**СЕМИНАРСКА РАБОТА**

**ПО**

**ДИГИТАЛНО**

**ПРОЦЕСИРАЊЕ НА СЛИКА**



Тема **:**

**Image Inpainting**

Ментор**: Ивица Димитриовски** Студент**: Марија Цветаноска**

Индекс: **223109**

Универзитет ,,Св.Кирил и Методиј” – Скопје, ФИНКИ

**СЕПТЕМВРИ 2024**

**Содржина**

1. Основни компоненти............................................................3

2. Функции и објаснување........................................................3

3. load\_image(image\_path)……..................................................3

**4.**create\_automated\_mask (image)**...............................................4**

**5.** load\_external\_mask(mask\_path, image\_shape).......................5

6. apply\_inpainting(image, mask, method)..................................6

7.save\_and\_display\_images

(original, mask, inpainted\_images)……………………………...7

8.Главната функција main().......................................................9

9. Алгоритмите за Inpainting......................................................10

10.Telnea алгоритам....................................................................10

11. Navier-Stokes алгоритам......................................................13

12. Заклучок и слики...................................................................16

**Основни компоненти**

* **Inpainting** e техника која се користи за пополнување на оштетени или недостасувачки делови од сликата со информации од околните пиксели, така што сликата изгледа природно.

Во овој код, користиме две главни методи за inpainting:

* **Telea алгоритам**(cv2.INPAINT\_TELEA): Овој алгоритам користи метод на **fast marching**  кој работи со брзина и дава добри резултати за мали и тенки оштетени области.
* **Navier-Stokes алгоритам**(cv2.INPAINT\_NS): Овој алгоритам се базира на динамика на флуиди, и е подобар за поголеми оштетени области, но може да биде побавен во процесирањето.

**Функции и објаснување**

**1. load\_image(image\_path)**

Опис:

Оваа функција е одговорна за вчитување на сликата од дадена патека.

Клучни моменти:

**cv2.imread():** Оваа функција од OpenCV ја вчитува сликата во меморија како матрица (низови на пиксели). Патеката на сликата се предава како аргумент. Доколку сликата не е пронајдена, функцијата враќа None.

Грешка ако не постои сликата: Доколку не се најде сликата на зададената патека, се фрла грешка со raise FileNotFoundError. Ова е корисно за да знаеме веднаш ако нешто не е во ред со патеката на сликата.

Код:

**def load\_image(image\_path):**

**image = cv2.imread(image\_path)**

**if image is None:**

**raise FileNotFoundError(f"Image not found at {image\_path}")**

**return image**

Како работи:

Оваа функција се грижи за валидно вчитување на сликата од дискот и подготвување за понатамошна обработка. Ако сликата не е пронајдена, веднаш известува за тоа.

**2. create\_mask(image)**

**Цел:**

Функцијата create\_automated\_mask создава бинарна маска за покривање на оштетени или бели области во сликата користејќи автоматски техники. Овој пристап е корисен кога областите што сакате да ги маскирате се визуелно различни од остатокот на сликата.

**Чекори:**

**Конвертирање во Грејски Скала:**

**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

Зошто: Конвертирањето на сликата во грејска скала го поедноставува процесот бидејќи го намалува бројот на канали на еден, што ја прави полесно примената на прагот.

**Како: cv2.cvtColor ја конвертира сликата од BGR (боја) во грејска скала.**

**Праговање:**

**\_, binary\_mask = cv2.threshold(gray, lower\_thresh, upper\_thresh, cv2.THRESH\_BINARY)**

Зошто: Прагувањето помага во создавањето бинарна слика каде пикселните вредности се или 0 или 255, што го олеснува разликувањето на областите на интерес (оштетени области) од остатокот.

**Како: cv2.threshold применува праг на грејската слика:**

Пикселите со вредности над lower\_thresh и под upper\_thresh се поставуваат на 255 (бела). Пикселите надвор од оваа граница се поставуваат на 0 (црна).

**Наоѓање Контури:**

**contours, \_ = cv2.findContours(binary\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)**

Зошто: Наоѓањето контури помага во идентификување на границите на поврзани компоненти (оштетени области) во бинарната маска.

**Како: cv2.findContours ги детектира контурите во бинарната слика:**

**cv2.RETR\_EXTERNAL ги враќа само надворешните контури.**

**cv2.CHAIN\_APPROX\_SIMPLE го компресира хоризонталните, вертикалните и дијагоналните сегменти во нивните краеви.**

**Цртање Контури на Маската:**

**mask = np.zeros\_like(binary\_mask)**

**cv2.drawContours(mask, contours, -1, 255, thickness=cv2.FILLED)**

**Зошто: Цртањето на контурите на нова маска помага да се создаде цврста област каде оштетените региони се.**

**Како: cv2.drawContours ги црта детектираните контури на празна маска:**

-1 укажува дека сите контури треба да се нацртани. 255 ги пополнува контурите со бела боја. thickness=cv2.FILLED ги пополнува сите области внатре во контурите.

**Резиме:**

Конвертирај ја сликата во грејска скала за да го поедноставиш процесот на обработка. Применувај прагување на грејската слика за да создадеш бинарна маска која ја истакнува потенцијалните оштетени области. Најди ги контурите во бинарната маска за да ги детектираш границите на оштетените региони. Цртај ги контурите на новата маска за да покриеш оштетените области со бела боја. Оваа функција е дизајнирана да се справи со слики каде оштетените области се визуелно различни од остатокот на сликата, што овозможува автоматско создавање на маска.

**3. load\_external\_mask(mask\_path, image\_shape)**

**Опис:**

Ако имате готова маска зачувана како слика (на пример, слика што означува оштетени делови со бела боја), оваа функција ја вчитува и ја прилагодува нејзината големина да одговара на големината на оригиналната слика.

**Клучни моменти:**

**cv2.imread():** Оваа функција се користи за вчитување на маската. Ова мора да биде слика во едноканална (ч/б) форма, што значи дека треба да биде зачувана како црно-бела слика.

**cv2.resize():** Ако маската не одговара на големината на оригиналната слика, оваа функција ја менува нејзината големина, за да можеме да ја користиме при обработката.

**Код:**

**def load\_external\_mask(mask\_path, image\_shape):**

**mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)**

**if mask is None:**

**raise FileNotFoundError(f"Mask not found at {mask\_path}")**

**mask\_resized = cv2.resize(mask, (image\_shape[1], image\_shape[0]))**

**return mask\_resized**

Како работи:

Оваа функција ни овозможува да користиме надворешни, претходно подготвени маски. Маската е прилагодена на големината на сликата за да може правилно да се примени inpainting.

**4. apply\_inpainting(image, mask, method)**

**Опис:**

Оваа функција ја применува inpainting техниката врз сликата користејќи го избраниот метод (Telea или Navier-Stokes).

**Клучни моменти:**

cv2.inpaint(): Оваа функција е клучна за inpainting. Таа користи алгоритам кој врз основа на маската ги пополнува оштетените делови од сликата.

**Методи за inpainting:**

**cv2.INPAINT\_TELEA:** Ова е Telea методот, кој користи брз марширачки метод за пополнување. Најдобар е за мали и тесни оштетени делови.

**cv2.INPAINT\_NS:** Ова е Navier-Stokes методот, кој е заснован на решавање на флуидни динамики. Подобар е за поголеми оштетувања.

**Код:**

**def apply\_inpainting(image, mask, method):**

**inpaint\_radius = 3 # Колку пиксели околу оштетената област да се земат предвид**

**if method == 'telea':**

**return cv2.inpaint(image, mask, inpaint\_radius, cv2.INPAINT\_TELEA)**

**elif method == 'ns':**

**return cv2.inpaint(image, mask, inpaint\_radius, cv2.INPAINT\_NS)**

**else:**

**raise ValueError("Invalid inpainting method. Choose 'telea' or 'ns'.")**

**Како работи:**

Во зависност од избраниот метод, алгоритмот ќе ги пополни белите области на маската. Telea работи брзо и е соодветен за помали оштетени делови, додека Navier-Stokes е соодветен за поголеми, но е посложен и побавен.

**5. save\_and\_display\_images(original, mask, inpainted\_images)**

**Опис:**

Оваа функција ги прикажува оригиналната слика, маската, и резултатите од inpainting за секој метод. Дополнително, резултатите се зачувуваат како нови слики.

**Клучни моменти:**

**cv2.imshow():** Оваа функција ги прикажува сликите во нови прозорци. Корисникот може да ги види оригиналната слика, маската и резултатите од различните алгоритми.

**cv2.imwrite():** Резултатите се зачувуваат во нови фајлови, така што корисникот може да ги користи или анализира подоцна.

**Код:**

**def save\_and\_display\_images(original, mask, inpainted\_images):**

**cv2.imshow('Original Image', original)**

**cv2.imshow('Mask', mask)**

**for method\_name, img in inpainted\_images.items():**

**cv2.imshow(f'Inpainted Image - {method\_name}', img)**

**output\_path = f'C:/Users/user/Downloads/inpainted\_image\_{method\_name}.jpeg'**

**cv2.imwrite(output\_path, img)**

**print(f"Image saved to {output\_path}")**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

Како работи:

Оваа функција ги прикажува резултатите од секој алгоритам за inpainting, што ни овозможува да ги споредиме различните пристапи. Исто така, резултатите се зачувуваат на диск за подоцнежна анализа.

**6. Главната функција main()**

**Опис:**

Оваа функција ги комбинира сите претходни функции и ги организира чекорите за вчитување на сликата, креирање или вчитување маска, примена на inpainting, и прикажување и зачувување на резултатите.

**Код:**

**def main():**

**image\_path = 'C:/Users/user/Downloads/slika1.jpeg'**

**mask\_path = 'C:/Users/user/Downloads/maska1.jpeg'**

**# Вчитување на оригиналната слика**

**image = load\_image(image\_path)**

**# Креирање или вчитување на маска**

**use\_external\_mask = False # True ако користиме надворешна маска**

**if use\_external\_mask:**

**mask = load\_external\_mask(mask\_path, image.shape)**

**else:**

**mask = create\_mask(image)**

**# Примена на inpainting методи**

**inpainted\_telea = apply\_inpainting(image, mask, 'telea')**

**inpainted\_ns = apply\_inpainting(image, mask, 'ns')**

**# Прикажување и зачувување на резултатите**

**inpainted\_images = {'telea': inpainted\_telea, 'ns': inpainted\_ns}**

**save\_and\_display\_images(image, mask, inpainted\_images)**

**Како работи:**

Главниот дел од програмата: Ја вчитува сликата, ја подготвува маската (рачно креирана или надворешна), го применува inpainting со двата алгоритми и ги прикажува и зачувува резултатите.

**3. Алгоритмите за Inpainting**

**1. Telea Method (cv2.INPAINT\_TELEA)**

Овој алгоритам користи Fast Marching Method (FMM), кој работи со брзина и ефикасност. Тој пополнува малите и тесни оштетени делови базирајќи се на околните пиксели.

Како работи: Алгоритамот прво ја обработува границата на оштетениот дел и постепено се шири кон внатрешноста, користејќи информации од соседните пиксели за пополнување.

**2. Navier-Stokes Method (cv2.INPAINT\_NS)**

Овој метод користи математички модел базиран на динамика на флуиди. Подобар е за поголеми области, бидејќи го зема предвид движењето на пикселите во сликата и го пополнува оштетениот дел со информации кои изгледаат природно.

Како работи: Алгоритамот ги решава едначините на флуиди за да пополни поголеми празнини, но е побавен во споредба со Telea.

**Telnea алгоритам**

Основи на Telea алгоритмот

Telea алгоритмот е базиран на техниката на блиско-поле дифузија (анг. fast marching method). Главната идеја е да се користи информации од соседните делови на сликата за да се пополнат оштетените пиксели.

Клучни чекори:

Маркирање на оштетената област:

Пред да започне алгоритмот, потребно е да се има маска која ја означува оштетената област (каде пикселите треба да се пополнат). Во оваа маска, белите пиксели (со вредност 255) означуваат оштетени делови, додека црните пиксели (со вредност 0) се неоштетените делови на сликата.

Проширување на границата на оштетените пиксели:

Алгоритмот започнува со следење на границите на оштетените пиксели (односно областите каде оштетените делови се блиски до неоштетените). Ова е основата за да се одлучи кои информации да се користат за пополнување.

Процес на дифузија (ширење):

Telea алгоритмот го користи fast marching method, кој е техника за брзо ширење на информациите низ сликата. Ова значи дека алгоритмот ја проширува информацијата од неоштетените делови на сликата кон оштетените области, постепено пополнувајќи ги пикселите еден по еден. За секој оштетен пиксел, се пресметува нова вредност базирана на соседните неоштетени пиксели, користејќи тежини кои се пропорционални на нивната дистанца од оштетената област.

Интерполација на пикселите:

Кога се пополнува оштетен пиксел, Telea користи интерполација за да пресмета соодветна вредност на пикселот. Оваа интерполација користи информации од соседните пиксели со слична боја и текстура, за да се осигури дека новиот пиксел изгледа природно и непрекинато се вклопува со околината.

Ширење на текстурата и структурата:

Еден од клучните моменти во Telea алгоритмот е тоа што успешно ги задржува текстурата и структурата на сликата. Оштетените области се пополнуваат на начин што продолжува визуелниот континуитет на линиите и текстурите на сликата.

Формално, процесот на Telea алгоритмот може да се сведе на следниве чекори:

Прво, ги идентификува границите на оштетената област.

Потоа, користи информации од соседните пиксели, вклучувајќи ја нивната боја, интензитет и текстура, за да се пополни оштетената област.

Оваа информација се комбинира со математички методи за дифузија, при што се шири вредноста од здравите пиксели кон оштетените.

Пополнувањето се изведува постепено, почнувајќи од рабовите на оштетените области кон внатрешноста.

**Предности на Telea алгоритмот**

**Брзина:**

Telea е многу брз метод за inpainting благодарение на fast marching method, што овозможува брзо ширење на информациите низ сликата. Ова го прави погоден за апликации каде брзината е клучен фактор.

**Прецизност за мали оштетувања:**

Овој алгоритам најдобро функционира за мали или тенки оштетувања на сликата, каде што околната структура и текстура лесно се пренесуваат преку дифузија.

**Задржување на текстура и детали:**

Telea добро го зачувува визуелниот континуитет на сликата. На пример, ако оштетениот дел вклучува линии или рабови, алгоритмот ќе се обиде да ги продолжи тие линии во пополнетата област.

**Недостатоци на Telea алгоритмот**

**Проблеми со големи оштетувања:**

Кога оштетената област е голема, Telea алгоритмот може да има потешкотии да ја пополни соодветно, бидејќи е базиран на близина и директна интерполација од соседни пиксели. Големите оштетени области честопати бараат повеќе информации за пополнување.

**Недостаток на сложени детали:**

Овој алгоритам може да не биде многу успешен во области каде што текстурата и деталите на сликата се многу сложени или каде што има многу остри рабови. Во такви случаи, пополнувањето може да изгледа неприродно.

Зошто е популарен?

Telea алгоритмот е многу популарен поради својата едноставност и брзина. Тој функционира многу добро за слики со мали оштетувања и нуди задоволителни резултати за повеќето апликации, како на пример пополнување на мали грешки, отстранување на водени жигови, или поправање на оштетени фотографии.

Пример со Telea алгоритмот во кодот:

Во претходниот пример со OpenCV, Telea алгоритмот беше користен на следниот начин:

inpainted\_telea = cv2.inpaint(image, mask, inpaint\_radius, cv2.INPAINT\_TELEA)

Објаснување:

image: Ова е оригиналната слика што треба да се пополни.

mask: Ова е маската што ја означува оштетената област.

inpaint\_radius: Ова е радиусот околу оштетените пиксели, кој алгоритмот го користи за да ги земе предвид соседните пиксели.

**Navier-Stokes алгоритам**

Navier-Stokes image inpainting е техника за поправање на оштетени или недостасувачки делови на слика преку употреба на методи засновани на Navier-Stokes-овите равенки. Овој пристап се користи во дигитално обработување на слики, при што оштетените делови на сликата се пополнуваат на начин што го задржува природниот тек на информации во сликата, слично како што течностите течат според Navier-Stokes равенките. Еве како ова функционира подетално:

1. Основна идеја за image inpainting

Image inpainting (поправање на слика) е процес во кој недостасувачките или оштетените делови на сликата се пополнуваат, така што финалниот резултат изгледа како да е природен и кохерентен. Ова се постигнува преку продолжување на постојните текстури и граници во сликата низ недостасувачките делови. Целта е сликата да изгледа целосно, без видливи траги на оштетувања.

2. Употреба на Navier-Stokes равенките

Navier-Stokes-овите равенки ги опишуваат законите на движење на течностите, и заедно со други концепти како изохорна дифузија (дифузија со константна запремнина) се користат за да се пресмета како информации (боја, текстура) "течат" од недостасувачките делови на сликата кон околината.

3. Принципи на Navier-Stokes image inpainting

Navier-Stokes-овиот модел за image inpainting го симулира процесот на пренос на информации (боја и текстура) низ сликата слично на тоа како течност би се движела низ нејзините контури. Овој процес го задржува континуитетот на бојата и текстурите за да создаде природен премин во оштетените делови.

Основните принципи вклучуваат:

Континуитет на кривините: Контурите и кривините во сликата продолжуваат преку недостасувачките делови, како што течностите би продолжиле низ отворениот простор.

Дифузија на информации: Боите и текстурите се "прошируваат" низ недостасувачките делови, како што температурните или концентрациските полиња би се проширувале низ флуид според дифузионите равенки.

4. Навер-Стоуксов модел за inpainting

Navier-Stokes inpainting користи равенки кои ги одредуваат брзинските полиња во флуидот и го применуваат на сликата за пополнување на празните простори. Процесот обично се состои од неколку чекори:

Чекори:

Модел на континуум за граници и контура: Се идентификуваат кривините и контурата на сликата околу недостасувачкиот дел. Овие граници служат како услови за продолжување на информациите.

Пресметка на брзинско поле: Се користи Navier-Stokes-овиот модел за да се генерира брзинско поле кое одредува во кој правец и со која брзина треба да "течат" информациите (боја, текстура) низ оштетениот дел. Брзинското поле ја дефинира динамиката на пополнување.

Примена на дифузија: Брзината на пренос на информациите низ недостасувачкиот дел се заснова на принципите на изохорна дифузија, при што боите и текстурите "се шират" преку сликата на начин што одржува конзистентност.

Гранични услови: Како флуидот да се движи низ рамка, се одредуваат гранични услови, со што се избегнува неестетско "излевање" на информации во деловите каде што не се потребни.

5. Примена и предности

Navier-Stokes image inpainting се користи за реконструкција на оштетени слики во многу области:

Реставрација на уметнички дела: Применето за поправање на оштетени или изгубени делови на дигитални верзии на слики и уметнички дела.

Филмска продукција и визуелни ефекти: Поправка на оштетени рамки од стари филмови.

Медицински слики: Поправање на оштетени делови од медицински слики (како MRI) каде што недостасува дел од информациите.

6. Недостатоци и предизвици

Комплексност на алгоритмите: Решавањето на Navier-Stokes-овите равенки е пресметковно скапо и бара посериозни ресурси за реално време имплементации.

Гранични артефакти: Понекогаш може да се појават артефакти на границите помеѓу пополнетите и оригиналните делови.

7. Заклучок

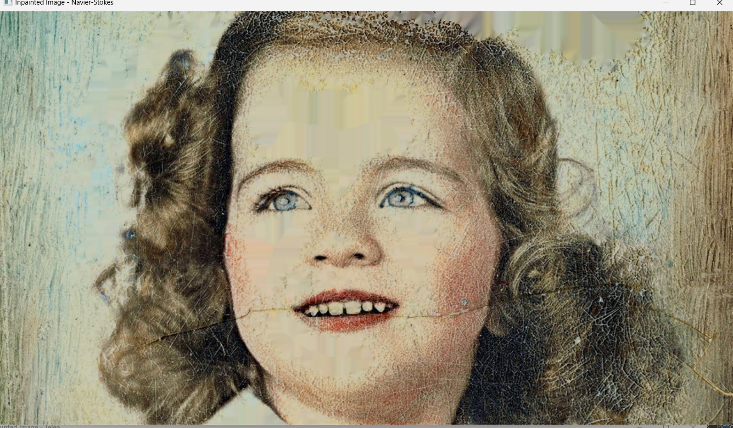
Navier-Stokes inpainting е моќен алгоритам за реконструкција на оштетени слики бидејќи имитира како флуидот би се движел преку границите на оштетениот дел. Овој пристап е особено корисен кога треба да се одржи природниот тек на контурата и текстурата на сликата, нудејќи автентични резултати во многу области.

Заклучок и слики

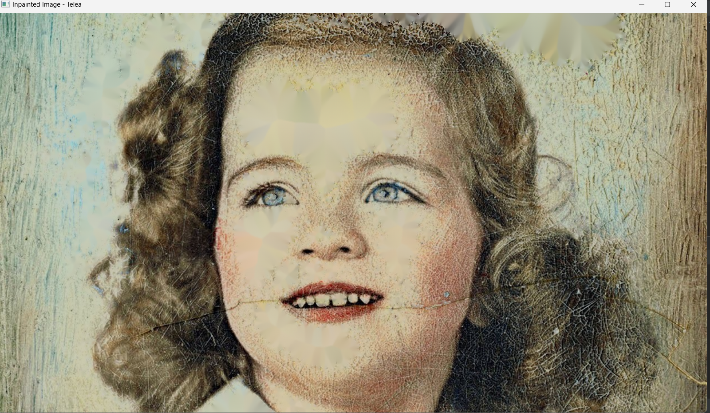
Кодот користи два различни алгоритми за пополнување на оштетени делови од сликата. Telea е брз и ефикасен за помали оштетувања, додека Navier-Stokes е посоодветен за поголеми оштетувања, но е побавен. Преку маската можеме да дефинираме кои делови од сликата ќе бидат обработени.



**Оригинална Слика**



**Слика Image inpainting-Navier-Stokes**



**Слика Image inpainting-T**



**Маска**