4 чекори за извршување на замаглување лица и анонимизација



Слика 2: Замаглувањето на лицето со OpenCV и Python може да се подели на четири чекори.

Чекор број 1 е да се изврши детекција на лицето во потребната слика.



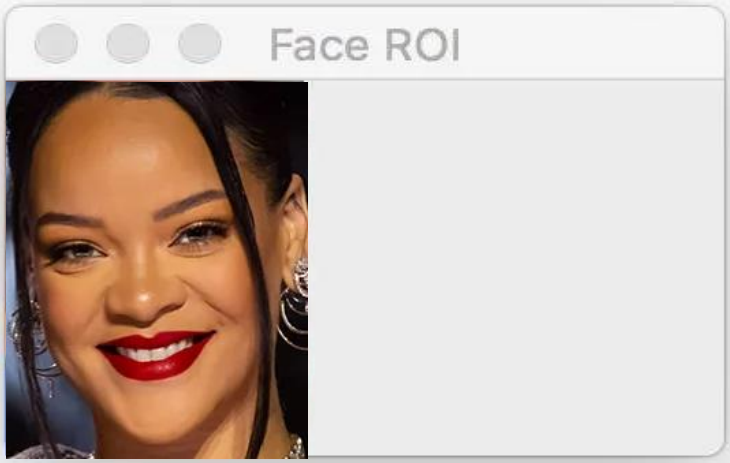
Слика 3: Првиот чекор за замаглување на лице со OpenCV и Python е да се детектираат сите лица во слика/видео

Овде може да се користи кој било детектор за лице, под услов да може да ги произведе координатите на граничните полиња на лицето во пренос на слика или видео.

Најчестите детектори за лице се:

- Хаар каскади
- HOG + Линеарен SVM
- Детектори за лице базирани на длабоко учење

Откако ќе се окрие лицето, **Чекор #2 е да се извлече регионот на интерес (ROI):**



Слика 4: Вториот чекор за замаглување на лица со Python и OpenCV е да се извлече регион на лицето од интерес(ROI).

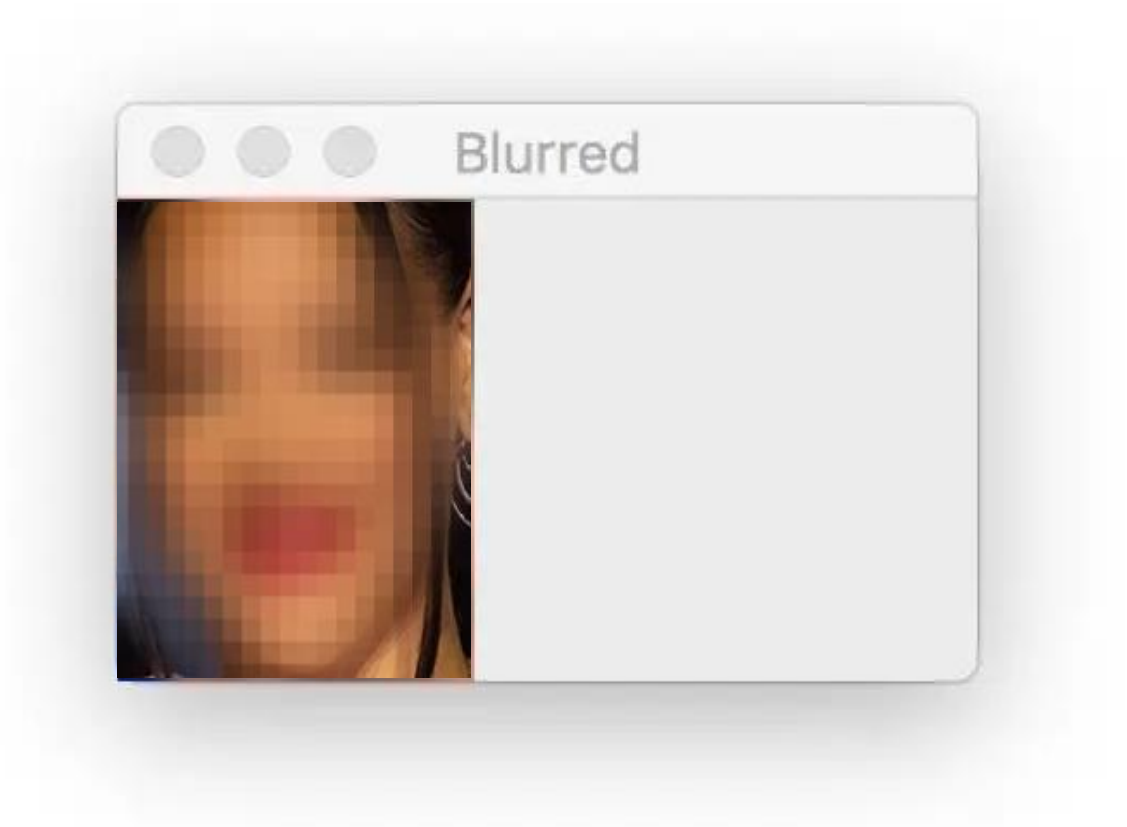
Детекторот за лице ја дава рамката на (x, y)-координати на лице во слика.

Овие координати обично претставуваат:

- Почетната x-координата на кутијата за ограничување на лицето
- Завршната x-координата на лицето
- Почетната y-координата на локацијата на лицето
- Завршната y-координата на лицето

Потоа можете да ги користите овие информации за да го извлечете самиот ROI на лицето, како што е прикажано на **Слика 4** погоре.

Откако се добива ROI, **чекор бр. 3 е всушност да се замагли/анонимизира лицето**:

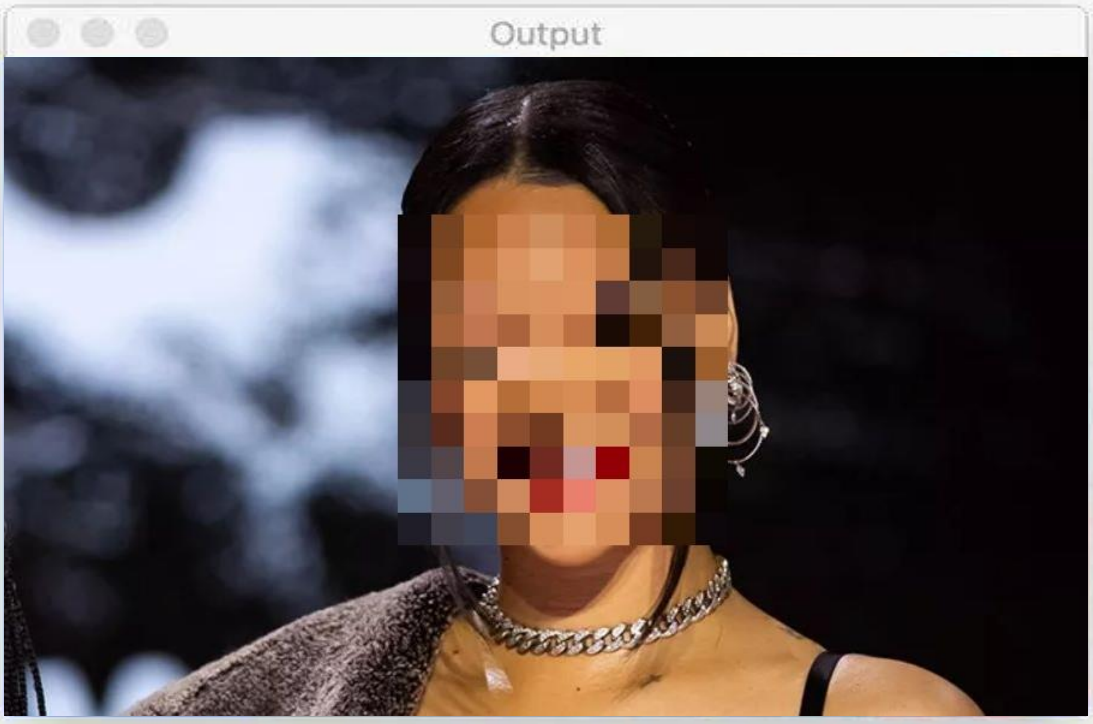


Слика 5: Третиот чекор е да го примените алгоритам за заматување. Во ова упатство, учиме два такви алгоритми за замаглување - Гаусово заматување и пикселирање.

Најчесто се користи Гаусовиот метод, меѓутоа тој знае да не наликува естетски убаво, па затоа се користи и методот на замаглување со пиксели.

Како точно ќе ја „замаглите“ сликата зависи од вас - *важен дел е што лицето е анонимизирано.*

Откако лицето е заматено и анонимизирано, **Чекор #4 е да го зачувате заматеното лице назад во оригиналната слика:**



Слика 6: Четвртиот и последен чекор за замаглување на лице со Python и OpenCV е да се замени оригиналниот ROI на лице со ROI на заматено лице.

Користејќи ги оригиналните (x, y)-координати од откривањето лице (т.е., чекор бр. 2), можеме да го земеме анонимизираното лице и потоа да го складираме на оригиналната слика.

Лицето на оригиналната слика е заматено и анонимизирано - во овој момент процесот за анонимизација на лицето е завршен.

Замаглување на лица со Гаусово замаглување и OpenCV

Слика 7: Вака изгледа замагленото лице со користење на Гаусовиот метод.

```
1 usage
def apply_gaussian_blur(image, x, y, w, h):
    face = image[y:y+h, x:x+w]
    blurred_face = cv2.GaussianBlur(face, ksize: (99, 99), sigmaX: 30)
    image[y:y+h, x:x+w] = blurred_face
```

def apply_gaussian_blur(image, x, y, w, h): - функцијата apply_gaussian_blur, прима пет параметри:

- image: Оригиналната слика која ќе се обработува.
- x: X координатата на горниот лев агол на правоаголникот што го опкружува лицето.
- y: Y координатата на горниот лев агол на правоаголникот што го опкружува лицето.
- w: Ширината на правоаголникот.
- h: Висината на правоаголникот.

face = image[y:y+h, x:x+w]: Овој ред ја извлекува дел од сликата кој претставува лице.

- y:y+h означува редовите од y до y+h (од горниот до долниот дел на правоаголникот).
- x:x+w означува колоните од x до x+w (од левата до десната страна на правоаголникот).

- Резултатот е дел од сликата кој се чува во променливата face, која содржи само дел од сликата каде што се наоѓа лицето.

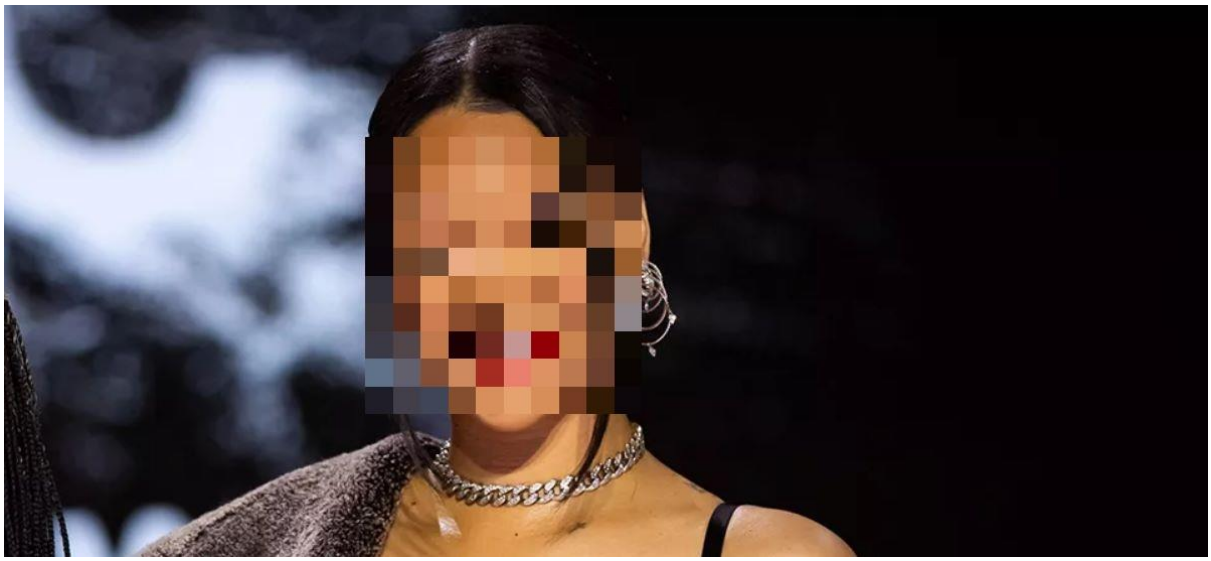
blurred_face = cv2.GaussianBlur(face, (99, 99), 30): Овој ред применува Gaussian Blur на извлеченото лице.

- **face**: Ова е делот од сликата кој претставува лице, на кој ќе се применува Gaussian Blur.
- **(99, 99)**: Ова е големината на јадрото (kernel). Во овој случај, 99x99 е големината на матрицата која ќе се користи за филтрирање на сликата. Бројот 99 е избран за големината на јадрото. Поголемо јадро ќе предизвика поголемо замаглување, додека помало јадро ќе предизвика помало замаглување. За оптимални резултати, големината на јадрото обично треба да биде некоја парна вредност.
- **30**: Ова е sigmaX, стандардната девијација во X правецот. Овој параметар контролира колку силно ќе биде замаглувањето. Поголема вредност на sigmaX ќе резултира со поголема распрснатост на ефектот на замаглување.

image[y:y+h, x:x+w] = blurred_face: Овој ред ги заменува оригиналните пиксели на лицето со замаглената верзија.

- **image[y:y+h, x:x+w]**: Овој сегмент на оригиналната слика.
- **blurred_face**: Обработената слика (замаглената верзија на лицето).

Создавање пикселирано замаглување на лице со OpenCV



Слика 8: вака изгледа замаглено лице со пикселизација

```
1 usage
def apply_pixelated_blur(image, x, y, w, h):
    face = image[y:y+h, x:x+w]
    small = cv2.resize(face, dsize: (10, 10), interpolation=cv2.INTER_LINEAR)
    pixelated_face = cv2.resize(small, dsize: (w, h), interpolation=cv2.INTER_NEAREST)
    image[y:y+h, x:x+w] = pixelated_face
```

def apply_pixelated_blur(image, x, y, w, h):: Ова ја дефинира функцијата apply_pixelated_blur која прима пет параметри исто како и претходната.

- image: Оригиналната слика која ќе се обработува.
- x: X координатата на горниот лев агол на правоаголникот што го опкружува лицето.
- y: Y координатата на горниот лев агол на правоаголникот што го опкружува лицето.
- w: Ширината на правоаголникот.
- h: Висината на правоаголникот.

face = image[y:y+h, x:x+w]: Овој ред ја извлекува дел од сликата кој претставува лице.

- y:y+h означува редовите од y до y+h (од горниот до долниот дел на правоаголникот).
- x:x+w означува колоните од x до x+w (од левата до десната страна на правоаголникот).
- Резултатот е дел од сликата кој се чува во променливата face, која содржи само дел од сликата каде што се наоѓа лицето.

small = cv2.resize(face, (10, 10), interpolation=cv2.INTER_LINEAR): Овој ред ја намалува резолуцијата на извлеченото лице до 10x10 пиксели.

- **cv2.resize(face, (10, 10), interpolation=cv2.INTER_LINEAR)**:
  - **face**: Извлечениот дел од сликата (лицето).
  - **(10, 10)**: Новата големина на сликата (намалена на 10x10 пиксели).
  - **interpolation=cv2.INTER_LINEAR**: Метод на интерполација кој се користи за намалување на сликата. INTER_LINEAR е добар избор за подобрување на квалитетот на намалената слика.

pixelated_face = cv2.resize(small, (w, h), interpolation=cv2.INTER_NEAREST): Овој ред ја зголемува намалената слика назад на оригиналната големина (w x h).

- **cv2.resize(small, (w, h), interpolation=cv2.INTER_NEAREST)**:
  - **small**: Намалената слика (10x10 пиксели).
  - **(w, h)**: Новата големина на сликата, што е оригиналната големина на лицето.
  - **interpolation=cv2.INTER_NEAREST**: Метод на интерполација кој се користи за зголемување на сликата. INTER_NEAREST се користи за

пикселизација, што значи дека секој пиксел ќе го задржи својот оригинален тон без флуидност, создавајќи пикселизиран ефект.

image[y:y+h, x:x+w] = pixelated_face: Овој ред ги заменува оригиналните пиксели на лицето со пикселизираниот дел.

- **image[y:y+h, x:x+w]**: Овој сегмент на оригиналната слика.
- **pixelated_face**: Обработената слика (замаглената верзија на лицето).

Гаусов метод или пикселизација, кој да се избере?

- **Гаусов метод**: Ако сакате да постигнете ефект на замаглување кој е помек и помалку агресивен, и ако сакате да одржите некои детали на сликата, Гаусовото замаглување е добар избор.
- **Пикселизација**: Ако сакате да направите детали на сликата целосно нечитливи, како што е анонимизацијата на лица или другите објекти, и не ви е важно изгледот да биде многу природен, пикселизацијата ќе биде поадекватна.

Пример на избор:

- **Гаусов метод**: Може да се користи за ублажување на бучава или создавање на ефект на размаз во уметнички фотографии.
- **Пикселизација**: Може да се користи за заштита на приватноста во фотографии со лица или за контрола на достапноста на детали на сликата.

Вие треба да одберете метод според вашиот конкретен случај на употреба и ефектот што сакате да го постигнете.

Спроведување на замаглување на лице во слики со OpenCV

Сега кога ги имаме имплементирано нашите два методи за замаглување на лица, да видиме како може да се применат за замаглување лице во слика користејќи OpenCV и Python.

```python
def main():
    image = cv2.imread('input.jpg')
    if image is None:
        print("Грешка: Не може да се вчита сликата.")
        return

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    print("Избери тип на замаглување:")
    print("1: Gaussian Blur")
    print("2: Pixelated Blur")
    choice = input("Внеси го бројот на твојот избор: ")

    for (x, y, w, h) in faces:
        if choice == '1':
            apply_gaussian_blur(image, x, y, w, h)
        elif choice == '2':
            apply_pixelated_blur(image, x, y, w, h)
        else:
            print("Невалиден избор!")
            return

    cv2.imshow( winname: 'Blurred Faces', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

def main():: Ова ја дефинира функцијата main. Функцијата main е главната функција која ќе се изврши кога програмата ќе се покрене.

image = cv2.imread('input.jpg'): Овој ред ја вчитува сликата од фајлот наречен input.jpg и ја чува во променливата image. Функцијата cv2.imread од библиотеката OpenCV ја чита сликата и ја враќа како матрица (numpy array).

- **if image is None:**: Овој услов проверува дали сликата не е успешно вчитана (т.е., дали image е None).
- **print("Грешка: Не може да се вчита сликата.")**: Ако сликата не може да се вчита, се испечати порака за грешка.
- **return**: Овој ред го прекинува извршувањето на функцијата main, што значи дека ако сликата не може да се вчита, функцијата ќе заврши тука.

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'): Овој ред создава објект CascadeClassifier користејќи фајл со (haarcascade_frontalface_default.xml) кој се користи за детектирање на лица. cv2.data.haarcascades е патеката до директориумот каде што се чуваат предефинираните .xml на Haar каскадите.

1. **cv2.CascadeClassifier**:
    - CascadeClassifier е класа во OpenCV која се користи за детектирање на објекти во слики. Оваа класа работи со каскадни класификатори, кои се специјални алгоритми за детектирање на објекти, како што се лица, во слики.
2. **cv2.data.haarcascades**:
    - cv2.data.haarcascades е патека до директориумот каде што се чуваат предобуките на Haar каскадите. Оваа патека е зададена од OpenCV библиотеката и покажува на место каде што се наоѓаат XML фајловите со

обучени модели за различни видови детекција, како што се лица, очи, итн.

3. **'haarcascade_frontalface_default.xml'**:
    - Ова е името на XML фајлот кој содржи предобука за детектирање на фронтални лица. XML фајлот е обучен модел што користи Haar каскади за да препознае лица во слики.
4. **Комбинација на cv2.data.haarcascades и 'haarcascade_frontalface_default.xml'**:
    - Со комбинирање на патеката до директориумот (cv2.data.haarcascades) и името на XML фајлот ('haarcascade_frontalface_default.xml'), добивате целосна патека до фајлот за предобука. Оваа патека се користи за создавање на објектот CascadeClassifier.
5. **Создавање на објект CascadeClassifier**:
    - face_cascade е објект од класата CascadeClassifier кој е иницијализиран со предобука за детектирање на лица. Овој објект се користи за идентификување на лица во сликите со помош на алгоритамот на Haar каскади.

Како функционира целото:

- CascadeClassifier објектот (face_cascade) го учи како да препознава лица користејќи обучени податоци од XML фајлот.
- Кога ќе го примените објектот face_cascade на слика, тој го користи алгоритмот на Haar каскади за да ги детектира лицата и да ги врати координатите каде што се наоѓаат лицата во сликата.

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY): Овој ред ја конвертира оригиналната слика во сива слика. Претворањето во сива слика е потребно за детектирање на лица, бидејќи Haar каскадите работат со сиви слики.

1. cv2.cvtColor:
    - cv2.cvtColor е функција од библиотеката OpenCV која се користи за конверзија на слики од еден цветен простор во друг. Оваа функција може да се користи за различни видови на конверзии, како што се конверзија на боја во сиво, RGB во HSV и слично.
2. image:
    - image е променлива која ја содржи сликата која е прочитана од диск или генерирана од друг извор. Во овој контекст, тоа е слика во боја која е прочитана со cv2.imread.
3. cv2.COLOR_BGR2GRAY:
    - Ова е код за конверзија на боја кој го користите за да ја претворите сликата од боја (BGR) во сива слика (GRAYSCALE).

        - BGR: Блу, зелена, црвена (најчесто бојата на сликите во OpenCV е во BGR формат).
        - GRAY: Сива слика која има само едно ниво на интензитет, а не три канали на боја (червен, зелен и син).

4. gray_image:
    - Оваа променлива ќе ја чува резултантната сива слика по конверзијата. Секој пиксел во gray_image ќе има вредност која претставува интензитет на сивата боја, од 0 (црно) до 255 (бело).

Како работи целото:

- Конверзијата на боја во сиво: Кога конвертирате слика од боја во сива, сите боени информации се губат, а сликата се претвора во едноканална слика. Ова е полезно за многу алгоритми на обработка на слики и компјутерска визија, бидејќи работата со сива слика е поедноставна и бара помалку ресурси во однос на боени слики.

faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)): Овој ред користи face_cascade објект за детектирање на лица во сивата слика.

1. face_cascade:
    - face_cascade е објект на класата CascadeClassifier кој е иницијализиран со предобука (pre-trained model) за детектирање на лица. Овој објект ја користи Haar каскадната техника за препознавање на лица.
2. detectMultiScale:
    - detectMultiScale е метода на објектот CascadeClassifier која ја користи за детектирање на објекти (во овој случај, лица) во слика. Таа враќа листа на правоаголници (bounding boxes) каде што се наоѓаат објектите на сликата.
3. gray_image:
    - gray_image е сликата во сиво која ја добивме преку конверзијата од боја во сива. Оваа слика се користи за детектирање на лица бидејќи многу алгоритми на Haar каскади функционираат подобро со сиви слики.
4. scaleFactor=1.1:
    - scaleFactor е параметар кој го контролира како се скалира сликата во различни размери за време на детекцијата. Вредноста 1.1 значи дека сликата ќе се зголемува за 10% (1.1x) на секој чекор. Поголема вредност на scaleFactor ќе го намали бројот на скалирања и може да го зголеми времето на детекција, но може да ја намали прецизноста на детекцијата.
5. minNeighbors=5:
    - minNeighbors е параметар кој одредува колку соседни правоаголници треба да се пронајдат за да се смета дека едно лице е успешно детектирано. Поголема вредност ќе резултира со помалку лажни

позитиви, но може да ја намали чувствителноста. Вредноста 5 е компромис помеѓу лажни позитиви и пропуштање на вистински лица.

6. minSize=(30, 30):
    - minSize е параметар кој одредува минималната големина на објектите кои се детектираат. Оваа вредност го контролира минималниот размер на правоаголникот што се користи за детектирање на лице. Во овој случај, само објекти со големина најмалку 30x30 пиксели ќе се сметаат за лица.

Како функционира целото:

- detectMultiScale методот ги анализира сликата (gray_image) со користење на предобуката (Haar cascade) (face_cascade) и ја враќа листа на координати на правоаголници кои ги опфаќаат лицата пронајдени во сликата.
- Овие координати се користат за натамошно процесирање, како што е замаглување на лицата, означување на лицата на сликата, итн.

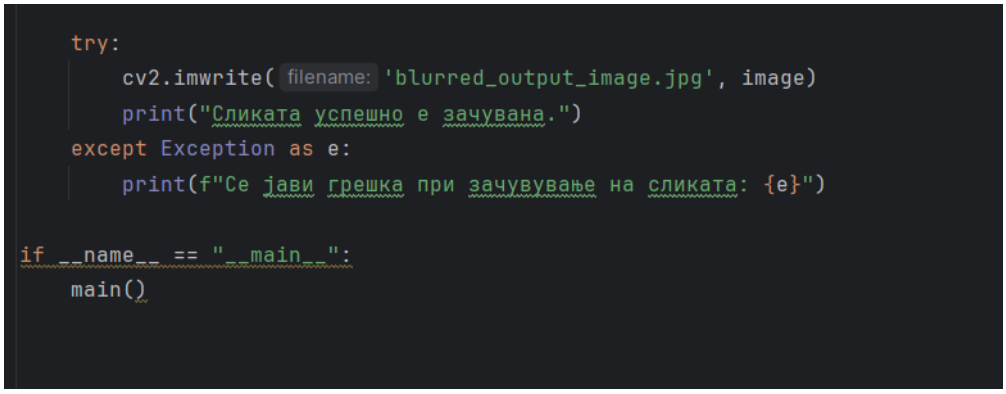
- **print("Избери тип на замаглување:")**: Испечати порака за избор на тип на замаглување.
- **print("1: Gaussian Blur")**: Испечати опција за Gaussian Blur.
- **print("2: Pixelated Blur")**: Испечати опција за Pixelated Blur.
- **choice = input("Внеси го бројот на твојот избор: ")**: Прими влез од корисникот за изборот на типот на замаглување.

- **for (x, y, w, h) in faces:**: Проаѓа низ сите детектирани лица. faces е листа од четворки (x, y, w, h) каде што:
- **x**: X координатата на горниот лев агол на лицето.
- **y**: Y координатата на горниот лев агол на лицето.
- **w**: Ширината на лицето.
- **h**: Висината на лицето.
- **if choice == '1':**: Ако корисникот избрал Gaussian Blur, применува apply_gaussian_blur на лицето.
- **elif choice == '2':**: Ако корисникот избрал Pixelated Blur, применува apply_pixelated_blur на лицето.
- **else:**: Ако изборот не е ни 1 ни 2, печати порака за невалиден избор и прекинува извршување на функцијата.

cv2.imshow('Blurred Faces', image): Прикажува сликата со замаглените лица во нов прозорец наречен "Blurred Faces"

cv2.waitKey(0): Чека корисникот да притисне копче. Ова го задржува прозорецот отворен додека корисникот не го затвори.

cv2.destroyAllWindows(): Ги затвора сите отворени прозорци по притискањето на копчето.



try:

- Овој ред започнува блок за обработка на исклучоци (грешки). Сето што е напишано под try ќе се обиде да се изврши. Ако се појави грешка во било кој од овие редови, програмата ќе премине на блокот except за да се справи со грешката.

cv2.imwrite('blurred_output_image.jpg', image)

- Овој ред се обидува да ја зачува сликата image во фајлот со име blurred_output_image.jpg. Функцијата cv2.imwrite од библиотеката OpenCV е користена за да се напише слика на диск. Првиот аргумент е патеката до фајлот и името на фајлот, а вториот аргумент е сликата што ќе се зачува.

print("Сликата успешно е зачувана.")

- Ако сликата успешно се зачува без грешки, ќе се изврши оваа линија. Таа ќе испечати порака на конзолата која информира дека сликата е успешно зачувана.

except Exception as e:

- Овој ред започнува блок за обработка на исклучоци. Ако се појави грешка во блокот try, ќе се префрли на овој дел од кодот. Exception as e значи дека се фаќа секоја грешка која е исклучок (извонредна ситуација) и таа ќе се чува во променливата e.

print(f"Се јави грешка при зачувување на сликата: {e}")

- Ако се појави грешка при зачувување на сликата, оваа линија ќе испечати порака која вклучува опис на грешката (e). Пораката ќе ја информира корисникот дека нешто тргнало наопаку при обидот за зачувување на сликата и ќе ја прикаже деталната грешка која се случила.

Како работи целото заедно:

- Кодот се обидува да зачува слика на диск.
- Ако се појави грешка (на пример, ако не можете да пристапите до дискот или ако името на фајлот е некоректно), блокот except ќе ја фати грешката и ќе ја испечати пораката со деталите за грешката.
- Ако не се појават грешки, ќе се испечати порака дека сликата е успешно зачувана.

Овој механизам на обработка на исклучоци помага во управување со грешки и подобрување на стабилноста на програмата.

if __name__ == "__main__":: Овој услов проверува дали скриптата се извршува директно (а не импортирана како модул). Ако е така, ќе ја повика функцијата main().

main(): Ја повикува главната функција за извршување на целиот код.

Како изгледа конзолата во оваа демо апликација:

```
Избери тип на замаглување:
1: Gaussian Blur
2: Pixelated Blur
Внеси го бројот на твојот избор: 2
Сликата успешно е зачувана.

Process finished with exit code 0
```

Како изгледа конзолата при успешно извршување на демо апликацијата.

```
Избери тип на замаглување:
1: Gaussian Blur
2: Pixelated Blur
Внеси го бројот на твојот избор: 8
Невалиден избор!

Process finished with exit code 0
```

Што ќе се испише доколку се внесе некој друг број освен 1 или 2 како избор за метод на имплементација.

```
C:\Users\PC\PycharmProjects\domasna4\.venv\Scripts\python.exe C:\Users\PC\PycharmProjects\seminarska3\demo_app.py
Грешка: Не може да се вчита сликата.
[ WARN:0@0.088] global loadsave.cpp:248 cv::findDecoder imread_('inputt.png'): can't open/read file: check file path/integrity

Process finished with exit code 0
```

Како изгледа конзолата кога не може да ја пронајде бараната слика.

Во продолжение уште неколку примери:

Најлево е оригиналот, во средина е применет гаусов метод, а десно е пикселизираниот.

Најлево е оригиналот, во средина е применет гаусов метод, а десно е пикселизираниот.

Проверка дека гаусовиот и пикселовиот метод, соодветно, работат и кога на истата слика има повеќе лица