

Софийски университет “Св. Климент Охридски”

Факултет по математика и информатика

Проект

по

“Размити множества и приложения”

на тема

“Размито запълване на тъмни
пиксели в зашумено изображение”

Изготвен от:

Георги Пачов, ф.н.3MI3400212,
Изкуствен Интелект, I курс

10.02.2023г.

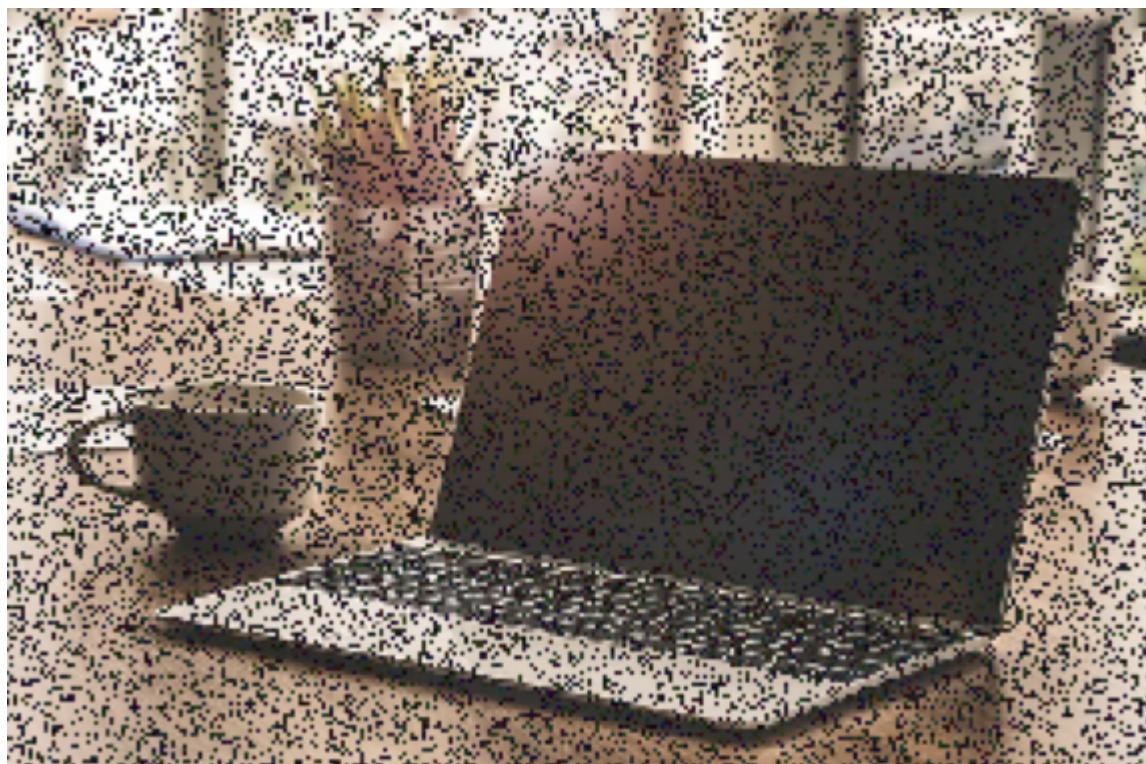
1. Задача - запълване на тъмни пиксели в зашумено изображение

В практиката често се случва да боравим с зашумени изображения. Съществуват различни типове шум - гаусов, импулсен, периодичен, шум, породен от грешки в квантуването и т.н. Най-често разглежданите шумове при изображения се проявяват при последователност от аналогово и дигитално предаване.

Един по-рядко разглеждан шум в изображенията е от тип черни пиксели. Той се получава при по-редки и по-специфични обстоятелства - например при realtime distributed rendering системи, където 60 пъти в секундата трябва да се визуализира изображение. Заради разпределения характер на изчислителната система, различни части от изображението се разпределят на различни изчислителни машини. Ако обаче се получи грешка, забавяне или друг проблем в изчислителната инфраструктура, част от изображението би била готова за визуализация, докато друга би била недоизчислена.

Пример за такъв тип система е distributed raytracing. В някои реализации цветът за даден пиксел или група от пиксели се смята от отделни задача. В други различни задачи се смятат от компоненти от изчислението на финалния цвят (например ефект на отражението на огледалото), които трябва да бъдат "смесени" накрая. В такъв случай при липса на налични резултати, свързани с разпространение на светлината, липсата им води до тъмни, но не изцяло черни пиксели.

Има различни способи за разпределяне на изчислението - по цвят, по зона, по трасиран обект и т.н. Има и различни начини за маркиране и справяне с проблема с несметнатите навреме части от визуализацията. В конкретния случай ще приемем, че шумът е равномерно разпределен по изображението, а несметнатните пиксели ще са със стойност от 0 до 63 (включително), равномерно разпределена (тъмни, но не задължително черни).



(примерно изображение - равномерно разпределен шум, визуализиран чрез тъмни пиксели)

2. Съществуващи начини за справяне с липсващи пиксели

Съществуват немалък брой класически (crisp) алгоритми за справяне с шумове в изображение. Те имат своите силни и слаби страни и успешното им приложение зависи от типа на шума, количеството на зашуменост, количество светлина в снимката и т.н.

Например едни от най-разпространените са Gaussian Blur и Fast Non-Local Means Denoising. Първият работи много добре при остри изображения с добра резолюция, а вторият се справя отлично при наличие на големи “зони” от изображението със сходни цветове.



(горе в ляво - входно изображение

горе в дясно - зашумено изображение

долу в ляво - gaussian blur denoising

долу в дясно - fast non local means denoising)

3. Цел на проекта

Целта на проекта е да предложи размит алгоритъм за запълване на тъмни пиксели, който дава визуално и естетически по-добър резултат от гореспоменатите crisp начини за справяне с проблема. Като допълнителна цел е да се демонстрират лостове за контрол на резултата посредством подходящи избори на представяне, операции, t-норми/t-конорми и т.н.

4. Размит алгоритъм за запълване на тъмни пиксели (и неговия crisp еквивалент)

В предложения алгоритъм всеки пиксел в изображението се обхожда (еднократно) и интензитетите по 3-те му компонента (RGB) се комбинират по размит начин с интензитетите на 4-те му съседни* пиксела и съответните им компоненти. Чрез избиране на подходящо представяне, конорми за смесване, реализация на averaging

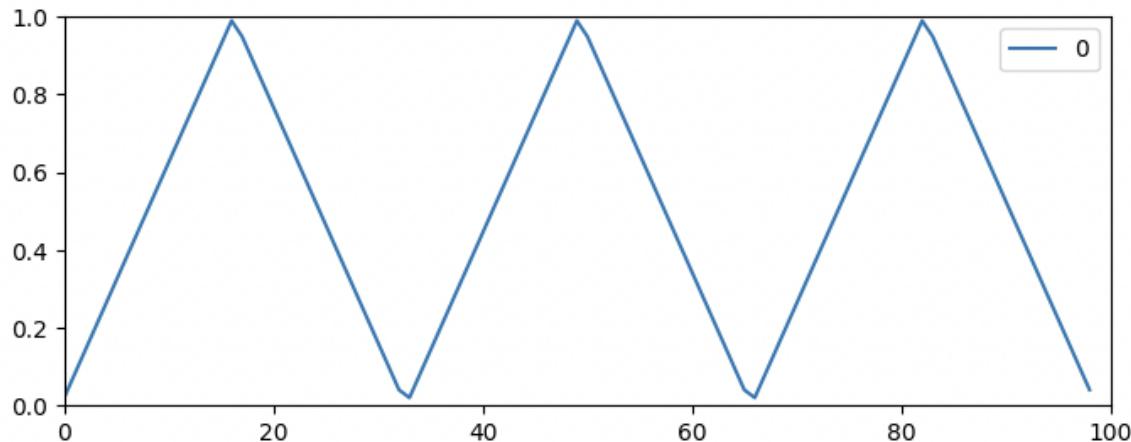
функция и процес на фъзификация/дефъзификация, имаме възможност да влияем на алгоритъма и на крайния резултат.

(Поради дискретното представяне и изискванията за добро изчислително време, алгоритъмът използва 4-те съседа. При бъдещо минаване в аналитично представяне, може да се ползват 8-те съседа.)

5. Представяния - размит пиксел, размито изображение, операции

Съществуват различни видове представяния на концепцията за размит пиксел. В конкретния случай сме избрали да работим в цветовото пространство RGB, без да допускаме покомпонентно смесване.

Всеки пиксел е представен като едно-единствено размито множество, с универсум от 0 до 99. Червената зона е от 0 до 32 (включително), зелената от 33 до 65 (включително), синята от 66 до края.



(Пример 1: размит пиксел - цвят бяло (255, 255, 255), с избрано триъгълно представяне)

При това представяне не е допуснато застъпване между компонентите.

Една от причините е за това е намаляването на контраста на изображение при допускане на застъпване. Например ако червената компонента е с много по-голяма стойност от останалите две (при цвят (255, 0, 0) и има застъпване, то при min норма, червеният компонент би се намалил, защото останалите са 0. При избор на max норма, останалите компоненти биха се увеличили заради пресичане с червената компонента. И в двата случая компонентите биха се “приближили” едни към други, ако допуснем пресичане и смесване. Това би довело до намаляне на контраста и

замъгляване на изображението и недобри резултати на алгоритъма. Съществуват представления, с които може да се допусне пресичане, след което да се прилага contrast intensification операция, но това би повлияло процеса на дефъзификация и би било по-подходящо при използване на LAB цветово пространство (където "нюансът" е само един компонент и може да се представи като размито множество с дефъзификация тип "център на тежестта")

Апаратът на размитите множества се прилага при "смесването" със съседните пиксели.

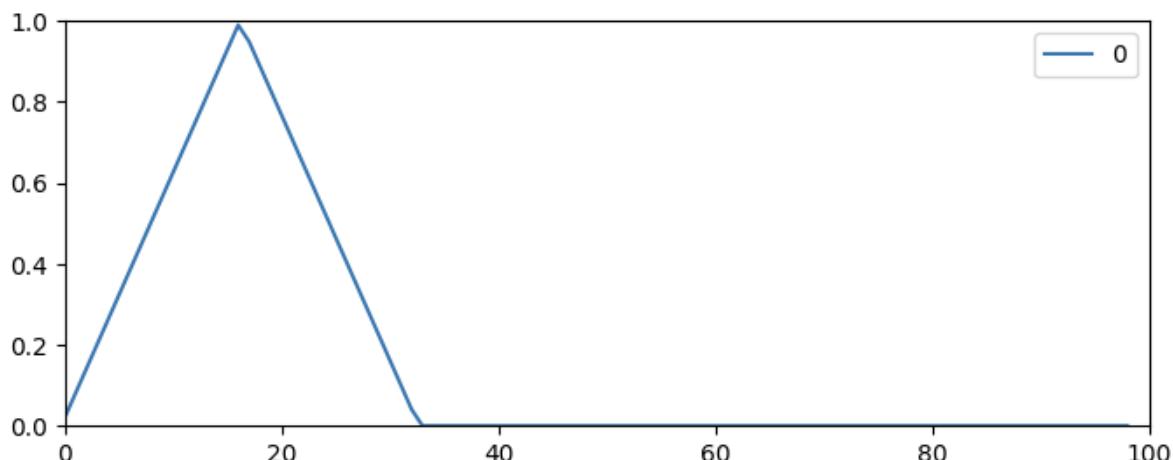
За "събиране" при размитите пиксели се прилага определена функция - t-норма (например - prod) или t-конорма (например einstein sum).

Вместо "деление", се използвана операцията power (степенуване).

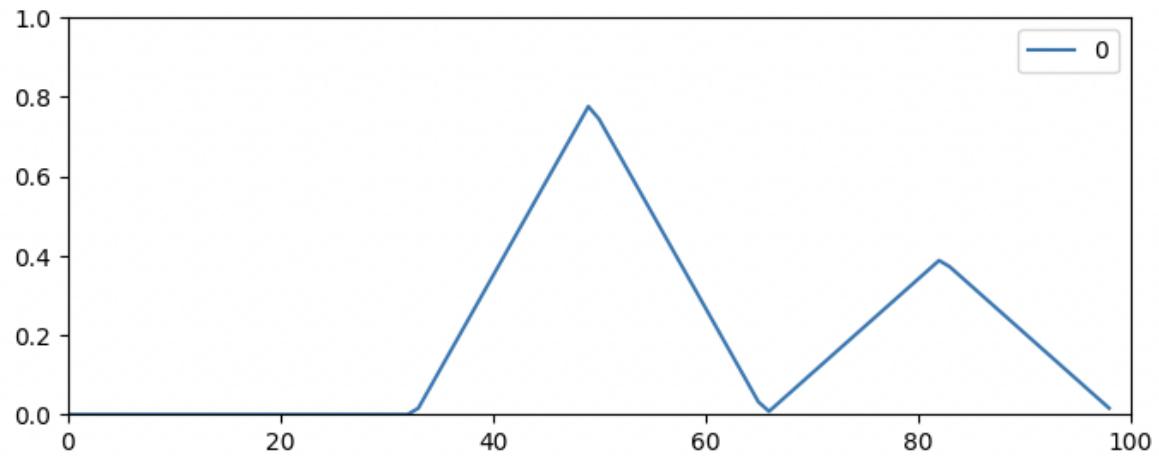
Примери

```
C1 = FuzzyColor((255, 0, 0)) # red  
C2 = FuzzyColor((0, 200, 100)) # green and blue  
C3 = C1 + C2
```

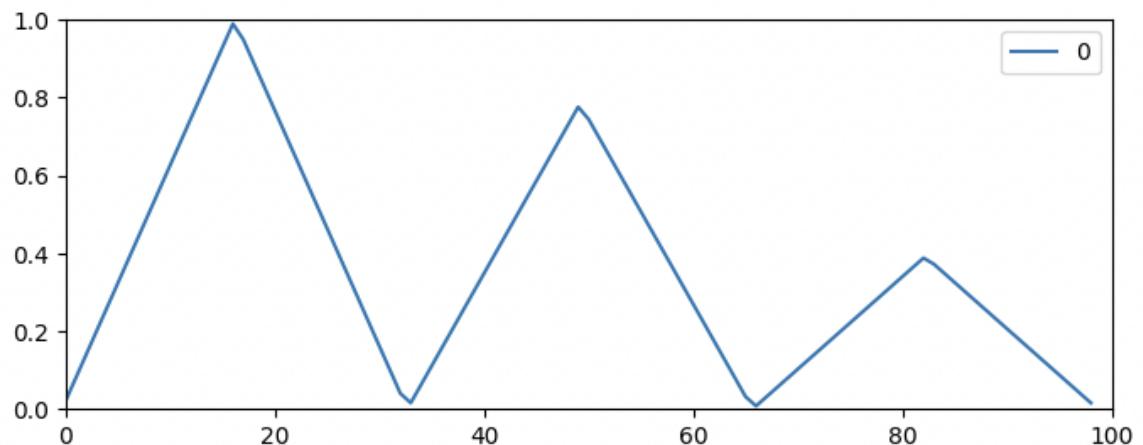
C1:



C2:

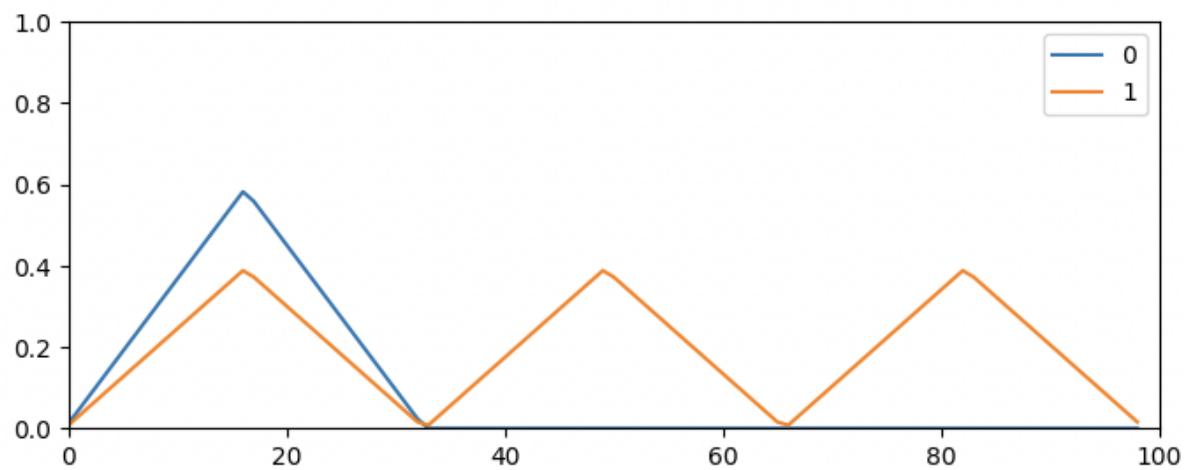


C3:

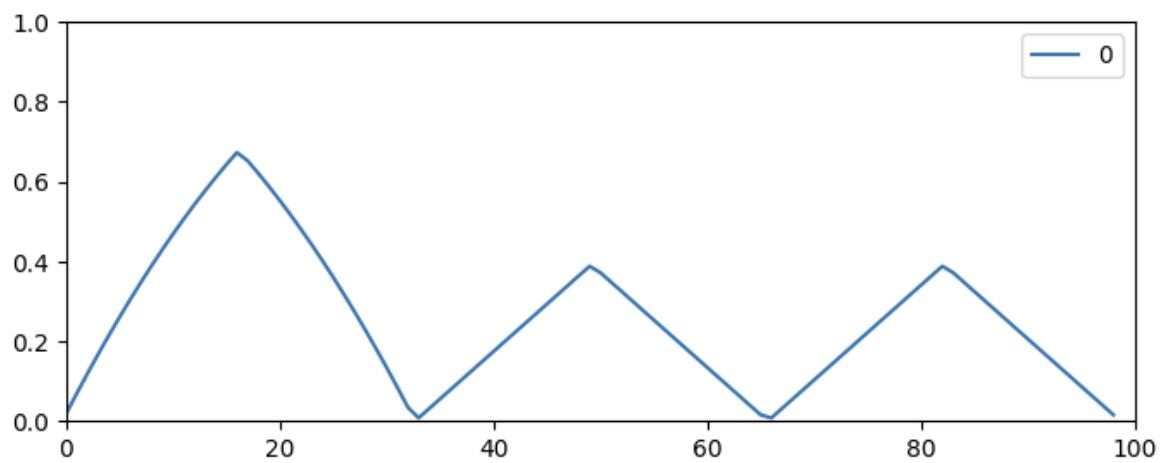


```
C4 = FuzzyColor((128, 0, 0)) # red  
C5 = FuzzyColor((100, 100, 100)) # gray  
C4 + C5
```

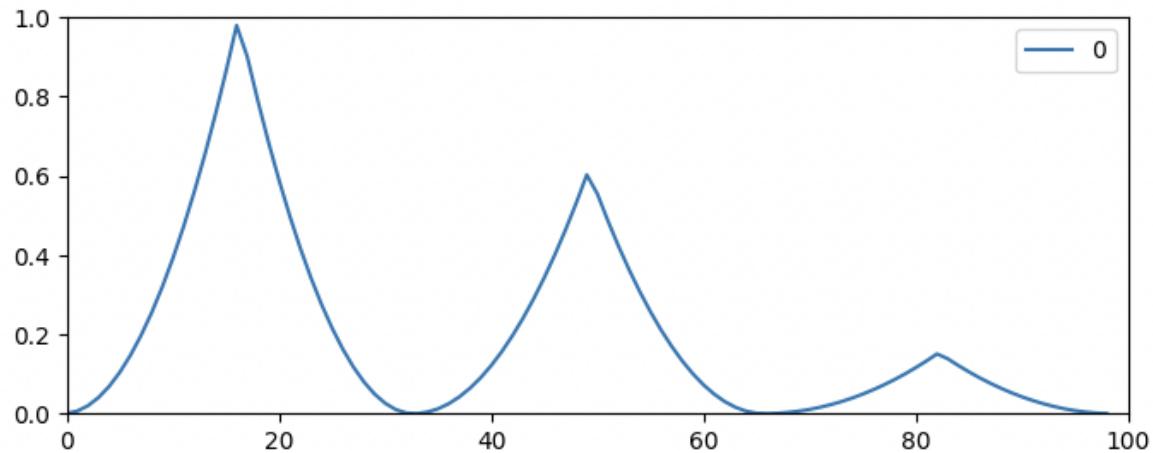
C4 (in blue), C5 (in orange):



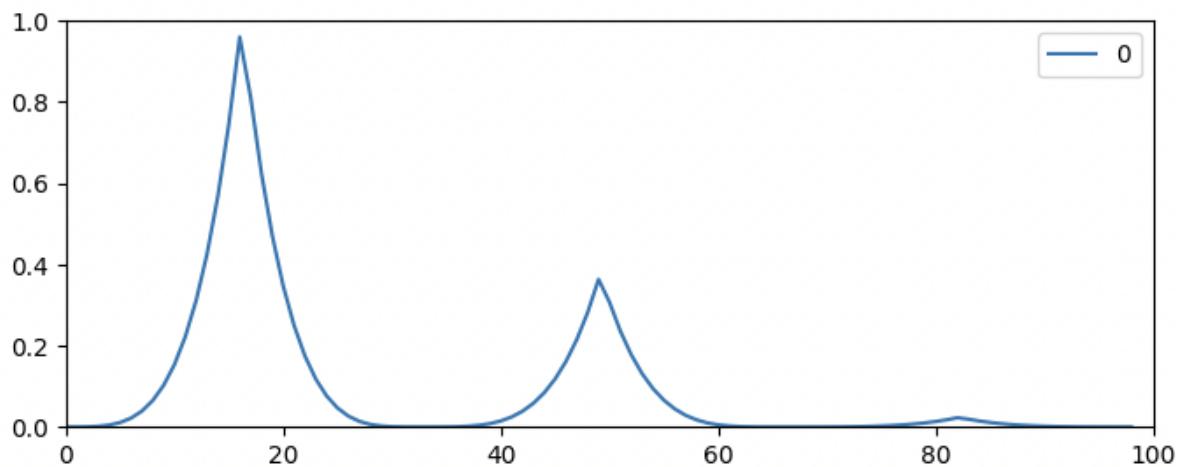
C4+C5 (using prod t-norm):



C3.power(2)



C3.power(4)



Фъзификация и дефъзификация

При фъзификация на даден пиксел се взимат трите интензитета - red, green, blue и се създава триъгълно множество в съответния range на универсума:

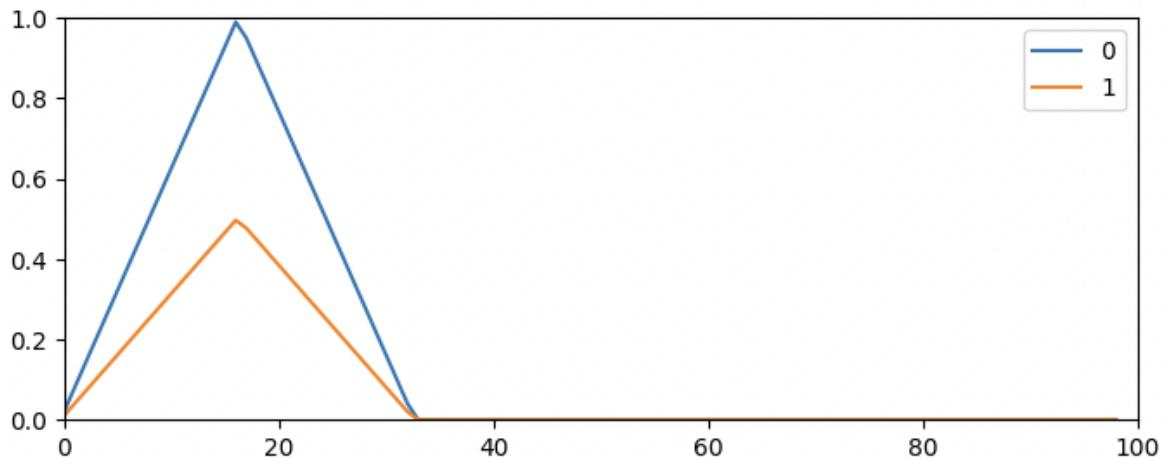
- 0 до 32 - червено
- 33 до 65 - зелено
- 66 до 98 - синьо

В зависимост от интензитета на компонента се определя **височината** на триъгълника.

Например:

r0 = FuzzyColor((255, 0, 0))

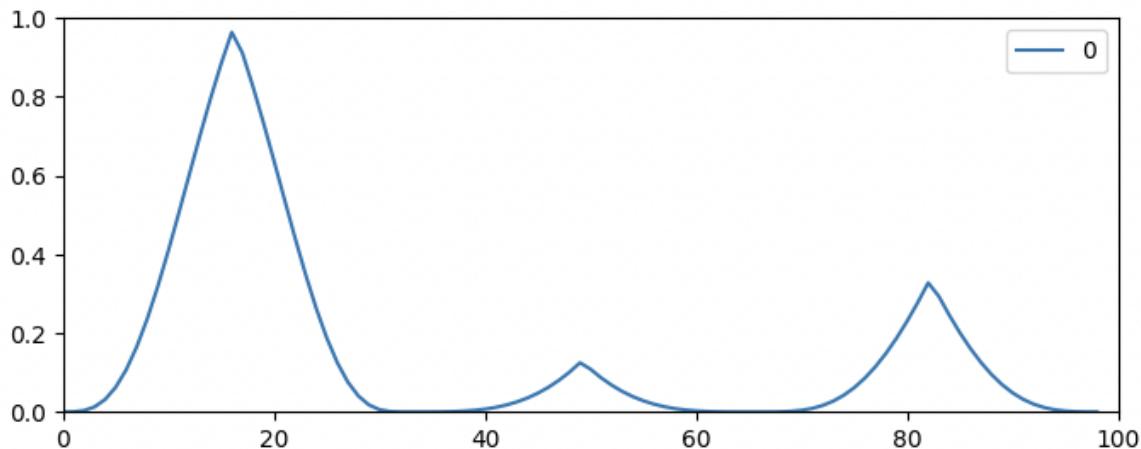
r1 = FuzzyColor((128, 0, 0))



За дефъзификация се използва съотношението на площта на размитото множество към максималната възможна такава.

Например нека за даден размит пиксел сме приложили алгоритъма и сме извършили следните операции чрез съседите му: $(C1+C2+C3+C4).power(4)$

Нека резултатът е например:



При избраното триъгълно представяне на компонентите максималната възможна площ за компонент е $(33 * 1)/2 = 15.5$.

За всеки цветови компонент се взема площта на съответния триъгълник в размития пиксел и се разделя на 15.5

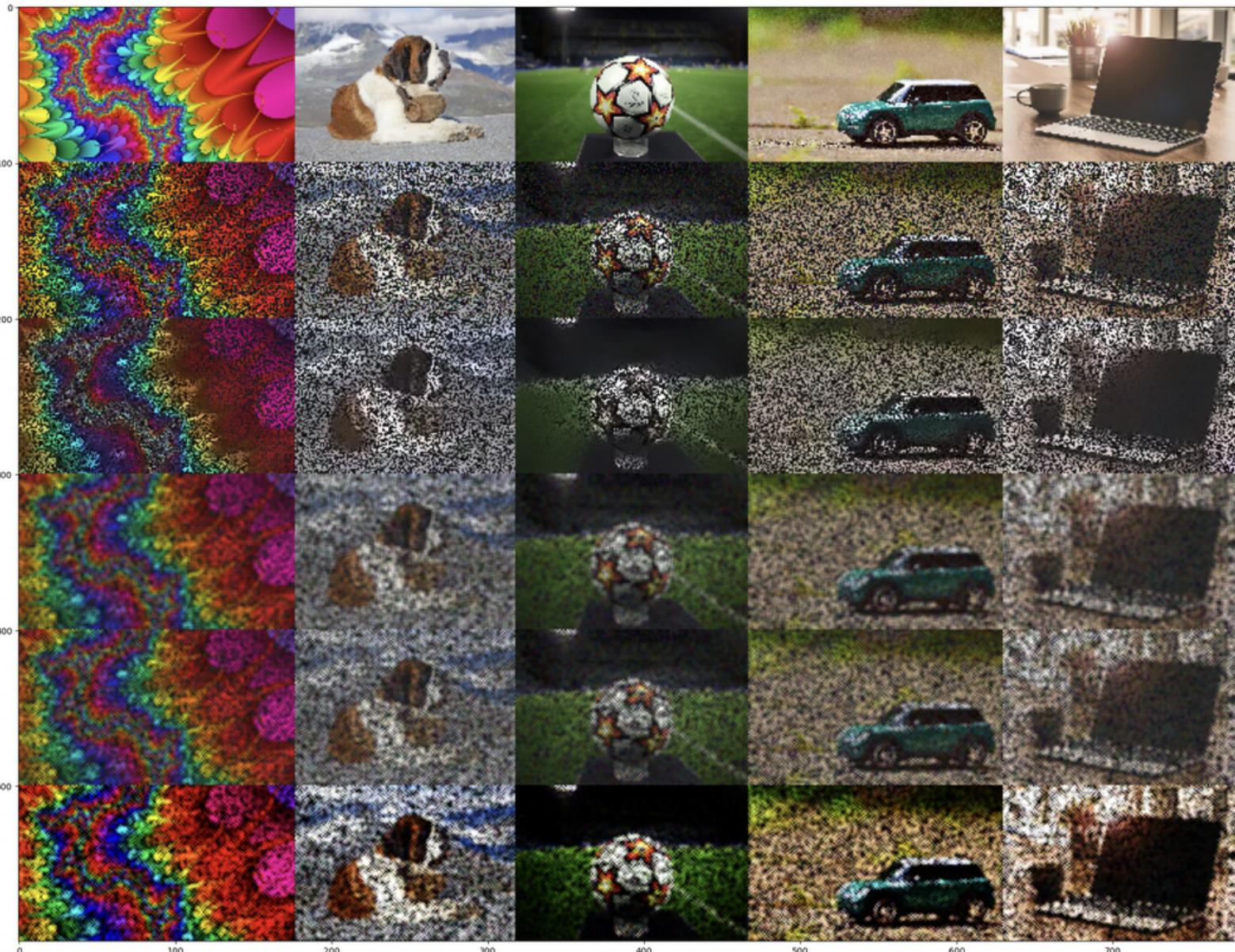
За примерния пиксел получаваме:

Red = 170, Green = 15, Blue = 44

Представянето е реализирано дискретно, с ограничена точност.

Операциите смесване и повдигане на степен на размит пиксел са реализирани чрез Python.

6. Резултати и сравнение



(първи ред - оригинални изображения)

(втори ред - зашумени изображения)

(трети ред - *fast non-local means* със стандартни параметри)

(четвърти ред - *gaussian blur*)

(пети ред - алгоритъм за запълване на тъмни пиксели в *crisp* вариант)

(шести ред - алгоритъм за запълване на тъмни пиксели в размит *вариант*)

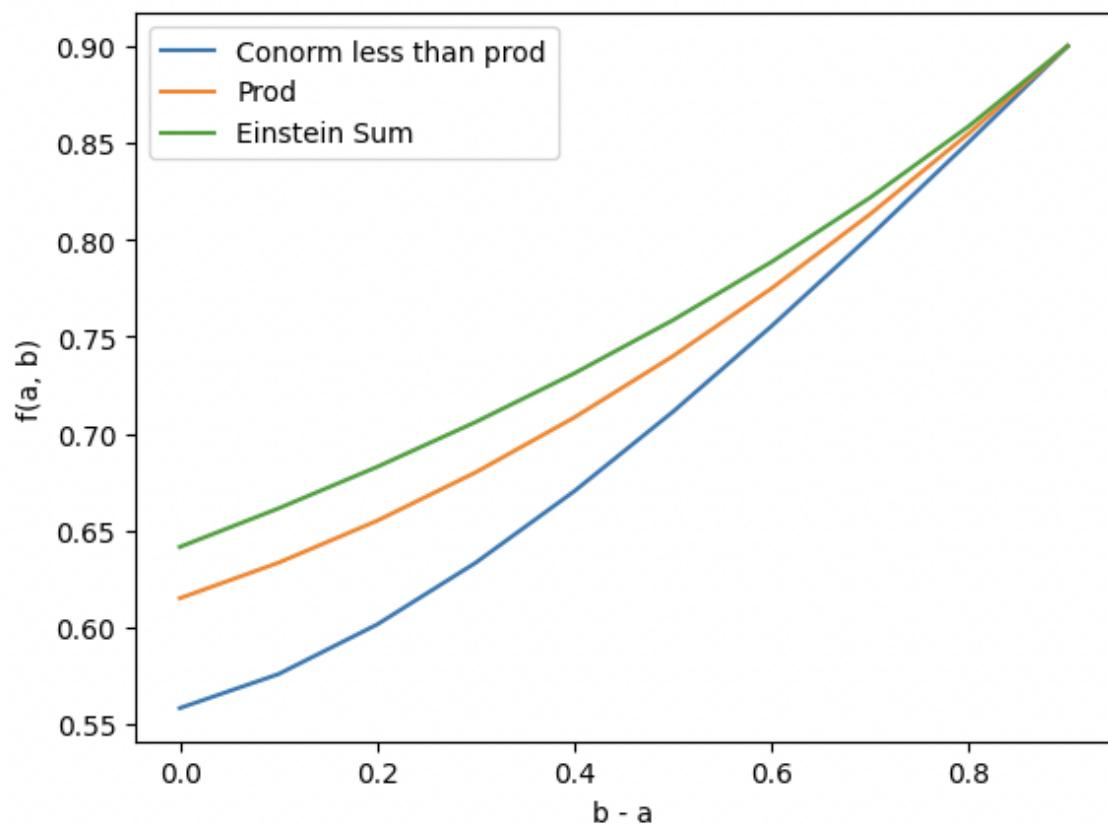
Резултатите са получени чрез следните параметри:

- Триъгълно представяне на цветовете
- Product t-норма за комбиниране на размити пиксели
- Зашумяване на пиксели - равномерно разпределено между 0 и 64
- Ниво на зашуменост - 50% от пикселите

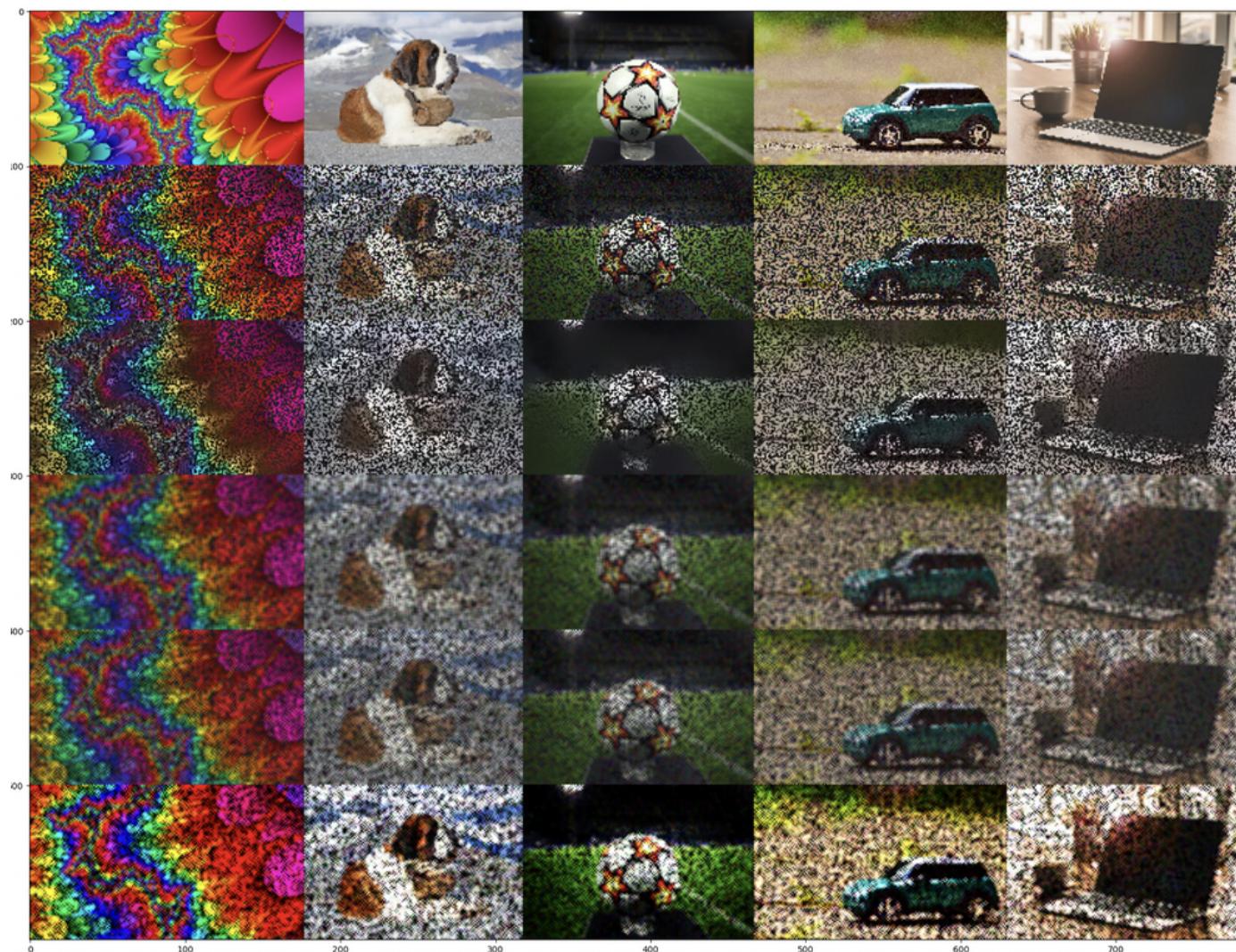
Впечатление правят запазването на контраста и “цветността” на изображенията, получени от размитият алгоритъм спрямо останалите.

Интересен резултат представлява възможността за “контрол” на осветеността на изображенията чрез смяна на функцията за комбиниране на размити пиксели.

- Ако например искаме изображението да е възможно по-светло (например, при много тежко зашумени изображения), можем да използваме “по-агресивна” функция за комбиниране - т-конормата einstein sum: $\frac{(a+b)}{(1+ab)}$ (в зелено на графиката)
- Обратно, ако искаме да “затъмним” изображението, можем да използваме функция с по-плавно поведение - например $\sqrt{a^2 + b^2 - a^2b^2}$ (в синьо на графиката)



При използване на Einstein sum:



(първи ред - оригинални изображения)

(втори ред - зашумени изображения)

(трети ред - fast non-local means със стандартни параметри)

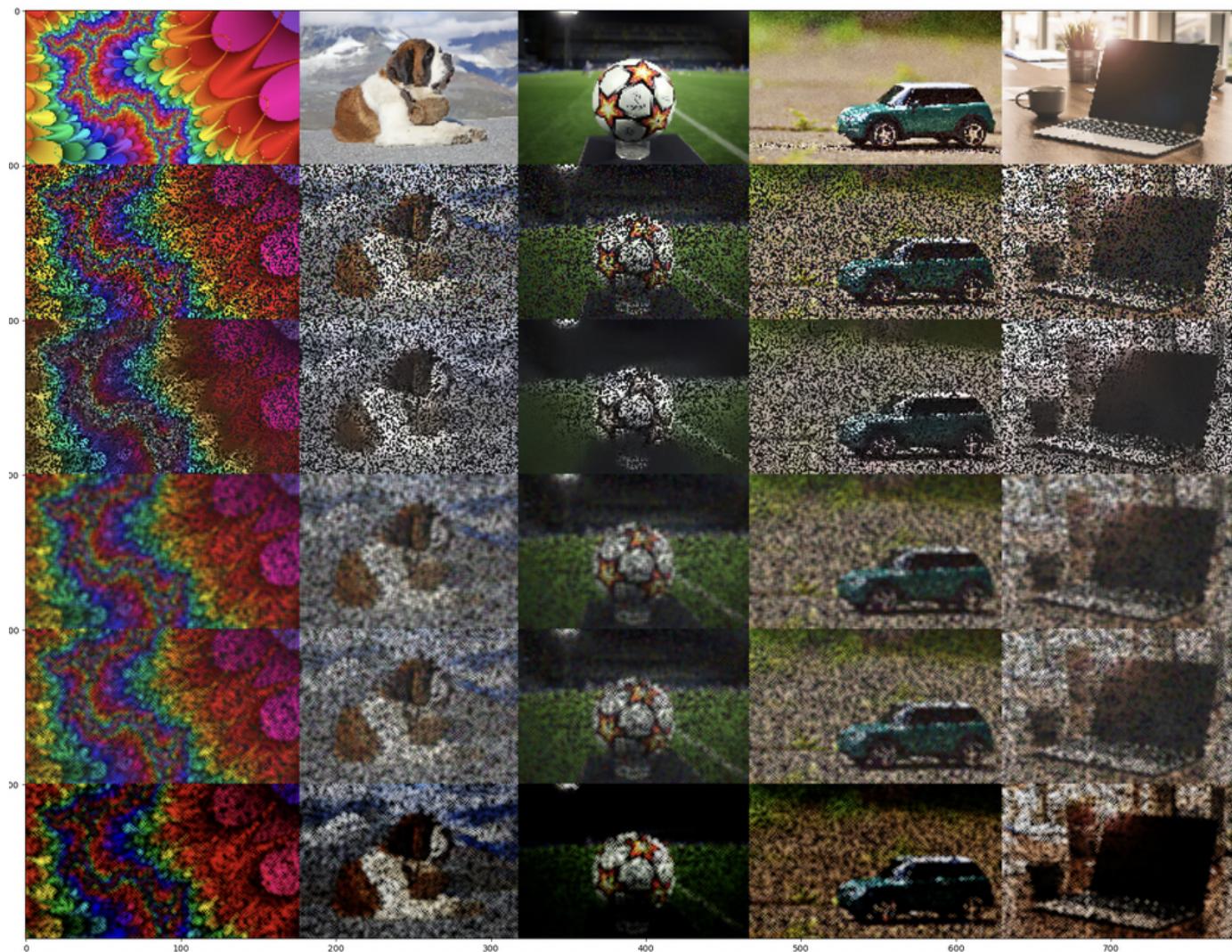
(четвърти ред - gaussian blur)

(пети ред - алгоритъм за запълване на тъмни пиксели в crisp вариант)

(шести ред - алгоритъм за запълване на тъмни пиксели в размит variant)

Прави впечатление значителната яркост и запазване на цветовете, засилено спрямо предходните резултати.

Обратното, ако решим да ги намалим, може да ползваме по-”слаба” функция за смесване на размити пиксели:



(първи ред - оригинални изображения)

(втори ред - зашумени изображения)

(трети ред - *fast non-local means* със стандартни параметри)

(четвърти ред - *gaussian blur*)

(пети ред - алгоритъм за запълване на тъмни пиксели в *crisp* вариант)

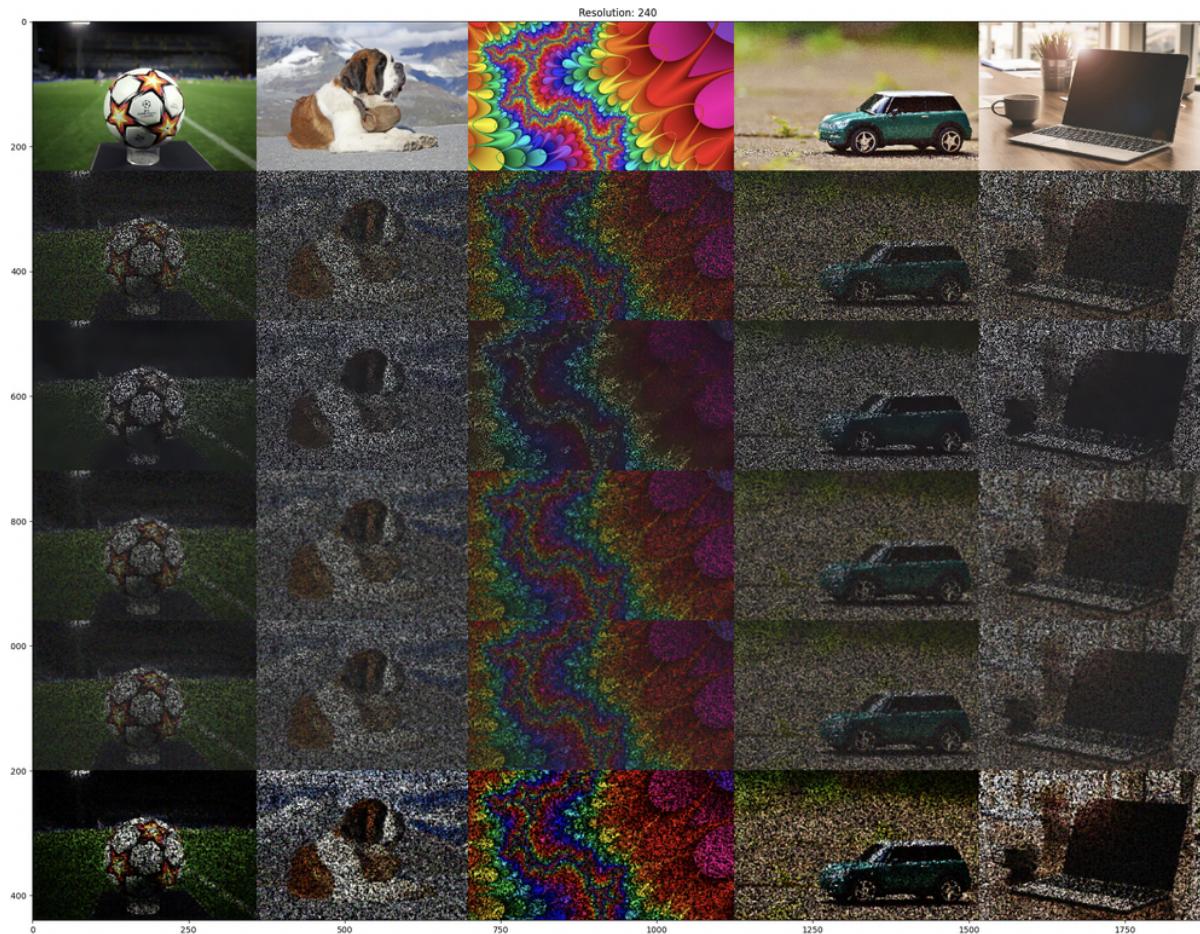
(шести ред - алгоритъм за запълване на тъмни пиксели в *размит* вариант)

Резултатът много прилича визуално на останалите методи, но е значително по-тъмен на места (около топката, около кучето и т.н.). Разбира се възможен е избор на weighted average между тези функции, за още по-фин контрол на осветеността.

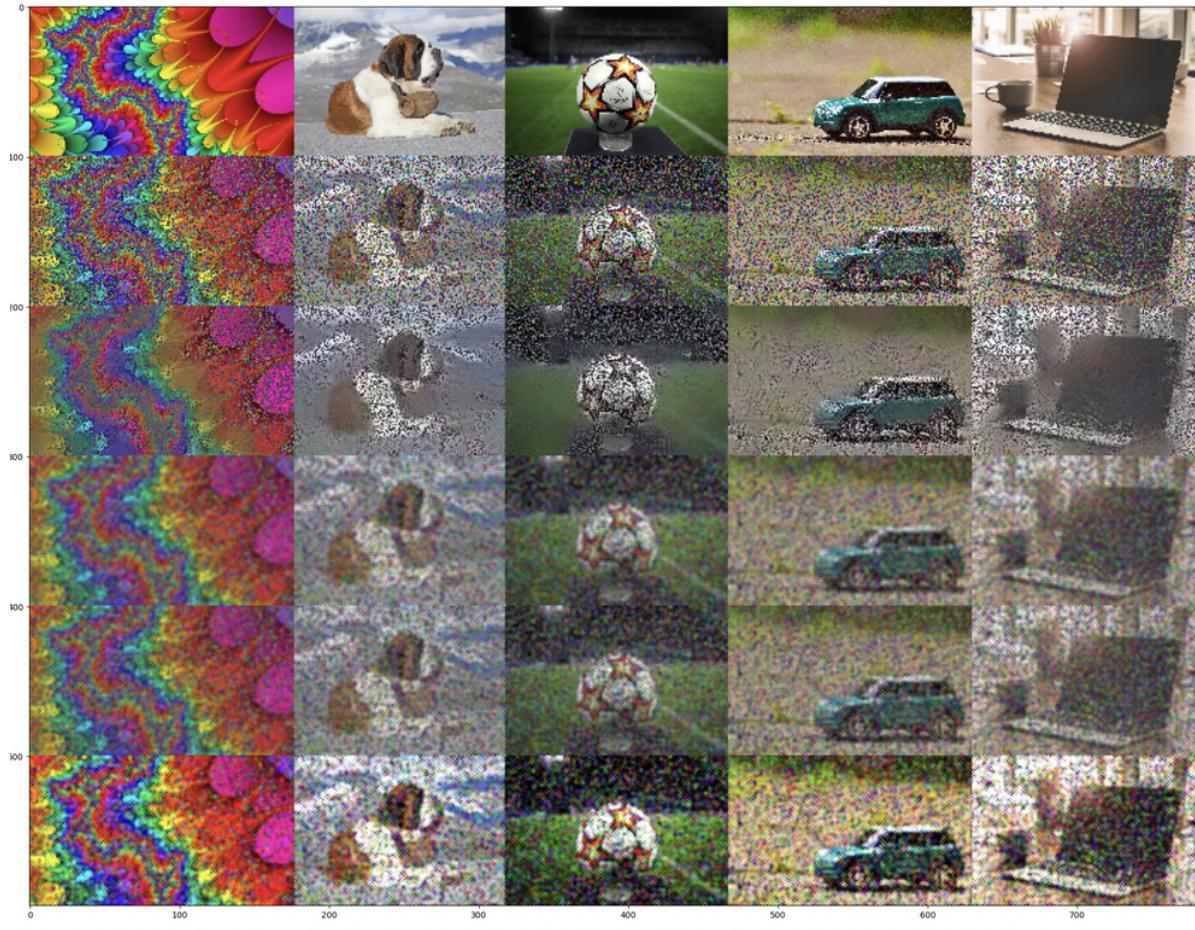
При предефиниране на задачата като запълване на “светли” точки, е възможно да се използват и други функции за смесване, които да предотвратяват “преосветяване” на пикселите и така да се справят по-добре от други алгоритми.

7. Приложение и бъдещи насоки

При много зашумени изображения (над 75%), размитият алгоритъм успява да запази повече от контраста и яркостта и спомага по-лесно да се предаде основата на информацията от изображението:



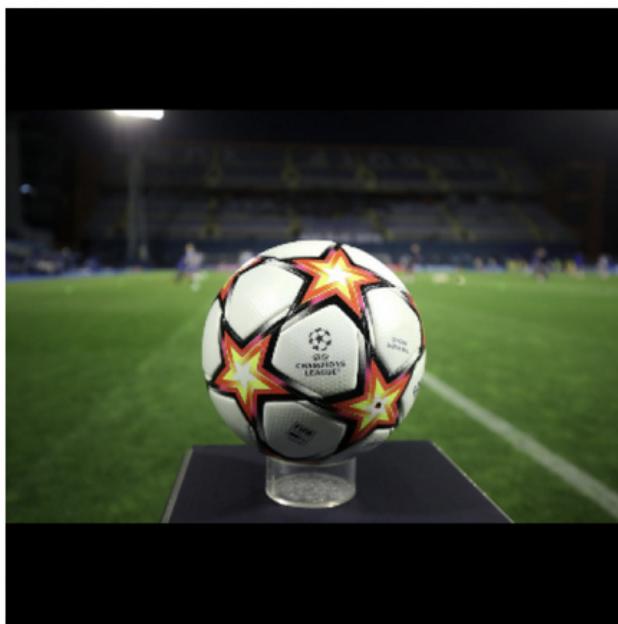
При псевдо-произволен шум - случаини равномерно разпределени стойности по всяка цветова компонента, размитият алгоритъм продължава да продуцира визуално приятни резултати:



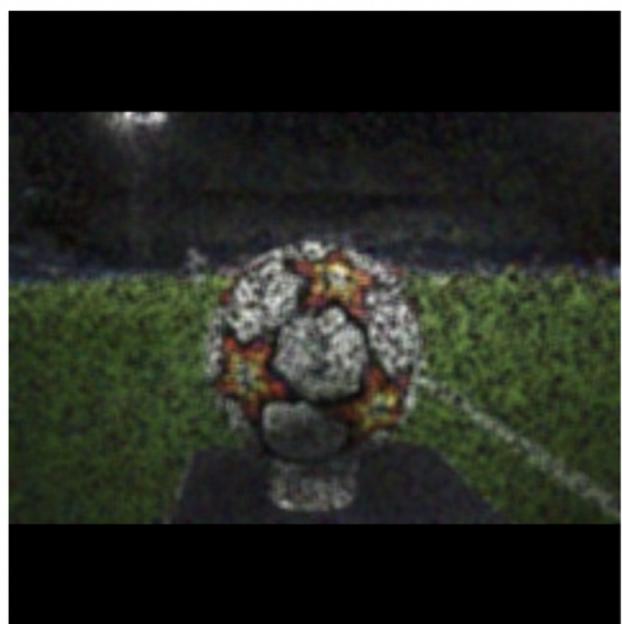
Интерес представлява хипотезата, че резултатите са визуално **“по-различими”** от останалите алгоритми. Въпреки че това е субективно, можем да направим едно интересно измерване - чрез използване на претрениран deep learning модел за визуална класификация на обекти. Конкретно ще използваме EfficientNetV2:

Резултати:

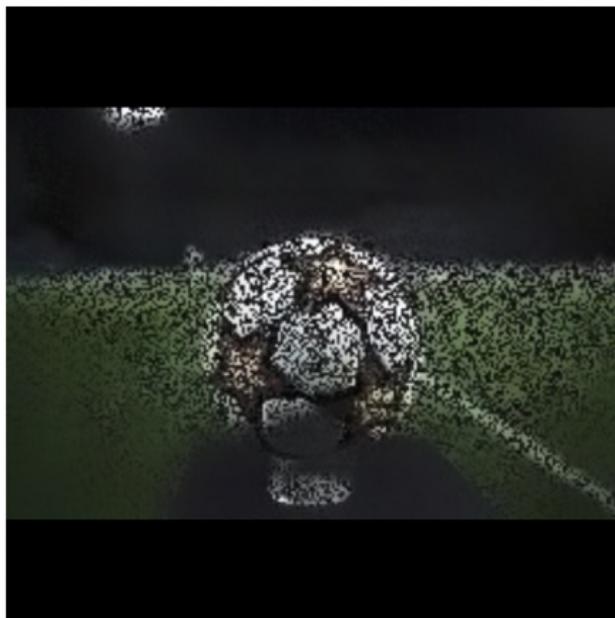
- (1) 806 - soccer ball: 0.9109391570091248
- (2) 769 - rugby ball: 0.003952098544687033
- (3) 891 - volleyball: 0.000889505899976939
- (4) 523 - croquet ball: 0.0008791134459897876
- (5) 430 - baseball: 0.0008567430777475238



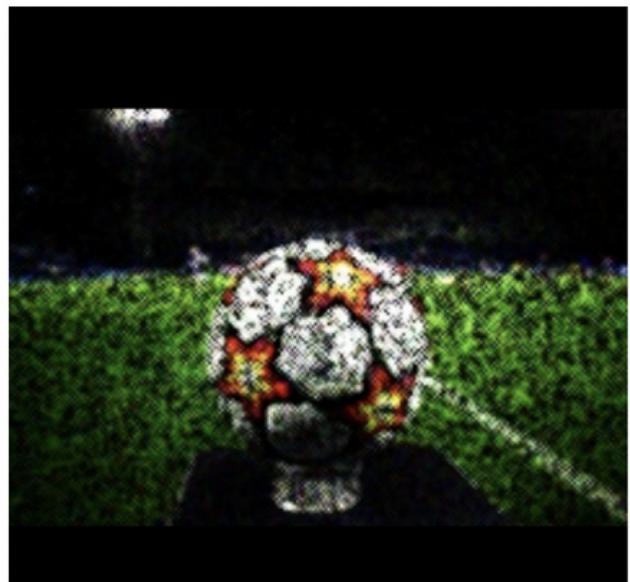
- (1) 621 - laptop: 0.19696304202079773
- (2) 894 - wallet: 0.14638620615005493
- (3) 749 - purse: 0.05648249015212059
- (4) 470 - cauldron: 0.03761843591928482
- (5) 440 - bearskin: 0.02718299813568592



- (1) 563 - fountain: 0.08877062052488327
- (2) 713 - Petri dish: 0.0692436695098877
- (3) 972 - bubble: 0.06847785413265228
- (4) 308 - weevil: 0.059118419885635376
- (5) 311 - ant: 0.025719303637742996



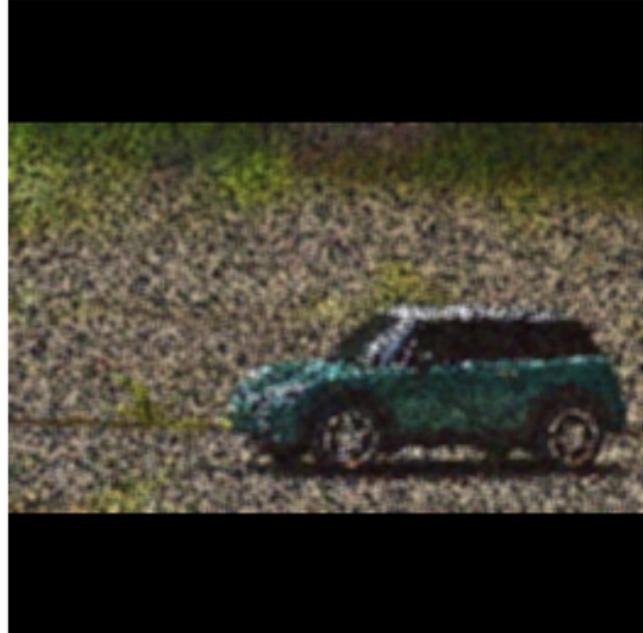
- (1) 540 - doormat: 0.30086854100227356
- (2) 647 - maze: 0.10257743299007416
- (3) 575 - golf ball: 0.07204648852348328
- (4) 585 - hair slide: 0.03225892409682274
- (5) 465 - buckle: 0.03172626718878746



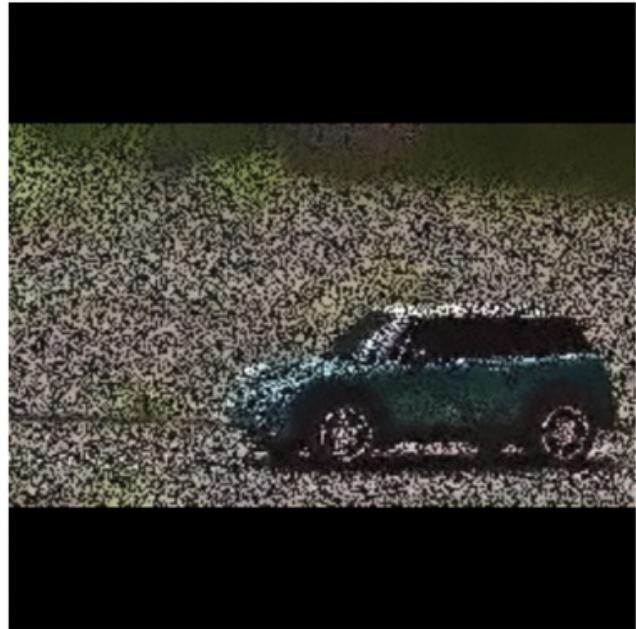
- (1) 610 – jeep: 0.6237126588821411
- (2) 437 – beach wagon: 0.060914523899555206
- (3) 480 – car wheel: 0.037095218896865845
- (4) 752 – racer: 0.026301728561520576
- (5) 818 – sports car: 0.022340716794133186



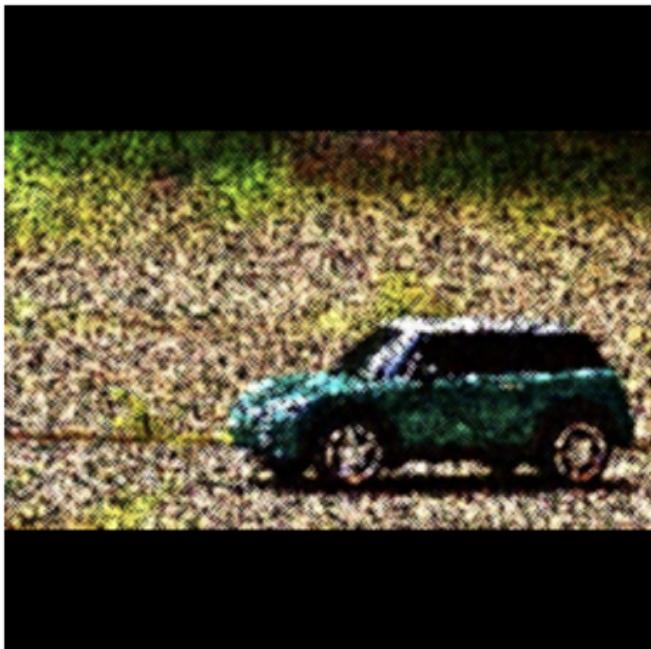
- (1) 657 – minivan: 0.18990851938724518
- (2) 437 – beach wagon: 0.15248188376426697
- (3) 610 – jeep: 0.061852000653743744
- (4) 718 – pickup: 0.05094611644744873
- (5) 576 – golfcart: 0.04506242275238037



- (1) 409 – amphibian: 0.20339855551719666
- (2) 657 – minivan: 0.1339776515966934
- (3) 437 – beach wagon: 0.08746148645877838
- (4) 718 – pickup: 0.051399700343608856
- (5) 628 – limousine: 0.04135199636220932



- (1) 437 – beach wagon: 0.5049242973327637
- (2) 657 – minivan: 0.15857744216918945
- (3) 610 – jeep: 0.07645699381828308
- (4) 628 – limousine: 0.024513404816389084
- (5) 718 – pickup: 0.02273288555443287



Вижда се, че някои от картинките са разпознати като категория по-близка до оригиналните такива. В останалите случаи не бе наблюдавана значителна разлика.

Бъдещи идеи за доразвитие биха включвали:

- Използване на аналитично представяне. Дискретното представяне е ограничено, натрупват се quantization errors и е бавно за изчисление. При увеличаване на точността се мултилицират изискванията за памет и процесорно време.
- Експериментиране с по-сложни представления на цветовете - например използване на гаусиани вместо триъгълни функции.
- Използване на 8-съседство вместо 4-съседство - поради бавното изпълнение на алгоритъма, 4 съседството е предпочитано като връщащо (визуално) достатъчни добри резултати и значително по-бързо за изпълнение. При наличие на аналитично представяне, спокойно би могло да се използва и 8-съседство.

8. Използвана литература

- Gaussian filter for denoising
<https://fiveko.com/gaussian-blur-filter/>
- 4 and 8-neighborhoods of pixels
<https://gigl.scs.carleton.ca/node/519>
- Fast non-local means for noise reduction
<https://arxiv.org/pdf/1407.2343.pdf>
https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html
- Triangular Norms and Conorms -
http://www.scholarpedia.org/article/Triangular_norms_and_conorms
- T-Conorms
<https://en.wiktionary.org/wiki/t-conorm>
- EfficientNet V2
<https://arxiv.org/abs/2104.00298>
https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_s/classification/2

9. Използван софтуер

- Python
- Numpy
- Matplotlib
- Jupyter notebook
- Open CV
- Tensorflow