



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

Документација за семинарската работа по предметот
Дигитално процесирање на слика

Тема:

Алгоритми за промена на резолуција на слики

Изработил:

Сара Добревска, 221125

Содржина:

Вовед	3
Предуслови и користени технологии	3
Техники за зголемување на резолуција	4
Техники со супер резолуција	5
<i>Nearest-Neighbor Interpolation</i>	6
<i>Bilinear Interpolation</i>	7
<i>Bicubic Interpolation</i>	9
<i>Lanczos Resampling</i>	11
Техника со сложување на слики	13
<i>Астрофотографија</i>	15
Модели за машинско учење	16
<i>Конволуциони невронски мрежи (CNNs)</i>	16
<i>Генеративни Адверзаријални Мрежи (GANs)</i>	18
Проучување на алгоритми за зголемување на резолуцијата	20
EDSR (Enhanced Deep Super Resolution).....	20
ESRGAN (Enhanced Super-Resolution Generative Adversarial Network)	21
Зголемување на сликата со вештачка интелигенција	22
Заклучок.....	24
Референци	24

Вовед

Промената на резолуција на слики, или *image resampling*, е процес на прилагодување на големината на дигиталните слики со цел подобрување на нивниот квалитет или приспособување за различни намени. Во последните неколку години, оваа област на компјутерската визија бележи огромен напредок, благодарение на новите технологии и алгоритми. Овие достигнувања овозможуваат креирање на слики со повисок квалитет и повеќе детали, што е значајно за различни апликации како што се печатење, анализа на слики, медицинска дијагностика и дигитална уметност.

Алгоритмите за промена на резолуција на слики можат да бидат базирани на класични методи како што се најблиската соседство (*nearest neighbor*), линеарната интерполација (*bilinear interpolation*) и биквадратната интерполација (*bicubic interpolation*), како и на современи методи кои користат машинско учење и длабоко учење. Класичните методи се едноставни за имплементација и брзи за извршување, но често резултираат со загуба на квалитет и детали. Од друга страна, алгоритмите базирани на машинско учење можат да генерираат многу поквалитетни и пореалистични резултати, но се и многу посложени и побавни.

Целта на оваа семинарска работа е да ги опише и имплементира најчесто користените алгоритми за промена на резолуција на слики, како и да направи анализа на нивните перформанси во однос на квалитетот и брзината на извршување. Оваа анализа ќе овозможи подобро разбирање на предностите и недостатоците на секој од алгоритмите и ќе помогне при изборот на најсоодветниот метод за различни апликации.

Во следните делови од оваа семинарска работа, ќе се фокусираме на детално објаснување на принципите на функционирање на секој алгоритам, ќе презентираме примери на нивна имплементација и ќе направиме споредба на нивните резултати.

Предуслови и користени технологии

За започнување на проектот за промена на резолуција на слики, потребно е да имате инсталирано Python верзија 3.10.10 и PIP на вашиот систем. Потребни се следниве пакети и технологии:

- OpenCV – Open Source Computer Vision Library, претставува софтверска библиотека со отворен код за компјутерска визија, машинско учење и процесирање на слики. Овозможува голем избор на алатки и алгоритми за процесирање на слики и видеа и игра голема улога во операции во реално време, важни за современите системи. Во проектот оваа библиотека се користи за задачи како читање на слики и промена на нивната големина.

pip install opencv-python

- NumPy – Numerical Python, е моќна Python библиотека за нумеричко пресметување. Овозможува поддршка за големи, повеќедимензионални низи и матрици заедно со колекција на математички функции за ефикасна работа со овие низи. NumPy е

користена за задачи како нумерички симулации, анализа на податоци, линеарна алгебра, статистички пресметки и многу други. Во проектот се користи за вчитување на точки, креирање на низи и манипулирање со димензиите на низите.

pip install numpy

- Tkinter – стандардна Python библиотека за креирање на графички кориснички интерфејси (GUIs). Овозможува комплет од алатки за градење десктоп апликации со графички компоненти како копчиња, лабели, простор за текст и многу други. Вклучена е со стандардната библиотека на Python, така што немате потреба од посебна инсталација.
- PIL (Python Imaging Library) – библиотека за отворање, манипулација и зачувување на различни формати на слики. Овозможува основни задачи од областа на обработка на слики како промена на големина, сечење, ротирање, подобрување на квалитетот, примена на филтри и конвертирање на форматот на сликите. Развојот на оваа библиотека престанал со верзијата 1.1.7, но за да се обезбеди компатибилност со Python 3.x, била направена библиотеката Pillow, која е ажурирана и одржувана од заедницата.

pip install Pillow

- Random модулот – во Python овозможува функции за генерирање случајни броеви од различни типови, како цели броеви, float броеви и случаен избор од секвенци. Овој модул е дел од стандардната библиотека на Python и не бара дополнителна инсталација.

Овие библиотеки и технологии се основните алатки потребни за имплементација на алгоритмите за промена на резолуција на слики. Со нивна помош, ќе бидете способни да процесирате и манипулирате со слики, прилагодувајќи ја нивната резолуција според вашите потреби.

Техники за зголемување на резолуција

Алгоритмите за промена на резолуцијата на сликите, кои често се нарекуваат скалирање на сликата или зголемување на резолуцијата на сликата, вклучуваат техники кои го зголемуваат или намалуваат бројот на пиксели на сликата. Еве ги на кратко:

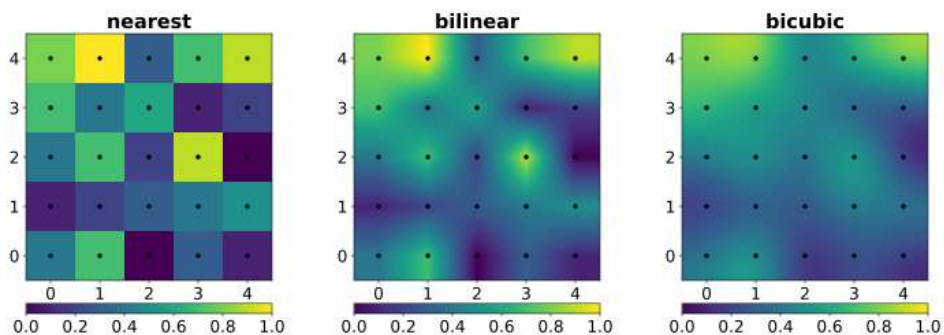
- **Техники со супер резолуција** (*Super-Resolution Techniques*): користи напредни алгоритми за компјутерска визија или машинско учење за интерполирање на информации за пиксели и додавање нови детали на сликата. Вообичаените алгоритми вклучуваат:

- **Интерполација на најблискиот сосед** (*Nearest-Neighbor Interpolation*): Едноставна и ефикасна.
 - **Билинеарна интерполација** (*Bilinear Interpolation*): Обезбедува прецизна проценка на пиксели.
 - **Бикубна интерполација** (*Bicubic Interpolation*): Напредна техника за подобра јасност.
 - **Ланцошева Интерполација** (*Lanczos Resampling*): Синусоидна функција што се користи за спојување на соседни пиксели при интерполацијата.
- **Сложување слики** (*Image Stacking*): комбинира повеќекратни експозиции зголемени до повисока резолуција за да го подобри конечниот квалитет на сликата.
 - **Модели за машинско учење** (*Machine learning models*): моделите како SRResNet можат да изведуваат супер-резолуција со висок квалитет со тренирање на вештачката интелигенција за подобрување на сликите.
 - **Проучување на алгоритми за зголемување на резолуцијата** (*Study of Upscaling Algorithms*): Алгоритмите како EDSR, ESPCN, FSRCNN и LAPSRN имаат уникатни карактеристики за зголемување на големината на сликите.
 - **Зголемување на сликата со вештачка интелигенција** (*AI Image Upscaling*): користи алгоритми за вештачка интелигенција за да ја подобри резолуцијата додека ги зачувува деталите.

Овие методи се клучни за различни апликации, вклучувајќи подобрување на фотографијата, медицински слики и системи за надзор.

Техники со супер резолуција

Супер-резолуцијата е процес кој овозможува зголемување на резолуцијата на слики, додавајќи нови детали и подобрувајќи ја нивната јасност. Оваа техника користи напредни алгоритми од областа на компјутерската визија и машинското учење за интерполација на пиксели и внесување на нови информации во сликата. Во продолжение на *Слика 1* може да ги видиме, а подолу се објаснети некои од најчесто користените алгоритми за супер-резолуција:




Слика 1.

Nearest-Neighbor Interpolation

Ова е наједноставниот и најефикасниот метод кој избира најблискиот пиксел за пополнување на новите пиксели. Иако е брз, резултира со блокови и остри рабови кои можат да изгледаат неестетски.

Еве како функционира:

1. *Разбирање на вредностите на пикселите:* Во дигиталните слики, секој пиксел има одредена локација дефинирана со неговите координати (x, y) и вредност на бојата (често претставена како RGB вредности).
2. *Фактор на скалирање:* Интерполацијата на најблискиот сосед обично се користи за зголемување на големината (зголемување на големината) на сликите. Факторот на скалирање одредува колку ќе бидат поголеми димензиите на новата слика во споредба со оригиналот.
3. *Пресметување нови димензии:* Ако оригиналната слика е со одредени димензии (ширина, висина). Ако треба да се зголеми резолуцијата на сликата за фактор 2,0 (што значи двојно зголемување на нејзината големина), новите димензии ќе бидат (ширина * 2, висина * 2).
4. *Процес на интерполација:* За секој пиксел на новата слика, неговата вредност на бојата се одредува врз основа на најблискиот пиксел на оригиналната слика. Поточно, за секој пиксел (x', y') на новата слика, се наоѓа соодветната локација на пикселот (x, y) во оригиналната слика. Вредноста на бојата на пикселот (x', y') на новата слика е поставена на вредноста на бојата на најблискиот пиксел (x, y) на оригиналната слика.
5. *Имплементација:*



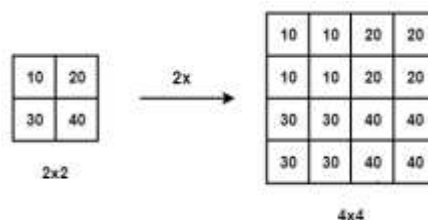
```

1  import cv2
2
3  # image
4  def nearest_neighbor_interpolation(image_path, scale_factor):
5      image = cv2.imread(image_path)
6      new_dimensions = (int(image.shape[1] * scale_factor), int(image.shape[0] * scale_factor))
7      upscaled_image = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_NEAREST)
8      cv2.imwrite('nearest_neighbor_upscaled.jpg', upscaled_image)
9
10 nearest_neighbor_interpolation(image_path='input_image.jpg', scale_factor=2.0)

```

Слика 2.

На Слика 2 може да го видиме кодот, преку кој сликата се вчитува од датотека користејќи `cv2.imread`. Со помош на `cv2.resize`, сликата се зголемува со зададен фактор на зголемување (`scale_factor`). Вградената функција `cv2.INTER_NEAREST` се користи за да се осигура дека секој нов пиксел во зголемената слика се пресметува со вредностите на најблискиот пиксел од оригиналната слика. На крај, зголемената слика се зачувува како JPEG датотека.



Слика 3.

На *Слика 3* го гледаме на едноставен начин како се движат пикселите по користењето на овој алгоритам.

6. Предности и недостатоци:

- Предности: Интерполацијата на најблискиот сосед е пресметковно ефикасна и лесна за имплементација. Ги зачувува острите рабови и деталите на сликата.
- Недостатоци: Зголемената слика може да изгледа како заматена, особено при скалирање според големи фактори, бидејќи не ги зема предвид вредностите на соседните пиксели за мазни транзиции.

Севкупно, интерполацијата на најблискиот сосед е погодна за сценарија каде што зачувувањето на острите рабови и минимизирањето на пресметковната комплексност се приоритети, но можеби нема да обезбеди најдобар визуелен квалитет во споредба со понапредните методи на интерполација како што се биланеарна или двокубна интерполација.



Слика 4.



Слика 5.

На *Слика 4* ја гледаме оригиналната слика, додека пак на *Слика 5*, сликата по веќе искористената функција за најблизок сосед со зголемување со фактор 2.

Bilinear Interpolation

Овој метод користи линеарна интерполација за да пресмета нови пиксели врз основа на тежински просек на четири најблиски пиксели. Обезбедува попрецизна проценка на пикселите и помазни резултати од најблискиот сосед.

Еве како функционира биланеарната интерполација:

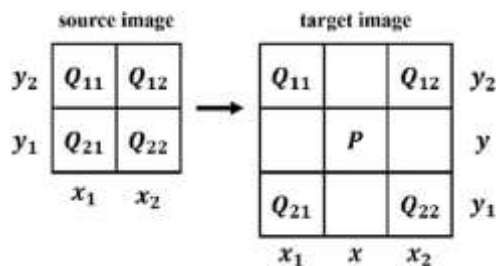
1. *Разбирање на координатите на пиксели:* Кај дигиталните слики, секој пиксел има специфични координати (x, y) и соодветна вредност на бојата.
2. *Фактор на скалирање и нови димензии:* Билинеарната интерполација обично се користи и за зголемување и за намалување на скалирањето на сликите. За зголемување на резолуцијата, да размислиме за зголемување на големината со фактор на скалирање од 2,0. Ова значи дека новите димензии на сликата ќе бидат (ширина * 2, висина * 2).

3. *Пресметување интерполирани вредности:* За секој пиксел (x', y') на новата слика, се пресметува неговата позиција во оригиналната слика. Ова вклучува одредување на соодветните координати со подвижна записка (x, y) во оригиналната слика, што не мора да биде совршено усогласено со позициите на пиксели со цели броеви.
4. *Одредување на вредностите на околните пиксели:* Се идентификуваат четирите најблиски пиксели на оригиналната слика што ги опкружуваат координатите со подвижна записка (x, y) . Овие пиксели обично се означени како (x_1, y_1) , (x_1, y_2) , (x_2, y_1) и (x_2, y_2) , каде што (x_1, y_1) е горниот лев агол на околните пиксели.
5. *Просечна пресметка:* Се пресметува просек на вредностите на бојата на овие четири пиксели за да ја одредите вредноста на бојата на пикселот (x', y') на новата слика. Тежините се одредуваат врз основа на растојанијата помеѓу (x', y') и околните координати на пиксели. Колку се поблиску координатите на пикселот до (x', y') , толку е поголема неговата тежина во просек.
6. *Имплементација:*



Слика 6.

На Слика 6 ја гледаме имплементацијата каде се чита од зададената патека користејќи `cv2.imread(image_path)`. Со помош на факторот за скалирање, кој го прима како влезен параметар, се пресметуваат новите димензии на сликата. Потоа, функцијата `cv2.resize()` се користи за да се изведе биланеарна интерполација на оригиналната слика до новите димензии, користејќи `cv2.INTER_LINEAR` како метод на интерполација. Конечниот резултат се зачувува како нова слика со име `'bilinear_upscaled.jpg'` преку `cv2.imwrite()`. На крај, функцијата `bilinear_interpolation()` се повикува со конкретна слика `('input2_image.jpg')` и фактор на скалирање од `2.0`, што резултира со зголемување на големината на сликата за двојно.



Слика 7.

На Слика 7, може да го видиме алгоритмот, односно како изгледаат пикселите пред и по искористувањето на функцијата.

7. Предности и недостатоци:

- Предности: Биланеарната интерполација произведува помазни транзиции и генерално подобар визуелен квалитет во споредба со интерполацијата на најблискиот сосед. Тоа

помага да се намалат артефактите, како што е алиасирањето што може да се случи со поедноставни методи.

- Недостатоци: Иако е подобра од најблискиот сосед, биланеарната интерполација сепак може да резултира со мало замаглување или губење на острината на сликите, особено при скалирање со големи фактори или кога има остри рабови или детали.



Слика 8.



Слика 9.

Преку следните две слики, Слика 8 и Слика 9 може да видиме пример како изгледа биланеарната интерполација. Зголемувањето е двојно, односно ширина*2 и висина*2.

Накратко, биланеарната интерполација е широко користен метод за промена на големината на сликата поради неговата рамнотежа помеѓу пресметковната ефикасност и подобрениот визуелен квалитет во споредба со интерполацијата на најблискиот сосед. Тој е ефикасен за широк опсег на фактори на скалирање и е погоден за многу апликации за обработка на слики.

Bicubic Interpolation

Напредна техника која користи кубична интерполација на 16 најблиски пиксели за да создаде нови пиксел вредности. Овој метод нуди подобра јасност и помазни резултати во споредба со билинеарната интерполација, правејќи ја сликата појасна и поприродна.

Еве како функционира:

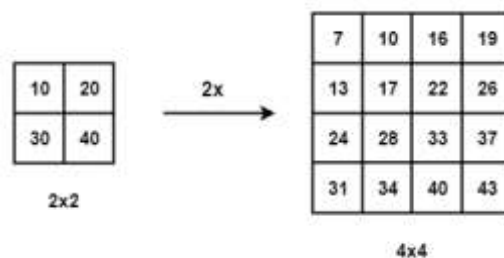
1. *Разбирање на координатите на пикселите:* Слично на другите методи на интерполација, бикубичната интерполација работи на координатите и вредностите на бојата на пикселите во дигиталните слики.
2. *Фактор на скалирање и нови димензии:* Бикубичната интерполација обично се користи и за зголемување и за намалување на скалирањето на сликите. За зголемување на резолуцијата, ако се земе фактор на скалирање од 2,0, новите димензии на сликата ќе бидат (ширина * 2, висина * 2).
3. *Избор на опкружувачки пиксели:* за секој пиксел (x' , y') на новата слика, се определуваат неговите соодветни координати со подвижна записка (x , y) во оригиналната слика. За разлика од биланеарната интерполација, која користи 2x2 мрежа на околните пиксели, двокубичната интерполација користи поголема 4x4 мрежа на околните пиксели за интерполација.

4. *Конструирање на интерполациониот полином:* Бикубната интерполација вклучува конструирање на кубен полином кој непречено се интерполира помеѓу 16-те најблиски пиксели (мрежа 4x4) во оригиналната слика. Овој полином обично се изведува со помош на Лагранжова интерполација или сличен метод.
5. *Пресметување вредности на пиксели:* Откако ќе се конструира интерполираниот полином за секој канал во боја (R, G, B), се одредува интерполираната вредност на бојата за пикселот во новата слика. Овој процес осигурува дека добиената слика има мазни транзиции и зачувува пофини детали во споредба со поедноставните методи на интерполација.
6. *Имплементација:*



Слика 10.

Преку Слика 10 може да забележиме дека прво сликата се чита од зададената патека со помош на `cv2.imread(image_path)`. Со помош на факторот за скалирање, кој се дефинира како влезен параметар, се пресметуваат новите димензии на сликата користејќи ги димензиите на оригиналната слика и факторот за скалирање. Потоа, функцијата `cv2.resize()` се користи за да се изведе бикубична интерполација на сликата до новите димензии, користејќи `cv2.INTER_CUBIC` како метод на интерполација. Конечниот резултат се зачувува како нова слика со име `'bicubic_upscaled.jpg'` преку функцијата `cv2.imwrite()`. На крај, функцијата `bicubic_interpolation()` се повикува со конкретна слика и фактор на скалирање од `0.5`, што ја намалува големината на сликата за половина.



Слика 11.

Овој алгоритам може да се користи и за намалување и зголемување, функцијата која е имплементирана е за двојно намалување, додека пак преку Слика 11 може да видиме како тоа функционира при двојно зголемување.

7. Предности и недостатоци:

- Предности: Бикубната интерполација дава висококвалитетни резултати со помазни транзиции и подобро зачувување на деталите во споредба со најблискиот сосед и биланеарната интерполација. Тој е особено ефикасен за зголемување на резолуцијата на сликите и широко се користи во професионални апликации за обработка на слики.

- Недостатоци: Бикубната интерполација може да биде пресметковно поинтензивна во споредба со поедноставните методи, што може да влијае на перформансите во апликациите во реално време. Исто така, може да внесе мало замаглување или артефакти на свонење, особено околу остри рабови.



Слика 12.



Слика 13.

Преку Слика 12 и Слика 13 може да видиме како бикубичната интерполација функционира доколку ја користеме за намалување со фактор 0.5, односно двојно намалување.

Накратко, бикубичната интерполација е моќна техника за промена на големината на сликата која воспоставува рамнотежа помеѓу сложеноста на компјутерот и визуелниот квалитет. Добро е прилагоден за сценарија каде што зачувувањето на деталите на сликата и постигнувањето непречени транзиции се клучни, како што се во дигиталната фотографија, медицинските слики и компјутерската графика.

Lanczos Resampling

Ланцошовата интерполација се базира на Ланцошовата функција, која е модифицирана синусна функција. Таа е дефинирана според равенката на Слика 14:

$$L(x) = \begin{cases} 1 & \text{if } x = 0, \\ \frac{a \sin(\pi x) \sin(\pi x/a)}{\pi^2 x^2} & \text{if } -a \leq x < a \text{ and } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Слика 14.

Параметарот a е позитивен цел број, обично 2 или 3, кој ја одредува големината на јадрото. Јадрото има $2a - 1$ лобуси: позитивен во центарот и -1 наизменични негативни и позитивни лобуси на секоја страна.

Ланцошовата интерполација работи со употреба на Ланцошовата функција за да ги пресмета вредностите на пикселите во околината на пикселот што се интерполира. Колку е поголем параметарот, толку повеќе соседни пиксели ќе бидат земени предвид во интерполацијата. Типично, се зема како 2 или 3.

Предности:

1. *Висок квалитет на сликите:* Ланцошовата интерполација обезбедува висок квалитет на интерполираните слики, со добро зачувување на детали и остри рабови.
2. *Намалени артефакти:* Овој метод има тенденција да ги намалува артефактите како блур, кои се чести при други видови интерполација.

Недостатоци:

1. *Комплексност и брзина:* Ланцошовата интерполација е покомплексна и побавна во споредба со едноставни методи како билинеарна или бикубична интерполација, поради потребата за пресметка на тежините за повеќе пиксели.
2. *Гранични ефекти:* Понекогаш, кога се користи поголем прозорец, може да се појават гранични ефекти, каде што интерполираните пиксели на рабовите на сликата можат да бидат неточни.

Имплементација:

```

lanczos.py
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Image
def lanczos_resample_image(image, scale_factor, sfs):
    def sinc(x):
        return np.sinc(x / np.pi)

    def lanczos_weight(x, a):
        if x == 0:
            return 1.0
        elif -a <= x <= a:
            return sinc(x) * sinc(x / a)
        else:
            return 0.0

    height, width = image.shape
    new_height = int(height * scale_factor)
    new_width = int(width * scale_factor)
    resampled_image = np.zeros((new_height, new_width), dtype=np.float32)

    for i in range(new_height):
        for j in range(new_width):
            x = j / scale_factor
            y = i / scale_factor
            x_int = int(np.floor(x))
            y_int = int(np.floor(y))
            total_weight = 0.0
            interpolated_value = 0.0

```

Слика 15.

```

        for n in range(x_int - a, x_int + a + 1, x_int + a):
            for m in range(y_int - a, y_int + a + 1, y_int + a):
                if 0 <= m <= height and 0 <= n <= width:
                    weight = lanczos_weight(x - n, a) * lanczos_weight(y - m, a)
                    interpolated_value += image[n, m] * weight
                    total_weight += weight

        resampled_image[i, j] = interpolated_value / total_weight if total_weight != 0.0 else image[x_int, y_int]

    return resampled_image.astype(np.float32)

original_image = cv2.imread('images/lena.jpg', cv2.IMREAD_GRAYSCALE)
if original_image is None:
    original_image = np.random.rand(300, 300) * 255

plt.figure(figsize=(10, 10))
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.show()

scale_factor = 2
resampled_image = lanczos_resample_image(original_image, scale_factor)

plt.figure(figsize=(10, 10))
plt.imshow(resampled_image, cmap='gray')
plt.title('Resampled Image (Lanczos Resampling, scale_factor=2)')
plt.axis('off')
plt.show()

```

Слика 16.

На Слика 15 и Слика 16 може да го видиме имплементиран овој алгоритам. Главната функција ***lanczos_resample*** прима влезна слика и фактор за зголемување на резолуцијата, и го применува Ланцошовото ресемплирање за да се генерира новата резултантна слика. Пресметките се изведуваат во двоен јазол, каде што се пресметуваат тежините на соседните пиксели и се добива интерполираната вредност.

Овој метод не само што го подобрува квалитетот на сликата туку и го чува нивниот визуелен интегритет, што го прави моќен алат за напредна обработка на слики во различни домени.

Ланцошовата интерполација е широко користена во обработка на слики и видео, особено за:

- Зголемување на резолуцијата на фотографии и слики.
- Зголемување на резолуцијата на видеа без значајно губење на квалитет.
- Примена во дигитални телевизиски и филмски продукции за постигнување на висока визуелна фиделност.

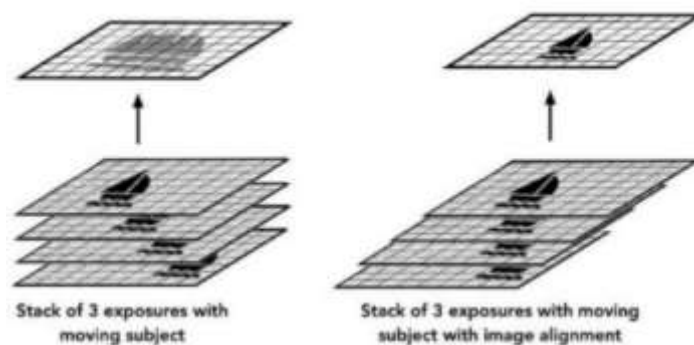


Слика 17.

На Слика 17 може да видиме пример за слика пред и после користење на Lanczos resampling.

Техника со сложување на слики

Техниката на "**Image Stacking**" или "стакнување/сложување на фотографии" се користи за да се зголеми резолуцијата, острината или да се намали шумот. Ова се прави со комбинирање повеќе слики од истата сцена, обично заснети со различни параметри како што се експозиција или фокус. Процесот на стакнување обично вклучува софтверска обработка за да се сливаат информациите од секоја слика и да се креира една крајна слика со подобра квалитет и детали.



Слика 18.

На Слика 18, може да видиме куп од слики со подвижен предмет од левата страна, а на десната како изгледа финалната слика после оваа техника.

Процесот на стакнување на слики обично вклучува следниве чекори:

1. *Собирање на слики:* Првично, се заснимаат повеќе слики од истата сцена со различни параметри како експозиција, фокус. Овие слики треба да бидат фотографирани без движење на камерата или на сцената за најдобри резултати.
2. *Подготовка:* Следува фазата на подготовка, каде што сите слики се внесуваат во специјализиран софтвер за стакнување. Овој софтвер ќе ги процесира и анализира сликите за да ги идентификува и комбинира деталите кои треба да се вклучат во крајната стакната слика.
3. *Алгоритми за стакнување:* Софтверот користи различни алгоритми за стакнување, како на пример алгоритмот за средна вредност (просечно стакнување), алгоритмот за максимална вредност (максимално стакнување), или алгоритми за сливање кои вклучуваат само најостриот дел од секоја слика.
4. *Обработка на шум:* Во некои случаи, софтверот може да вклучи и посебна обработка за намалување на шумот, особено кога стакнувањето вклучува слики заснети при високи ISO.
5. *Крајна слика:* На крајот од процесот, се создава една стакната слика која вклучува најдобрите аспекти од секоја од почетните слики. Оваа крајна слика обично има подобра резолуција, остринa и детали во споредба со било која од поединечните слики.

Имплементација:



```

1 import cv2
2 import numpy as np
3
4 image
5
6 def stack_images(image_paths):
7     images = [cv2.imread(path) for path in image_paths]
8     stacked_image = np.mean(images, axis=0).astype(np.uint8)
9     cv2.imwrite('stacked_image.jpg', stacked_image)
10
11 stack_images(['image.jpg', 'image2.jpg', 'image3.jpg'])

```

Слика 19.

Имплементацијата на *'image stacking'* може да ја видиме на Слика 19. Прво, сликите се читаат од патеките што се предаваат како аргументи на функцијата *stack_images*. Читањето се врши користејќи *list comprehension*, каде што секоја слика се чита со *cv2.imread()* и се зачувува во листата *images*. Потоа, користејќи *np.mean(images, axis=0)*, се пресметува средната вредност од сите слики во листата. Оваа операција резултира со стекната слика која претставува средна вредност на сите соодветни пиксели од сите влезни слики. На крај, финалната слика се зачувува како *'stacked_image.jpg'* користејќи *cv2.imwrite()*. Во овој пример, функцијата *stack_images* се повикува со листа од три слики (*'image.jpg'*, *'image2.jpg'*, *'image3.jpg'*), што резултира во создавање на стекната слика од тие три слики и зачувување на резултатот како *'stacked_image.jpg'*.

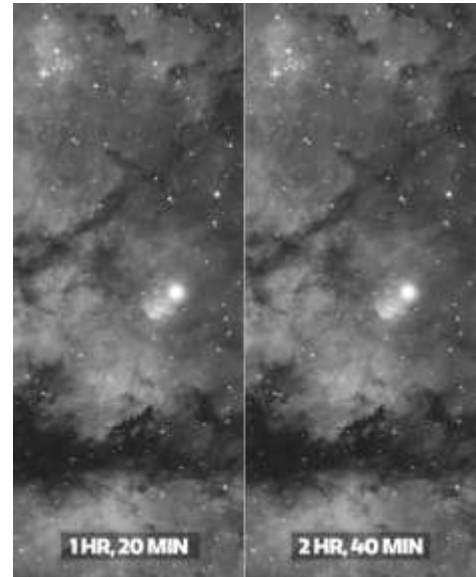
Астрофотографија

Може да се случи сликата што изгледала одлично на екранот на фотоапарат да изгледа сосема поинаку кога ќе ја погледнете одблизу на екранот на компјутер. Постојат докажани начини за намалување на шумот во вашите снимки при слаба осветленост и долга експозиција. Ова се прави преку оваа техника.

Кога ќе се намали количината на шум на сликите, ќе има подобрен сооднос сигнал-шум. Наредената слика ќе има многу „помазна“ позадина и севкупно подобрен квалитет на сликата. Ова е важно при обработување на слики од астрофотографија бидејќи овозможува да се изврши агресивна крива и прилагодување на нивото без да ги уништува или исече податоците.

Речење на поединечни експозиции на слики во вредност од многу часови може да има огромно влијание врз квалитетот на конечната слика. Целокупната интеграција е клучен фактор зад секоја одлична астрофотографија.

На *Слика 20* десно, ја гледаме разликата помеѓу податоците за сликата кога вкупното време на интеграција е двојно зголемено.



Слика 20.



Слика 21.

На *Слика 21* подолу гледаме како повеќекратните експозиции „Средување ѕвезди“ го подобруваат квалитетот на крајната слика.

Овој процес може да биде автоматизиран со софтверски алатки кои олеснуваат обработката и ги доставуваат најдобрите можно квалитет на сликата со помалку човечки напор.

Модели за машинско учење

Алгоритмите за промена на резолуција на слика, како што се супер-резолуција или скалирање на слики, се тесно поврзани со машинското учење, особено со напредните техники како длабокото учење. Еве како некои од машинските модели и техники се применуваат во оваа област:

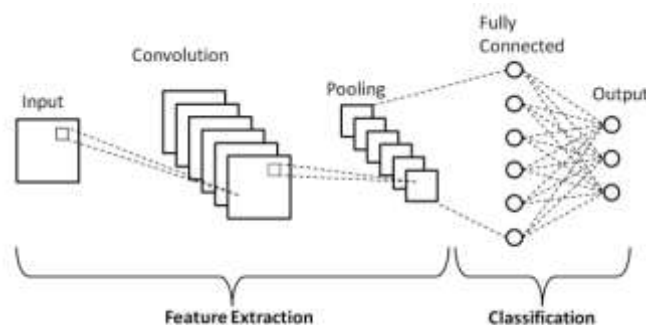
Конволуциони невронски мрежи (CNNs)

Конволуционите невронски мрежи (CNNs) се специјализирани видови на машинско учење кои се често користат за задачи што вклучуваат анализа на слика и препознавање на објекти. Овие мрежи се вдолжни на невронски архитектури кои користат операции на конволуција за обработка на податоците.

Основните карактеристики на CNNs вклучуваат:

1. *Конволуциони слоеви (Convolutional Layers)*: Главниот дел од CNNs се состои од конволуциони слоеви. Во овие слоеви, филтрите се превлекуваат низ влезната слика за да се креираат нови, претпроцесирани излези. Ова овозможува мрежата да научи карактеристики како што се острици, јазли, и текстури.
2. *Пулсни слоеви (Pooling Layers)*: Пулсни слоеви го намалуваат просторниот обем на податоците со земање на максимална или средна вредност во поголеми региони од влезната слика. Ова помага за намалување на бројот на параметри и способува идентификување на карактеристики кои се независни од позиција.
3. *Потполно поврзани слоеви (Fully Connected Layers)*: Потполно поврзаните слоеви на крајот од мрежата ги примаат карактеристиките од конволуционите слоеви и ги користат за класификација или регресија. Овие слоеви се користат за креирање на крајниот излез од мрежата.

Основните предности на CNNs вклучуваат способноста за автоматско изучување на карактеристики од слики, како и способноста за локално и просторно распределување на параметрите. Ова ги прави одлични за задачи како што се класификација на слики, детекција на објекти, сегментација на слики и др. Соодветната конфигурација на овие слоеви зависи од конкретната задача и карактеристиките на податоците.



Слика 22.

На претходната слика, Слика 22, преку цртеж може да видиме како ова функционира.

Да имплементираме основен пример за супер-резолуција користејќи *Convolutional Neural Networks (CNNs)* со *Python* и *TensorFlow/Keras*. Овој пример ќе користи модел наречен *SRCNN (Super-Resolution Convolutional Neural Network)*.

Прво, потребно е да ги инсталираме неопходните библиотеки:

pip install tensorflow numpy matplotlib opencv-python

Потоа, следи имплементацијата:

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input
from tensorflow.keras.models import Model
import numpy as np
import matplotlib.pyplot as plt
import cv2

#usage
def srcnn_model():
    input_layer = Input(shape=(None, None, 1))

    x = Conv2D(64, (9, 9), activation='relu', padding='same')(input_layer)
    x = Conv2D(32, (1, 1), activation='relu', padding='same')(x)
    x = Conv2D(1, (5, 5), activation='linear', padding='same')(x)

    model = Model(input=input_layer, output=x)

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_squared_error'])
    return model

#usage
def preprocess_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image = image.astype(np.float32) / 255.0
    return image
```

Слика 23.

```
#usage
def upscale_image(image, scale):
    height, width = image.shape[:2]
    new_height, new_width = height * scale, width * scale
    return cv2.resize(image, (new_width, new_height), interpolation=cv2.INTER_CUBIC)

#usage
def display_images(lr_image, sr_image):
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Low Resolution")

    plt.imshow(lr_image, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title("Super Resolution")
    plt.imshow(sr_image, cmap='gray')
    plt.show()

if __name__ == "__main__":
    lr_image = preprocess_image('low_res_image.png')
    upscaled_image = upscale_image(lr_image, 4)
    model = srcnn_model()
    sr_image = model.predict(upscaled_image[np.newaxis, :, :, np.newaxis])[0, :, :, 0]
    display_images(upscaled_image, sr_image)
```

Слика 24.

Овој код, прикажан на Слика 23 и Слика 24 е пример за користење на *TensorFlow* и *Keras* за креирање и тренирање модел за супер резолуција на слики, со помош на *SRCNN* (*Super-Resolution Convolutional Neural Network*).

Функцијата *srcnn_model()* креира моделот *SRCNN* со влезен слој (*Input*) за слики со недефинирана големина. Потоа има три конволуциски слоеви (*Conv2D*): првиот со 64 филтри и големина на јадрото (9, 9), вториот со 32 филтри и големина на јадрото (1, 1), третиот со 1 филтер и големина на јадрото (5, 5). Моделот се компајлира со оптимизатор *adam*, функција за губиток *mean_squared_error* и метрика *mean_squared_error*. Потоа функцијата *preprocess_image(image_path)* е за предпроцесирање на сликата вчитана од датотека. Сликата се вчитува во градиентен формат (*grayscale*) и потоа се нормализира на вредности помеѓу 0 и 1. Додека пак функцијата *upscale_image(image, scale)* за зголемување на резолуцијата на сликата со зададен мерител (*scale*). Се користи интерполацијата *cv2.INTER_CUBIC* за подобрување на квалитетот на зголемувањето. *Display_images(lr_image, sr_image)* функцијата за приказ на намалената резолуција и супер резолуција на сликите со помош на *matplotlib*. Во главниот дел на програмата се извршуваат сите функции во секвенца. Сликата со намалена резолуција се вчитува, потоа се зголемува со *upscale_image*, се креира и компајлира моделот со *srcnn_model()* и на крај се прикажуваат оригиналната слика и супер резолуцијата која се предвидува од моделот.

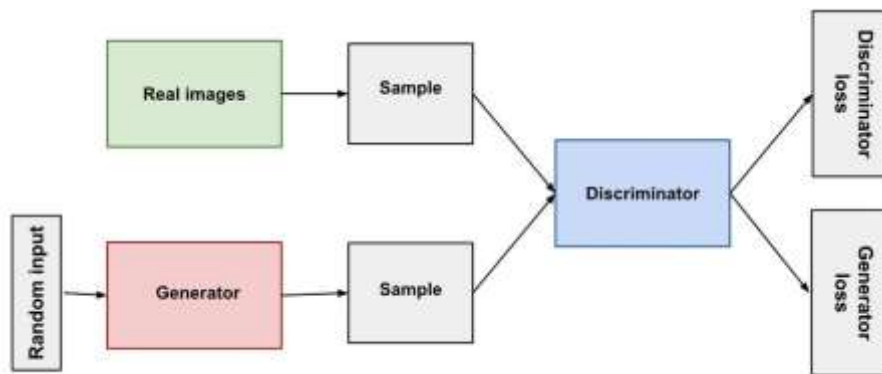
Ова е основниот тек на апликацијата што користи мрежи со длабока конволуција за подобрување на резолуцијата на слика.

Генеративни Адверзаријални Мрежи (GANs)

Генеративни адверзарни мрежи (GANs) се модели во областа на машинското учење кои се користат за генерирање на нови инстанци од податоци, како што се слики, звуци, текст и други видови податоци. Оваа техника на учење е особено популарна поради способноста да создава реалистични излези што се тешки за разлика од реалните податоци.

Принципот на работа на GANs вклучува два главни модела:

1. *Генератор (Generator)*: Ова е модел кој учи да генерира нови примероци од податоците. На пример, ако се работи со слики, генераторот прима случајен шум или почетен влез и го трансформира во слика што треба да изгледа како реални слики.
2. *Дискриминатор (Discriminator)*: Ова е модел кој се учи да разликува помеѓу генерираниот излез од генераторот и реалните примероци од податоците. Во случајот на слики, дискриминаторот прима слики и треба да одлучи дали сликата е реална (од тренирачкото множество) или е генерирана од генераторот.



Слика 25.

На Слика 25, имаме прилика да го видиме процесот на тренирање на GANs, кој се одвива како натпревар помеѓу генераторот и дискриминаторот:

-Фаза на генерација (*Generation Phase*): Генераторот создава нови примероци.

-Фаза на дискриминација (*Discrimination Phase*): Дискриминаторот ги оценува сликите и прави разлика помеѓу генерираните и реалните слики.

Целта е генераторот со текот на тренирање да научи да создава слики или други податоци кои се толку слични на реалните, што дискриминаторот не може да ги разликува.

На сликите кои следуваат - 26, 27 и 28 можеме да го видиме имплементираниот код за GANs. Овој код дефинира Генеративна Адверзаријална Мрежа (GAN) користејќи *TensorFlow* и *Keras* за генерирање на слики. Еве краток преглед на кодот. Ги внесуваме потребните библиотеки вклучувајќи *TensorFlow*, *Keras* за слоеви, *numpy* за работа со низи, *matplotlib* за визуелизација и *OpenCV* за обработка на слики. Потоа се дефинира ***build_generator()*** генератор модел користејќи конволуциски слоеви (*Conv2D*), активација со *leaky ReLU* (*LeakyReLU*) и транспонирана конволуција (*Conv2DTranspose*) за генерирање на слики од случајен шум. Потоа имаме дефиниран ***build_discriminator()*** модел користејќи конволуциски слоеви (*Conv2D*), *leaky ReLU* активација и денс слој (*Dense*) за бинарна класификација на сликите како реални или

генерирани. ***Compile_gan(generator, discriminator)*** го компајлира *GAN* моделот. Прво се компајлира дискриминаторот со бинарна загуба и *Adam* оптимизатор. Потоа, генераторот се комбинира со дискриминаторот (кој е замрзнат) за да се создаде *GAN*. *GAN* се компајлира со бинарна крос-ентропија загуба и *Adam* оптимизатор. ***Train_gan(generator, discriminator, gan, data, epochs, batch_size)*** обучува *GAN*. Се сменува обучувањето на дискриминаторот и генераторот. Се користат реални и лажни ознаки за да се пресмета загубата на дискриминаторот. Генераторот се обучува со помош на *GAN* моделот да генерира слики кои дискриминаторот ги класифицира како реални. Потоа се генерира и зачувува преку ***save_images(generator, epoch, examples, dim, figsize)*** примероци слики генерирани од генераторот за време на обучувањето со помош на *matplotlib*. Главниот блок иницијализира генераторот, дискриминаторот и *GAN* моделите, потоа ги компајлира и започнува обучувањето на *GAN* користејќи даден сет на податоци ('data').

Имплементација:

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, LeakyReLU, Flatten, Dense, Activation, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

input_shape = (64, 64, 3)
input_real = Input(shape=input_shape)
input_fake = Input(shape=input_shape)

def build_generator():
    input_layer = Input(shape=(64, 64, 3))

    x = Conv2D(64, (3, 3), activation='relu')(input_layer)
    x = LeakyReLU(alpha=0.2)(x)

    for _ in range(1):
        x = Conv2D(128, (3, 3), activation='relu')(x)
        x = LeakyReLU(alpha=0.2)(x)

    x = Conv2DTranspose(64, (3, 3), activation='relu')(x)
    x = LeakyReLU(alpha=0.2)(x)
    output_layer = Conv2D(3, (3, 3), padding='same', activation='tanh')(x)

    return Model(input_layer, output_layer)

def build_discriminator():
    input_layer = Input(shape=(64, 64, 3))

    x = Conv2D(64, (3, 3), activation='relu')(input_layer)
    x = LeakyReLU(alpha=0.2)(x)

    for _ in range(1):
        x = Conv2D(128, (3, 3), activation='relu')(x)
        x = LeakyReLU(alpha=0.2)(x)

    x = Flatten()(x)
    output_layer = Dense(1)(x)
    output_layer = Activation('sigmoid')(x)

    return Model(input_layer, output_layer)
```

Слика 26.

```
return Model(input_layer, output_layer)

def compile_gan(generator, discriminator):
    discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0001), metrics=['accuracy'])
    discriminator.trainable = False

    gan_input = Input(shape=(64, 64, 3))
    generated_image = generator(gan_input)
    gan_output = discriminator(generated_image)

    gan = Model(gan_input, gan_output)
    gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0001, 0.5))

    return gan

def train_gan(generator, discriminator, gan, data, epochs=10000, batch_size=64):
    real = np.zeros((batch_size, 3))
    fake = np.zeros((batch_size, 3))

    for epoch in range(epochs):
        idx = np.random.randint(0, data.shape[0], batch_size)
        imgs = data[idx]

        gen_imgs = generator.predict(imgs)

        d_loss_real = discriminator.train_on_batch(real, real)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        g_loss = gan.train_on_batch(imgs, real)

        if epoch % 100 == 0:
            print(f'Epoch: {epoch} | D loss: {d_loss[0]}, gen. loss: {g_loss} | D loss: {d_loss[1]} | G loss: {g_loss}')
            save_images(generator, epoch)
```

Слика 27.

```
def save_images(generator, epoch, examples=5, dim=(1, 1), figsize=(10, 10)):
    noise = np.random.normal(0, 1, (examples, 100))
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axo = plt.subplots(dim[0], dim[1], figsize=figsize)
    cnt = 0
    for i in range(dim[0]):
        for j in range(dim[1]):
            axo[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
            axo[i, j].axis('off')
            cnt += 1
    plt.show()

if __name__ == '__main__':
    generator = build_generator()
    discriminator = build_discriminator()
    gan = compile_gan(generator, discriminator)
```

Слика 28.

Главните предности на *GANs* вклучуваат способноста за генерирање на високо квалитетни и реалистични излези, но и предизвиците вклучуваат стабилност во тренирањето, контрола на конвергенцијата и потребата за напредни техники за стабилизација како што се поддршката на непрекинато.

Проучување на алгоритми за зголемување на резолуцијата

Претходно обучените модели за супер-резолуција, како што се *ESRGAN (Enhanced Super-Resolution GAN)* и *EDSR (Enhanced Deep Residual Networks for Single Image Super-Resolution)*, се користат за постигнување високи перформанси во промена на резолуцијата на слики. Овие модели користат техники на усвоено учење за да се обучат на големи податочни сетови и потоа да се применат на нови слики.

EDSR (Enhanced Deep Super Resolution)

Enhanced Deep Super-Resolution (EDSR) е техника за зголемување на резолуцијата на слики, базирана на длабоки невронски мрежи. *EDSR* е предложена од *Lim et al.* во 2017 година. Оваа метода користи подобрена верзија на *ResNet (Residual Network)* архитектурата, отстранувајќи непотребни операции како *batch normalization*, што ја подобрува перформансата и квалитетот на зголемените слики. *EDSR* е особено ефикасен за задачи каде што е потребна висока резолуција и квалитет на сликите. Овој модел успева да постигне извонредни резултати со минимални загуби на квалитет, благодарение на неговата способност да учи комплексни мапирања од ниска до висока резолуција.

Овој код претставен на *Слика 29* користи *EDSR (Enhanced Deep Super-Resolution)* модел за зголемување на резолуцијата на слика со помош на *TensorFlow*. Прво, се увезуваат потребните библиотеки: *'tensorflow'* за машинско учење и длабоко учење, *'numpy'* за работа со нумерички податоци и матрици, и *'PIL' (Python Imaging Library)* за работа со слики.

Функцијата *'load_image'* вчитува слика од дадена патека користејќи *'Image.open(image_path)'* и ја конвертира во *RGB* формат со *'image.convert('RGB')'*. Потоа, сликата се претвора во *numpy array* со *'np.array(image)'* и се нормализира вредноста на пикселите во опсегот од 0 до 1 со *'image / 255.0'*. Вчитаната и нормализирана слика се враќа како резултат.

```

1 import tensorflow as tf
2 from tensorflow.keras.models import load_model
3 import numpy as np
4 from PIL import Image
5
6 image
7
8 def load_image(image_path):
9     image = Image.open(image_path)
10    image = image.convert('RGB')
11    image = np.array(image)
12    image = image / 255.0
13    return image
14
15 image
16
17 def save_image(image, save_path):
18     image = (image * 255).astype(np.uint8)
19     image = image.fromarray(image)
20     image.save(save_path)
21
22 image
23
24 def edsr_super_resolution(image_path, save_path):
25     model = load_model('path_to_edsr_model.h5')
26     image = load_image(image_path)
27     image = np.expand_dims(image, axis=0)
28     upscaled_image = model.predict(image)
29     save_image(upscaled_image[0], save_path)
30
31 edsr_super_resolution(image_path, 'input_image.png', save_path, 'edsr_upscaled.png')

```

Слика 29.

Функцијата *'save_image'* ја зачувува сликата на дадена патека. Прво, пикселите на сликата се деинормализираат и се конвертираат во целобројни вредности со *'image * 255'*

и `astype(np.uint8)`. Потоа, *numpy array*-ом се претвора во *PIL* слика со `Image.fromarray(image)` и се зачувува на дадената патека со `image.save(save_path)`.

Функцијата `edsr_super_resolution` ја извршува целата постапка за зголемување на резолуцијата. Прво, *EDSR* моделот се вчитува со `load_model('path_to_edsr_model.h5')`. Потоа, влезната слика се вчитува со функцијата `load_image(image_path)`, а за потребите на моделот, на сликата и се додава нова димензија за батч процесирање со `np.expand_dims(image, axis=0)`. Моделот ја зголемува резолуцијата на сликата со `model.predict(image)`, а зголемената слика се зачувува на дадената патека со `save_image(upscaled_image[0], save_path)`.

На крај, функцијата `edsr_super_resolution` се повикува со аргументите `'input_image.png'` за влезна слика и `'edsr_upscaled.png'` за излезна зголемена слика.



Слика 30.

Сликата 30 прикажува три различни верзии на една иста слика на крило од пеперутка, кои демонстрираат различни техники за зголемување на резолуцијата. Првата слика е ниско-квалитетна верзија на сликата со мала резолуција. Веднаш може да се забележи дека пикселите се многу видливи и деталите на крилото на пеперутката се губат. Ова е оригиналната слика со мала резолуција пред зголемување. Сликата во средина е зголемена верзија на сликата со користење на традиционална интерполација (на пример, *bilinear* интерполација). Сликата е подобрена, но деталите не се јасни и краевите на крилото се замаглени. Додека пак, на сликата која се наоѓа десно е користен *EDSR* (*Enhanced Deep Super-Resolution*) модел. Деталите на крилото на пеперутката се многу појасни и краевите се остри. Ова покажува како *EDSR* моделот може значително да го подобри квалитетот на зголемената слика во споредба со традиционалните методи.

Сликата добро го илустрира напредокот во технологијата за зголемување на резолуцијата и колку може да се подобри квалитетот на сликите со користење на длабоки невронски мрежи како што е *EDSR*.

ESRGAN (Enhanced Super-Resolution Generative Adversarial Network)

ESRGAN (*Enhanced Super-Resolution Generative Adversarial Network*) е значаен метод во областа на супер-резолуција на слики. Дизајниран е да ги зголемува сликите со ниска резолуција до поголема резолуција, зголемувајќи ги деталите и произведувајќи визуелно апелативни резултати.

- *Цел*: Подобрување на квалитетот на сликите со супер-резолуција со зголемување на текстурните детали и одржување на веродостојноста.

- *Основен модел*: Се базира на рамнотежниот генеративно-аналитички модел *SRGAN*, но воведува неколку подобрувања.

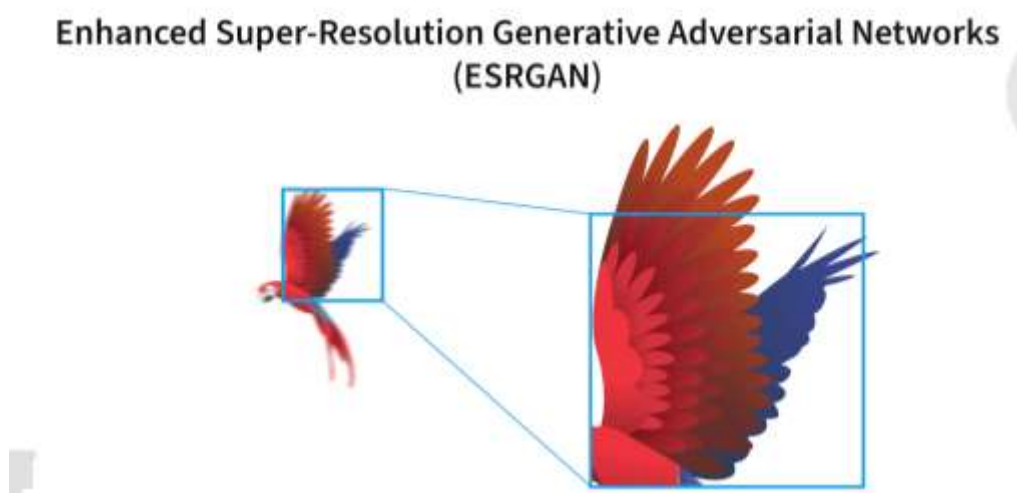
Клучни Подобрувања во споредба со *SRGAN*:

1. Резидуален во Резидуален Денс Блок (*RRDB*):

- Го подобрува капацитетот на мрежата да научи попрецизни детали со употреба на резидуални денс блокови без слоеви за нормализација на батерија. Ова овозможува подобро задржување и подобрување на карактеристиките на сликата.

2. Релативистички Среден Дискриминатор:

- Вместо традиционалниот дискриминатор на *GAN*, *ESRGAN*



Слика 31.

На Слика 31 се прикажани две слики од папагал, каде што втората е зголемена и прикажува многу повеќе детали, што илустрира подобрената резолуција која *ESRGAN* ја обезбедува.

Зголемување на сликата со вештачка интелигенција

Традиционалните методи за зголемување на резолуцијата, како што се најблискиот сосед (*nearest-neighbor*), билинеарната и бикубната интерполација, имаат свои ограничувања и често резултираат со губење на детали и појава на артефакти. За да се надминат овие предизвици, вештачката интелигенција (*AI*) и длабокото учење (*deep learning*) нудат напредни решенија кои можат значително да го подобрат квалитетот на зголемените слики.

AI Image Upscaling е процес кој користи напредни алгоритми базирани на машинско учење и длабоко учење за да ја зголеми резолуцијата на сликите додека ги зачувува и подобрува деталите. Овие алгоритми се обучуваат на големи бази на податоци со парови на слики со ниска и висока резолуција, што им овозможува да учат како да генерираат

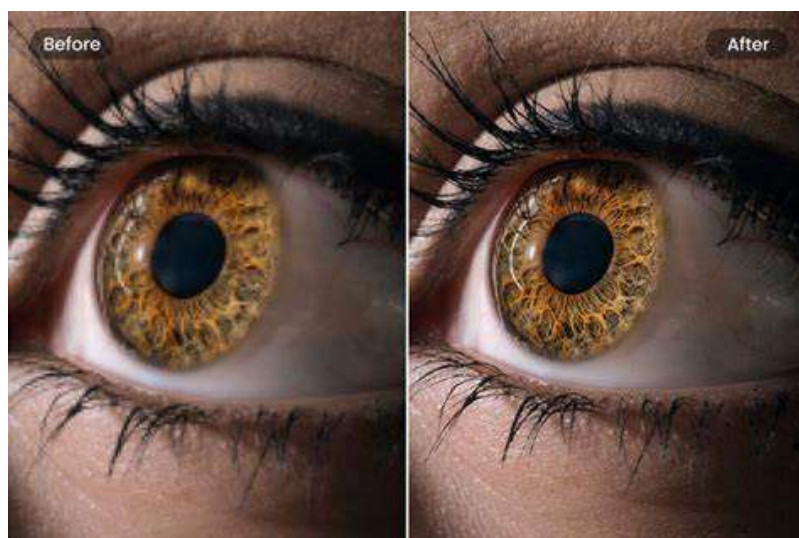
нови детали кои не постојат во оригиналните слики. Со примена на овие алгоритми, можеме да добиеме значително подобрување на визуелниот квалитет на зголемените слики.



Слика 32.



Слика 33.



Слика 34.

Во денешницата веќе постојат готови софтверски апликации со вештачка интелигенција кои го имаа зголемувањето на резолуција за цел. На Слика 32, Слика 33, како и Слика 34 може да видиме како *AI image upscaling* ги користи тие готови софтверски апликации кои се достапни за корисниците. Некои од најпопуларните апликации и алатки вклучуваат: *Topaz Gigapixel AI*, *Adobe Photoshop (Super Resolution)*, *Let's Enhance*, *Waifu2x*, *Deep Image*, *AI Image Enlarger*.

Овие апликации користат различни алгоритми и техники за да го зголемат квалитетот на сликите, а некои од нив нудат бесплатни верзии или пробни периоди за да може да се тестира нивната функционалност.

Заклучок

Во овој документ се истражуваат и споредуваат различни алгоритми за промена на резолуција на слики, вклучувајќи традиционални методи и напредни техники базирани на вештачка интелигенција (AI). Традиционалните методи, како што се билеарната и бикубичната интерполација, се добро воспоставени и често користени поради нивната едноставност и брзина. Сепак, овие техники имаат ограничувања во однос на зачувување на детали и квалитет при значително зголемување на резолуцијата.

Наспроти тоа, алгоритмите базирани на вештачка интелигенција, како што се *Super-Resolution Convolutional Neural Networks (SRCNN)* и *Enhanced Super-Resolution Generative Adversarial Networks (ESRGAN)*, покажуваат значителни подобрувања во квалитетот на сликите. Овие алгоритми користат длабоко учење за да разберат и реконструираат фини детали во сликите, што резултира со појасни и реалистични резултати.

AI-базираните методи се поефикасни за апликации каде што е критично зачувувањето на детали и квалитет. Со оглед на брзиот напредок во областа на машинското учење и вештачката интелигенција, се очекува овие техники да продолжат да се подобруваат и да најдат поширока примена во различни индустрии, вклучувајќи фотографија, видеографија, медицински слики и безбедност.

На крајот, изборот на соодветен алгоритам зависи од специфичните потреби на апликацијата, ресурсите за обработка и посакуваниот квалитет на резултатот. Традиционалните методи можат да бидат соодветни за брзи и едноставни задачи, додека AI-базираните методи се препорачуваат за сложени апликации каде што е потребен висок степен на деталност и прецизност.

Референци

- [1] [Deep Learning for Image Super-Resolution \[incl. Architectures\] \(v7labs.com\)](#)
- [2] <https://astrobackyard.com/tutorials/stack-exposures/>
- [3] <https://sumittagadiya.medium.com/image-super-resolution-using-edsr-and-wdsr-f4de0b00e039>
- [4] <https://pyimagesearch.com/2022/06/13/enhanced-super-resolution-generative-adversarial-networks-esrgan/>
- [5] [\[GIS\] Lanczos resampling useful for in a spatial context – Math Solves Everything \(imathworks.com\)](#)
- [6] Dong, C., Loy, C. C., He, K., & Tang, X. (2016). "Image super-resolution using deep convolutional networks."
- [7] Ledig, C., Theis, L., Huszár, F., Caballero, J., Aitken, A. P., Tejani, A., ... & Shi, W. (2017) "Photo-realistic single image super-resolution using a generative adversarial network."
- [8] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., ... & Tang, X. (2018) "ESRGAN: Enhanced super-resolution generative adversarial networks."
- [9] Yang, W., Wang, X., Shi, Y., Huang, T., & Ding, X. (2019). "Deep learning for single image super-resolution: A brief review."
- [10] Timofte, R., Gu, S., Wu, J., Van Gool, L., & Zhang, R. (2016). "NTIRE 2017 challenge on single image super-resolution: Methods and results."