Prelucrarea și Analiza Imaginilor Color – Proiect

Filtrarea zgomotului mixt din imaginile color prin algoritmul "Fuzzy Peer Group"

Introducere

De cele mai multe ori imaginile digitale sunt afectate simultan atât de zgomot impulsiv, cât și de zgomot gaussian. Filtrele simple de mediere statistică sau cele gaussiene reduc zgomotul, însă au neajusul de a estompa detalile sau contururile din imagini. O soluție ce apare în acest caz este de a detecta contururi sau detalii (prin metode de analiză statistică locală) și a le estompa mai puțin (sau chiar deloc) față de restul imaginii. Însă devine problematic în cazul zgomotului mixt, când pixelii afectați de zgomot impulsiv pot fi identificați ca detalii și ocoliți de filtrare. O reducere a zgomotului impulsiv se poate realiza prin eliminarea valorilor extreme (*outliers*) dintr-o fereastră de filtrare, impulsul fiind identificat ca o deviație extremă în cadrul setului de pixeli [1].

O metodă ce înglobează toate aceste soluții este reprezentată de algoritmul de filtrarea *Fuzzy Peer Group*, ce presupune ca pentru fiecare pixel să se găsească în vecinătatea sa (în cadrul ferestrei de filtrare) un grup de pixeli vecini similari, numit grup de similaritate (*peer group*), iar filtrarea celor două tipuri de zgomot să se facă în concordanță cu acesta. Se va expune o scurtă descriere a algoritmului *Fuzzy Peer Group*.

Descrierea algoritmului de filtrare

Pentru fiecare pixel din imaginea afectată de zgomot se aplică pașii următori pe o fereastră de dimensiune **n** x **n**. Pentru fiecare pixel din fereastra respectivă:

- Se calculează o funcție de similaritate **ρ** (*fuzzy similarity function*) care ia valori în intervalul [0, 1] și care denotă gradul de similaritate dintre pixelul curent și pixelul central din fereastră.

$$\rho(Fi, Fj) = e^{-\frac{\|Fi-Fj\|}{F\sigma}} \quad (1)$$

Unde: - Fi și Fj sunt pixelii între care se calculează similaritatea

- ‖·‖ reprezintă norma euclidiană (distanța euclidiană în spațiul RGB), cu cât distanța între cei doi pixeli este mai mică (culoarea apropiată) cu atât cei doi pixeli sunt mai similari

- Fσ este un parametru (o constantă mai mare ca 0)

- Se ordonează pixelii din fereastră descrescător în raport cu valorile funcțiilor de similaritate (valoarea maximă de 1 va corespunde chiar pixelului central care este evident cel mai similar cu el însuși).
- Se consideră o funcție **C** ce arată gradul de apartenență al unui pixel din fereastră la grupul de similaritate (*peer group*). Reprezentată chiar prin valorile funcției de similaritate ordonate, așadar **C** este o funcție descrescătoare.
- Se consideră o funcție **A** ce arată gradul de „asimilare de similaritate" până la un anumit pixel. Se determină prin însumarea valorilor funcției de similaritate până la valoarea corespunzătoare unui anumit pixel curent.

$$\mathrm{A(F(i))} = \sum_{k=0}^{k=i} \rho(F0, F(k)) \quad (2)$$

Unde: F(i) este pixelul de pe poziția *i* după ordonare și F0 este pixelul central

  - Nu se va folosi direct funcția **A**, ci o funcție **L** (funcție ce depinde de valorile lui **A**) pentru a putea cuantiza certitudinea frazei "**A**(**F**(i)) este o mulțime numeroasă" sau "gradul de similaritate acumulat este suficient".

$$L\big(F(i)\big) = -\frac{1}{(n^2-1)^2}\big(A\big(F(i)\big) - 1\big)\big(A\big(F(i)\big) - 2n^2 + 1\big) \quad (3)$$

- Pe baza funcțiilor introduse se estimează numărul optim de pixeli din grupul de similaritate (m_{optim})
  - Pentru fiecare **m** în {1, 2, …, n^2-1} se calculează ceritudinea **C_FR1**(m) (*Fuzzy Rule 1*)

$$C\_FR1(m) = C(F(m)) \cdot L(F(m)) \quad (4)$$

    - Valoarea lui **m** optimă va fi valoarea pentru care certitudinea C_FR1 este maximă.
  - Se calculează certitudinea **C_FR2**(m) (*Fuzzy Rule 2*) pentru pixelul central

$$C\_FR2(F0) = C(F(\mathrm{m}_{optim})) \cdot L(F(\mathrm{m}_{optim})) \quad (5)$$

Observație: Aflarea lui **C_FR2**(F0) nu mai implică alt efort computațional, fiind chiar valoarea C _FR1 pentru m_{optim}.

  - În funcție de valoarea **C_FR2**(F0) se ia decizia dacă pixelul central este sau nu rezultat din zgomot impulsiv. Daca **C_FR2**(F0) >= Ft (o constantă ce reprezintă valoarea de prag) atunci F0 nu este afectat de zgomot impulsiv, sunt destui pixeli vecini similari pentru a trage concluzia că pixelul central nu este impuls. În caz contrar pixelul central este afectat de zgomot impulsiv și va fi înlocuit prin VMF (*filtrarea vectorială marginală*).
  - După eliminarea zgomotului impulsiv urmează procedura de netezire a zgomotului gaussian. Pixelul F0 este înlocuit cu suma ponderată doar a pixelilor vecini ce aparțin grupului de similaritate.

$$Fout = \frac{\sum_{i=0}^{i=m_{optim}} FP\big(F(i)\big) * F(i)}{\sum_{i=0}^{i=m_{optim}} FP\big(F(i)\big)} \quad (7)$$

Unde FP(F(i)) este ponderea (gradul de similaritate) pixelului de pe poziția *i* după ordonare

Implementarea software a algoritmului prezentat

Implementarea se va face în Python în conformitate cu pașii algoritmului prezentați anterior.

Pe imagea color originală, reprezentată ca *numpy array*, se adaugă zgomot mixt (gaussian de deviație standard 10 și medie nulă și impulsiv cu 10% din pixeli afectați) cu funcția **add_mixed_noise**.

```
def add_mixed_noise(I):
    mean = 0
    stdDev = 10
    gaussianNoise = np.random.normal(mean,stdDev,[I.shape[0],I.shape[1],3])
    I = I + gaussianNoise
    I = np.clip(I, 0, 255)
    I = I.astype('uint8')
    pImpNoise = 0.1 # procent de pixeli afectati de zgomot impulsiv per canal de culoare
    N = int(pImpNoise*(I.shape[0]*I.shape[1])/3 ) # numar de pixeli afectati de zgomot impulsiv
    for channel in range(3):
        for j in range(N):
            linePos=np.random.randint(0,I.shape[0], [1,1])
            linePos=int(linePos)
            colPos=np.random.randint(0,I.shape[1], [1,1])
            colPos=int(colPos)
            I[linePos,colPos,channel]=(int(np.random.randint(0, 2, [1,1])))*255
            affectedPixels.append((linePos,colPos))
    return I
```

Filtrarea se va face pe ferestre de **n** x **n** (n va fi ales 3 pentru început). Parcurgerea pixelilor din imagine se va face evitând bordura de dimensiune $\frac{n-1}{2}$ unde nu există pixeli vecini suficienți. Alegem această metodă de ignorarea a pixelilor de la margine în locul *padding*-ului cu zerouri. Pixelul central din imaginea ce se vrea fără zgomot se înlocuiește cu valoarea returnată de funcția ce implementează algoritmul prezentat (**fuzzy_peer_groups_algorithm**).

```
borderSize = int((n-1)/2)
for i in range(borderSize, imgWithNoise.shape[0]-borderSize):
    for j in range(borderSize, imgWithNoise.shape[1]-borderSize):
        # F0 = imgWithNoise[i,j,:] (pixelul central)
        imgWithoutNoise[i,j,:] = fuzzy_peer_groups_algorithm(imgWithNoise, i, j)
```

Implementarea algoritmului se face conform pașilor descriși mai sus. Înainte însă se definește funcția de similaritate (**fuzzy_similarity_function**). Pentru reducerea complexității ulterioare a operației de sortare se folosește o structură de tip *dicționar* în care vor fi stocate sub *cheie* coordonatele pixelului și sub *valoare* gradul de similaritate al pixelului cu cel central. Se adaugă în dicționarul **similarityDict** fiecare pixel din fereastră. Ordonarea descrescătoare a pixelilor în funcție de gradul de similaritate se face direct pe dicționarul în cauză, iar valorile noi se reține în **sorted_similarityDict**. Valorile pentru funcțiile **C**, **A**, **L** și **C_FR1** se reținu sub formă de listă pornind de la dicționarul sortat. Se determină **m_optim** ca fiind indexul maximului din lista ce reprezintă C_FR1. **C_FR2** este valoarea din lista **C_FR1** de la indexul **m_optim**-1.

```
# functie de similaritate fuzzy
def fuzzy_similarity_function(Fi, Fj):
    return np.exp((-1)*(np.linalg.norm(Fi-Fj, ord=2)/Fsigma))

#algoritmul de filtrare (reducere de zgomot mixt - impulsiv si gaussian)
def fuzzy_peer_groups_algorithm(I, x, y):
    # window = I[i-borderSize:i+borderSize+1, j-borderSize:j+borderSize+1, :]
    F0 = I[x,y,:]
    # un dictionar in care se retin coordonatele pixelului (sub forma de cheie)
    # si functia de similaritate asociata (sub forma de valoare)
    similarityDict = dict()
    R = G = B = [] # pentru determinarea medianului marginal

    for i in range(x-borderSize, x+borderSize+1):
        for j in range(y-borderSize, y+borderSize+1):
            Fj = I[i,j,:]
            R.append(Fj[0])
            G.append(Fj[1])
            B.append(Fj[2])
            similarityDict[(i,j)] = fuzzy_similarity_function(F0, Fj)

    # Se sorteaza descrescator pixelii in functie de similaritate
    sorted_similarityDict = dict(sorted(similarityDict.items(), key=operator.itemgetter(1),
                                        reverse=True))
    # C este functia ce arata gradul de apartenenta a pixelului la grupul de similaritate
    C = list(sorted_similarityDict.values())
    # A este functia de asimilare de similaritate a pixelului (fct crescatoare)
    A = [sum(C[:i+1]) for i in range(len(C))]
    # L este o functie cuadratica L(F(i)) = u(A(F(i)))
    L = [(-(1/(pow(nSq-1, 2)))*(A[i]-1)*(A[i]-2*nSq+1)) for i in range(len(A))]
    # C_FR1 este certitudinea (Fuzzy Rule 1)
    C_FR1 = [C[m]*L[m] for m in range(1, nSq)]
    # m_optim este m pentru care C_FR1 are valoare maxima
    m_optim = C_FR1.index(max(C_FR1))+1
    # C_FR2 este certitudinea pixelului central (Fuzzy Rule 2)
    C_FR2 = C_FR1[m_optim-1]
```

Dacă **C_FR2** este mai mic decât valoarea de prag (**Ft**) pixelul central este afectat de zgomot impulsiv și trebuie înlocuit prin filtrarea vectoriala prin abordarea "distanță". Această filtrare presupune ca pornind de la un pixel median, format din componentele de culoare medii pe toți pixelii din fereastra de analiză, se sortează pixelii în funcție de distanța față de pixelul mediu și se alege drept înlocuitor pixelul de pe poziția de mijloc în lista sortată. După rezolvarea zgomotului impulsiv se returnează suma pixelilor din grupul de similaritate ponderați pe suma ponderilor pentru înlocuirea pixelului central.

```
    if (C_FR2 < Ft):
        # Pixelul F0 este afectat de zgomot. Trebuie inlocuit prin VMF
        detectedAffectedPixels.append((x, y))
        # Se implementeaza filtrarea vectoriala prin abordare "distanta"
        R.sort()
        G.sort()
        B.sort()
        neighbourPixels = dict()
        medianPixel = (R[math.ceil(nSq/2)],G[math.ceil(nSq/2)],B[math.ceil(nSq/2)])
        for i in range(x-borderSize, x+borderSize+1):
            for j in range(y-borderSize, y+borderSize+1):
                neighbourPixels[(i,j)] = np.linalg.norm(I[i,j,:]-medianPixel, ord=2)
        sorted_neighbourPixels = dict(sorted(neighbourPixels.items(), key=operator.itemgetter(1)))
        xy = list(sorted_neighbourPixels.keys())[0]
        pixelWithoutNoise = I[xy[0],xy[1],:]
    else:
        pixelWithoutNoise = F0
```

```
pixelsValuesInFuzzyPeerGroup = [I[i] for i in list(sorted_similarityDict.keys())[:m_optim]]
weightsInFuzzyPeerGroup = list(sorted_similarityDict.values())[:m_optim]
numerator = pixelWithoutNoise
denominator = 1
for i in range(1, m_optim):
    numerator = numerator + weightsInFuzzyPeerGroup[i]*pixelsValuesInFuzzyPeerGroup[i]
    denominator = denominator + weightsInFuzzyPeerGroup[i]
pixelWithoutNoise = numerator/denominator
pixelWithoutNoise = [int(x) for x in pixelWithoutNoise]
pixelWithoutNoise = np.clip(pixelWithoutNoise, a_min=0, a_max=255)
return pixelWithoutNoise
```

Conform studiului de referință [1] valorile celor doi parametrii întâlniți mai sus, **Fσ** și **Ft**, se aleg proporțional cu cantitatea de zgomot prezent în imagine. **Ft** depinde direct proporțional de procentul de pixeli afectați de zgomot impulsiv, iar **Fσ** depinde de valoarea deviației standard a zgomotului gaussian.

Percentage of impulse noise	F_t
Low (in $[0, 10]$)	0.05
Medium (in $[10, 20]$)	0.15
High (in $[20, 30]$)	0.25

(a)

Standard deviation σ of Gaussian noise	F_σ
Low (in $[0, 10]$)	50
Medium (in $[10, 20]$)	100
High (in $[20, 30]$)	175

(b)

Tabelul 1.1: Valori sugerate [1] *pentru (a) parametru Ft și pentru (b) parametrul Fσ*

Zgomotul impus în cerința proiectului este de 10% zgomot impulsiv și deviație standard de 10 pentru cel gaussian. Astfel, din aceste două valori impuse, se pot alege cei doi parametrii: **Fσ** 50 și **Ft** 0.15.

Dimensiunea ferestrei de analiză **n** x **n** se alege empiric observând rezultatele algoritmului la diferite valori ale lui **n** (e.g. 3, 5, 7). Se observă în **Figura 1.1** că odată cu creșterea dimensiunii ferestrei numărul de pixeli afectați de zgomot impulsiv detectați scade ca o consecință a unei vecinătăți mai largi, pragul de elemente din grupul de similaritate putând fi mult mai ușor depășit și astfel pixelul să fie considerat neafectat de zgomot. Totodată detaliile și contururile se estompează, suma ponderată făcându-se pe mai multe eșantioane.

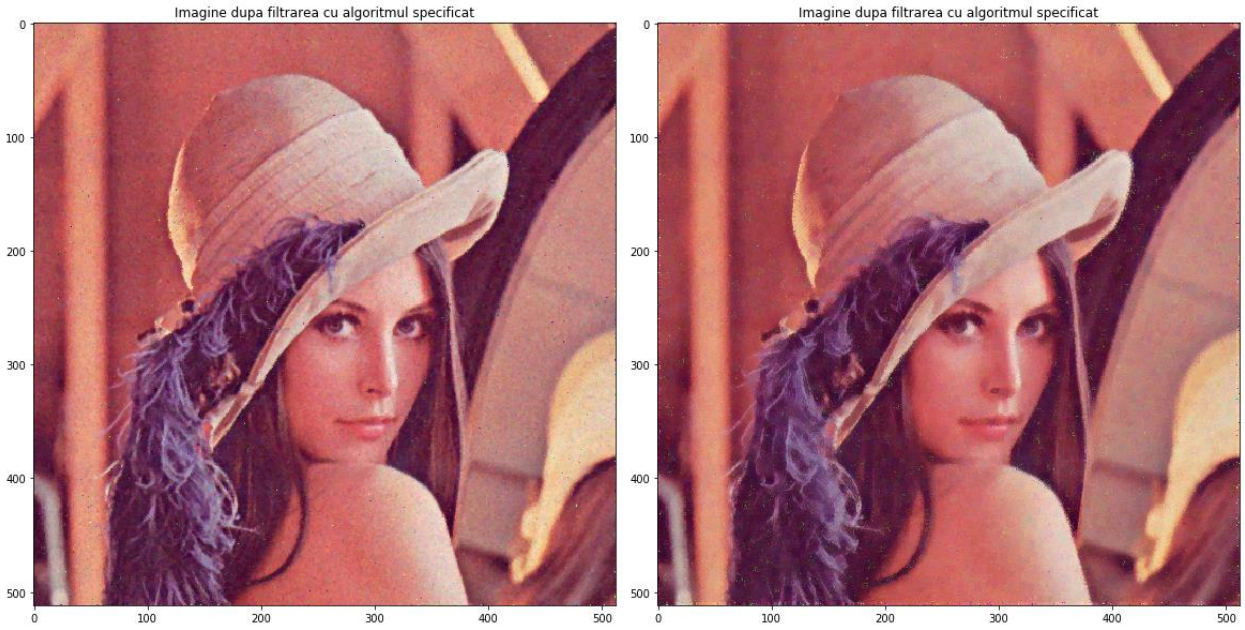

*Figura 1.1: Imaginea filtrată cu algoritmul specificat pe ferestre **n** x **n** cu **n** egal cu (stânga) 3 (dreapta) 5*

Se declară și definesc astfel cele trei constante (cei doi parametrii ai algoritmului și dimensiunea ferestrei) ca variabile globale în cod.

```
# Globals
Fsigma = 50
Ft = 0.15
n = 3 # nxn dimensiunea ferestrei de filtrare
nSq = pow(n, 2)
```

Rezultate experimentale și observații

Performanțele algoritmului de filtrare vor fi observate pe cinci imagini (de test de dimensiune 512x512). Se urmăresc atât aspectul general, ce poate fi concluzionat de observatorul uman, cât și anumite *metrici de performanță* ce oferă o măsură mult mai obiectivă. Metricile de performanță sunt necesare pentru testarea oricărei aplicație de reconstrucție sau filtrare de imagine, deoarece oferă o evaluare cantitativă [2]. Pentru algoritmul implementat vor fi folosite ca metrici de performanță *eroarea pătratică medie* (*Mean Square Error*) și *raportul semnal-zgomot vârf-la-vârf* (*Peak Signal-to-Noise Ratio*), mărime derivată din aceasta. Cele două mărimi se determină pe baza valorilor pixelilor din imaginea originală în raport cu valorile pixelilor pentru imaginea filtrată. Pentru simplificarea și lizibilitatea codului se alege ca cele două metrici să fie determinate folosind metodele *measure* ale modulului Python *skimage*: ***compare_mse*** și ***compare_psnr***.

```
print("PSNR pentru filtrarea clasica cu filtru median marginal: {}".format(skimage.measure.compare_psnr(img,I1)))
print("PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: {}".format(skimage.measure.compare_psnr(img,I2)))
print("PSNR pentru filtrarea cu algoritmul specificat: {}".format(skimage.measure.compare_psnr(img,imgWithoutNoise)))
print("")
print("MSE pentru filtrarea clasica cu filtru median marginal: {}".format(skimage.measure.compare_mse(img,I1)))
print("MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: {}".format(skimage.measure.compare_mse(img,I2)))
print("MSE pentru filtrarea cu algoritmul specificat: {}".format(skimage.measure.compare_mse(img,imgWithoutNoise)))
```

De asemenea, performanțele algoritmului implementat vor fi comparate și cu rezultatele unor filtre clasice: *filtrul median marginal* pentru eliminarea zgomotului impulsiv și *filtrul de medie aritmetică marginal* pentru eliminarea zgomotului gaussian. *Filtrarea mediană marginală* se construiește parcurgând toată imaginea pe ferestre de dimensiune 3x3, se constriește o listă ordonată de câte nouă elemente pentru fiecare plan de culoare, iar pixelul central ferestrei este înlocuit cu cele trei valori mediane (de la indexul 4) din fiecare listă. Astfel se elimină orice posibilitate de

```
def marginal_median_filter(I):
    # Filtru median marginal
    newI = I.copy()
    ch = I.shape[2]
    for i in range(1, I.shape[0]-1):
        for j in range(1, I.shape[1]-1):
            red = np.sort(I[i-1:i+2,j-1:j+2,0], axis=None)[4]
            green = np.sort(I[i-1:i+2,j-1:j+2,1], axis=None)[4]
            blue = np.sort(I[i-1:i+2,j-1:j+2,2], axis=None)[4]
            newI[i,j,:] = np.array([red,green,blue])
    return newI

def marginal_arithmetic_mean_filter(I):
    # Filtru de medie aritmetica marginal de dimensiune 3x3
    newI = I.copy()
    ch = I.shape[2]
    for i in range(1, I.shape[0]-1):
        for j in range(1, I.shape[1]-1):
            redMean = int(np.mean(I[i-1:i+2,j-1:j+2,0]))
            greenMean = int(np.mean(I[i-1:i+2,j-1:j+2,1]))
            blueMean = int(np.mean(I[i-1:i+2,j-1:j+2,2]))
            newI[i,j,:] = np.array([redMean,greenMean,blueMean])
    return newI
```

apariție de zgomot impulsiv, însă se corupe culoarea în sine a pixelului. *Filtrarea marginală prin mediere aritmetică* presupune ca parcurgând în același mod imaginea, pe ferestre de 3x3, elementele de culoare (R, G și B) pixelului central să fie înlocuite cu media elementelor pentru toți pixelii din fereastră pentru a se elimina zgomotul gaussian.

În continuare vor fi prezentate rezultatele pentru cele cinci imagini de test.

Figura 1.2: Imaginea originală (sus stânga), imaginea afectată de zgomot mixt (sus dreapta)

Imaginea filtrată cu filtrul median marginal (jos stânga) și cea prin mediere aritmetică (jos dreapta)

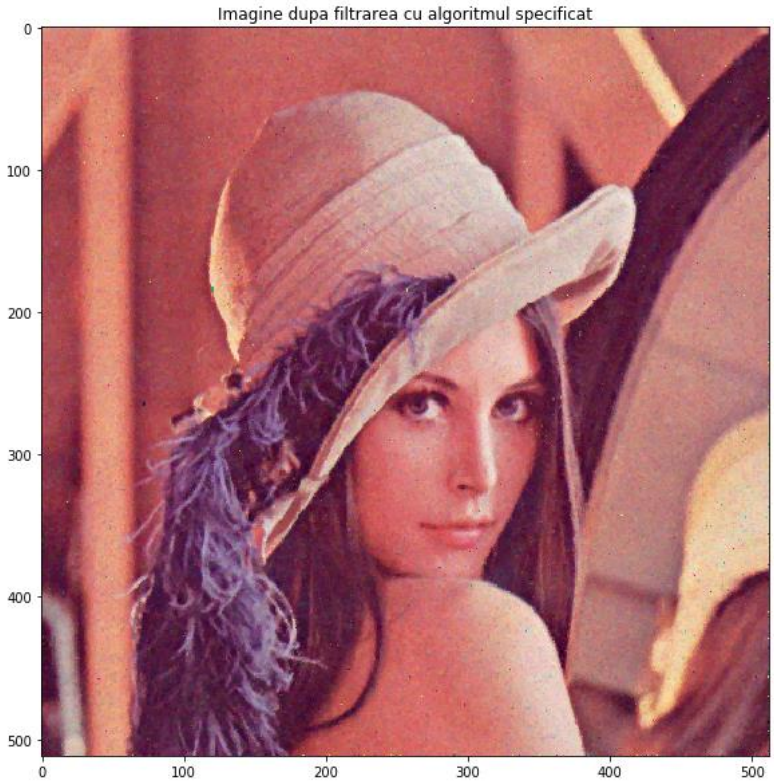

Figura 1.3: Imaginea filtrată prin algoritmul Fuzzy Peer Group

PSNR pentru filtrarea clasica cu filtru median marginal: 30.6

PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: 30.26

PSNR pentru filtrarea cu algoritmul specificat: 26.19

MSE pentru filtrarea clasica cu filtru median marginal: 56.61

MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: 61.17

MSE pentru filtrarea cu algoritmul specificat: 156.23

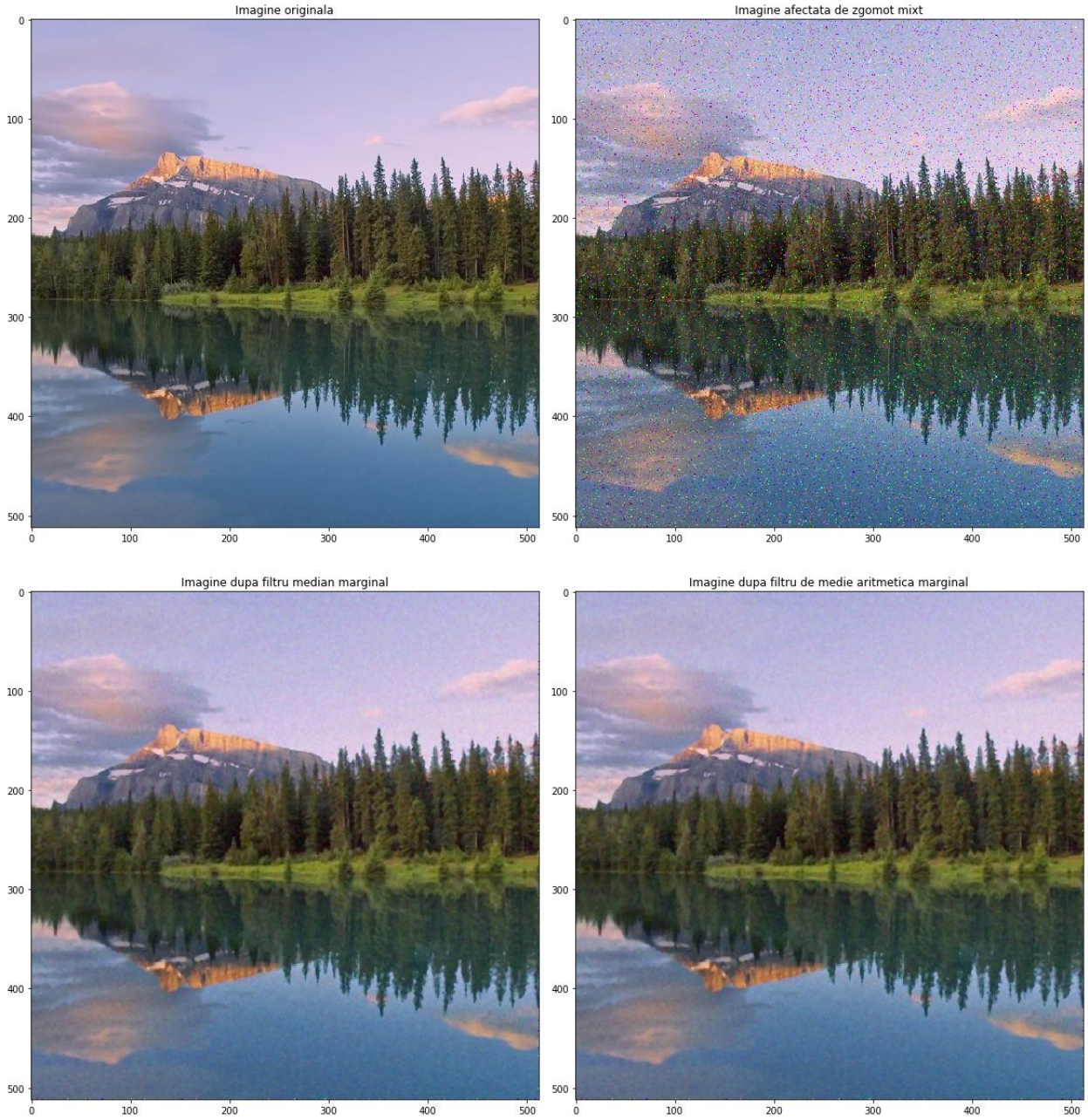

Figura 1.4: Imaginea originală (sus stânga), imaginea afectată de zgomot mixt (sus dreapta)

Imaginea filtrată cu filtrul median marginal (jos stânga) și cea prin mediere aritmetică (jos dreapta)

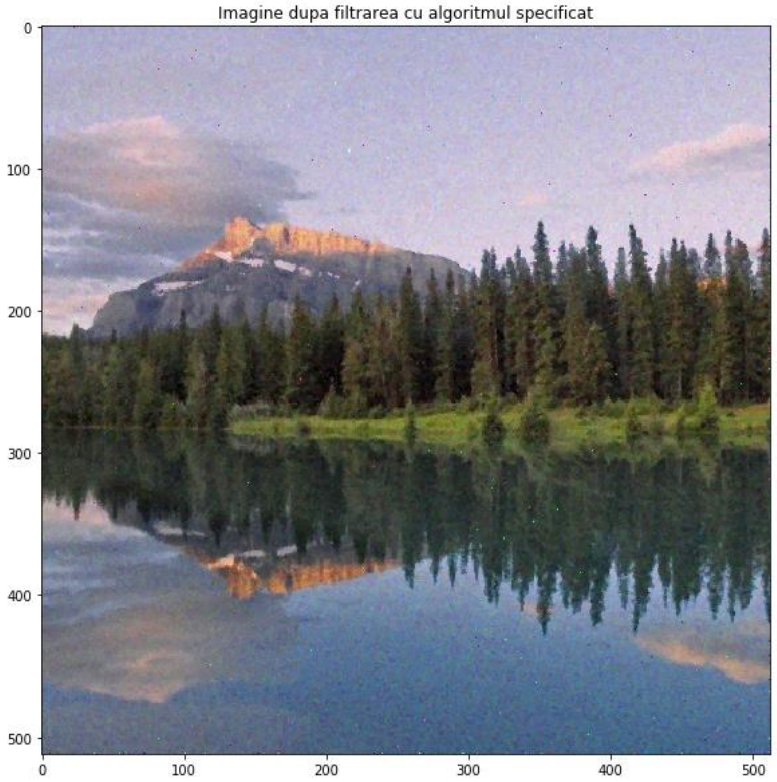

Figura 1.5: Imaginea filtrată prin algoritmul Fuzzy Peer Group

PSNR pentru filtrarea clasica cu filtru median marginal: 28.67

PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: 28.58

PSNR pentru filtrarea cu algoritmul specificat: 25.86

MSE pentru filtrarea clasica cu filtru median marginal: 88.26

MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: 90.15

MSE pentru filtrarea cu algoritmul specificat: 168.53

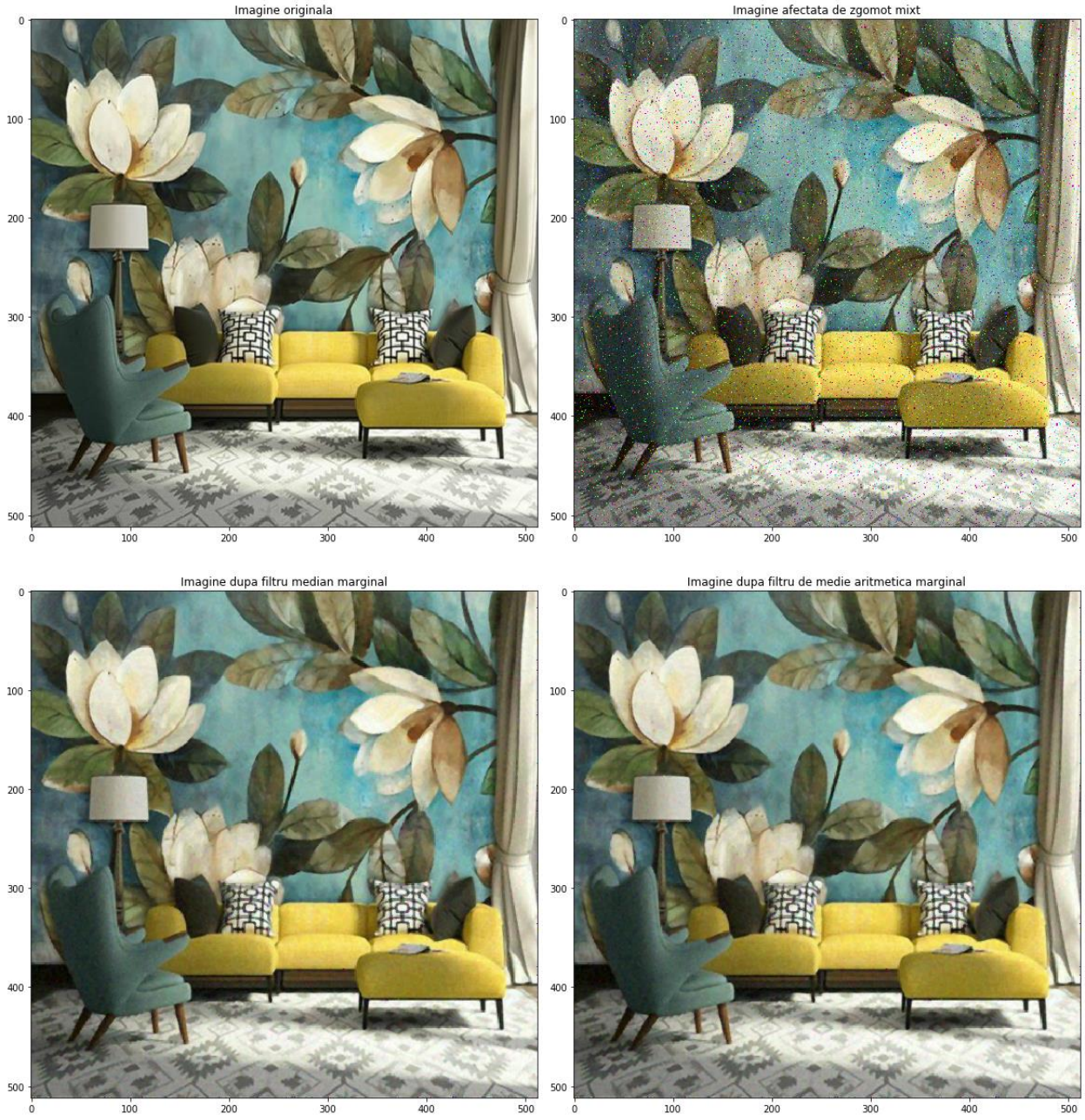

Figura 1.6: Imaginea originală (sus stânga), imaginea afectată de zgomot mixt (sus dreapta)

Imaginea filtrată cu filtrul median marginal (jos stânga) și cea prin mediere aritmetică (jos dreapta)

Figura 1.7: Imaginea filtrată prin algoritmul Fuzzy Peer Group

PSNR pentru filtrarea clasica cu filtru median marginal: 29.72

PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: 27.98

PSNR pentru filtrarea cu algoritmul specificat: 25.10

MSE pentru filtrarea clasica cu filtru median marginal: 69.24

MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: 103.37

MSE pentru filtrarea cu algoritmul specificat: 200.69

Figura 1.8: Imaginea originală (sus stânga), imaginea afectată de zgomot mixt (sus dreapta)

Imaginea filtrată cu filtrul median marginal (jos stânga) și cea prin mediere aritmetică (jos dreapta)

Figura 1.9: Imaginea filtrată prin algoritmul Fuzzy Peer Group

PSNR pentru filtrarea clasica cu filtru median marginal: 25.02

PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: 24.03

PSNR pentru filtrarea cu algoritmul specificat: 23.07

MSE pentru filtrarea clasica cu filtru median marginal: 204.37

MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: 256.80

MSE pentru filtrarea cu algoritmul specificat: 320.3

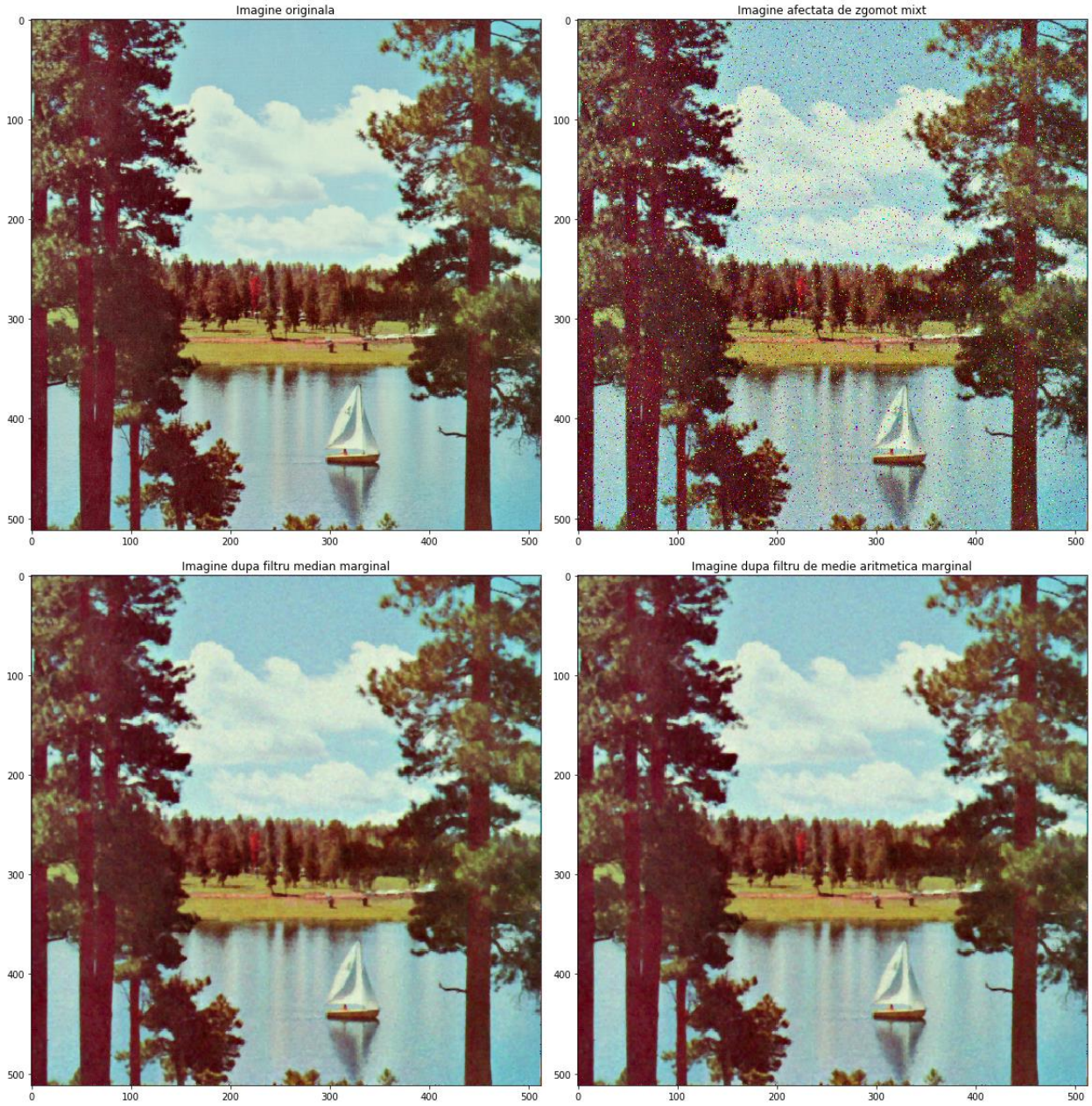

Figura 1.10: Imaginea originală (sus stânga), imaginea afectată de zgomot mixt (sus dreapta) Imaginea filtrată cu filtrul median marginal (jos stânga) și cea prin mediere aritmetică (jos dreapta)

Figura 1.11: Imaginea filtrată prin algoritmul Fuzzy Peer Group

PSNR pentru filtrarea clasica cu filtru median marginal: 27.38

PSNR pentru filtrarea clasica cu filtru de medie aritmetica marginal: 26.67

PSNR pentru filtrarea cu algoritmul specificat: 24.16

MSE pentru filtrarea clasica cu filtru median marginal: 118.82

MSE pentru filtrarea clasica cu filtru de medie aritmetica marginal: 139.68

MSE pentru filtrarea cu algoritmul specificat: 249.37

Figura 1.12: Imaginea afectată de zomot impulsiv (10%) și zgomot gaussian (deviație standard 5)

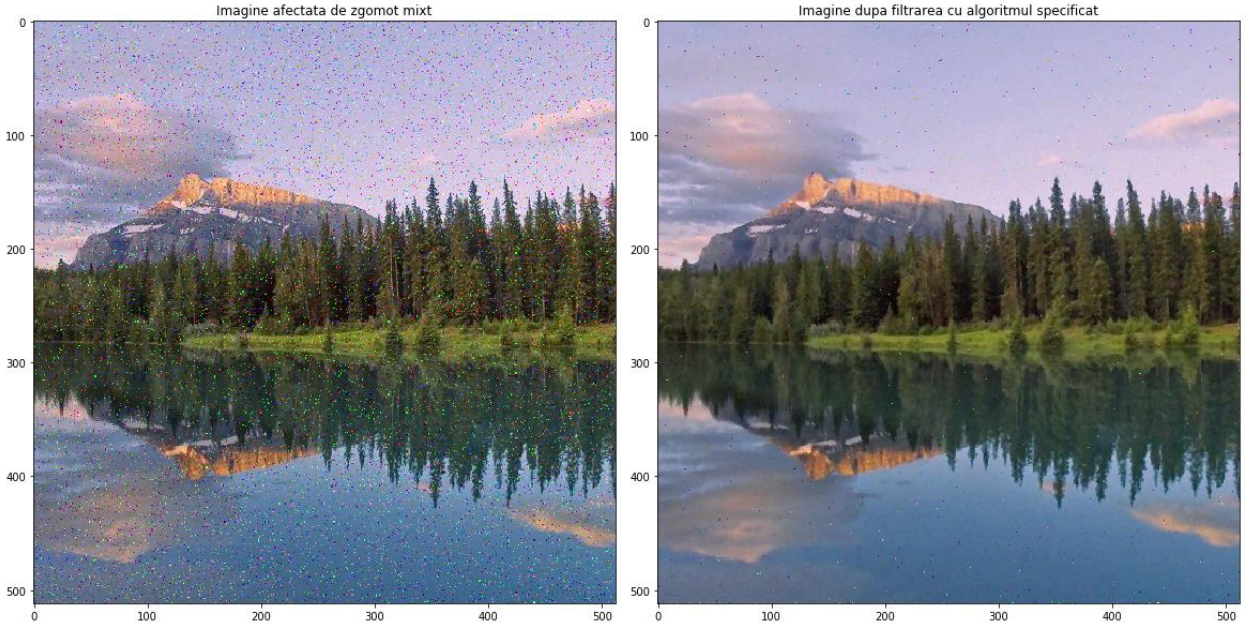

Figura 1.13: Imaginea afectată de zomot impulsiv (10%) și zgomot gaussian (deviație standard 5)

Figura 1.14: Imaginea afectată de zomot impulsiv (25%) și zgomot gaussian (deviație standard 10)

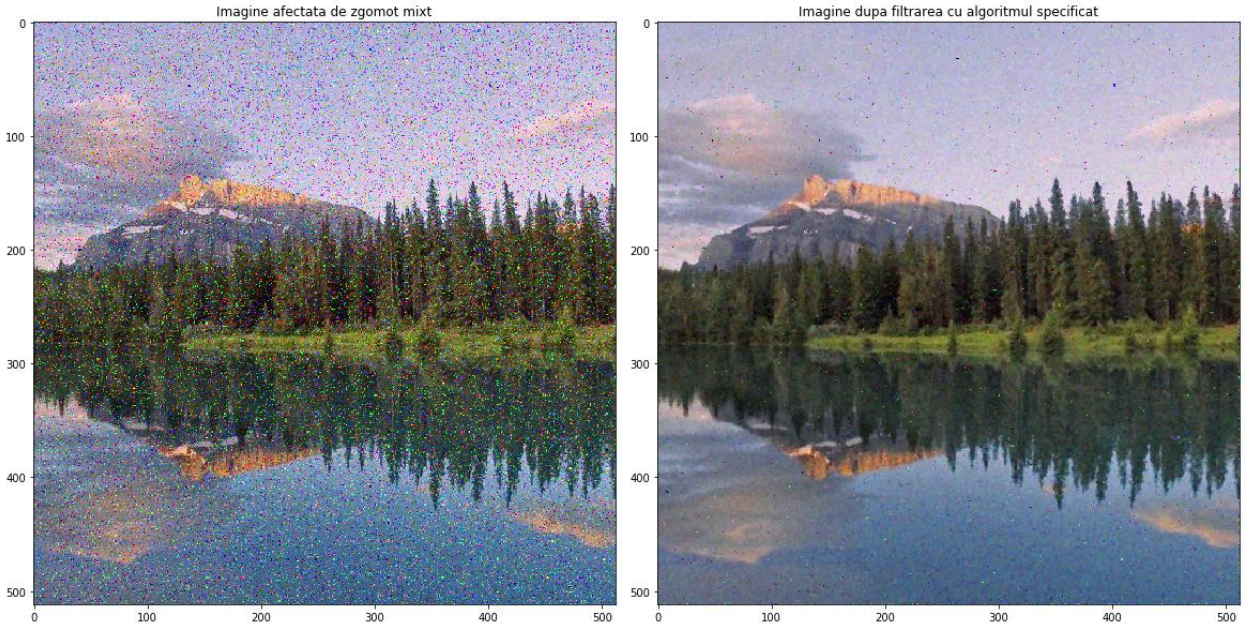

Figura 1.15: Imaginea afectată de zomot impulsiv (25%) și zgomot gaussian (deviație standard 10)

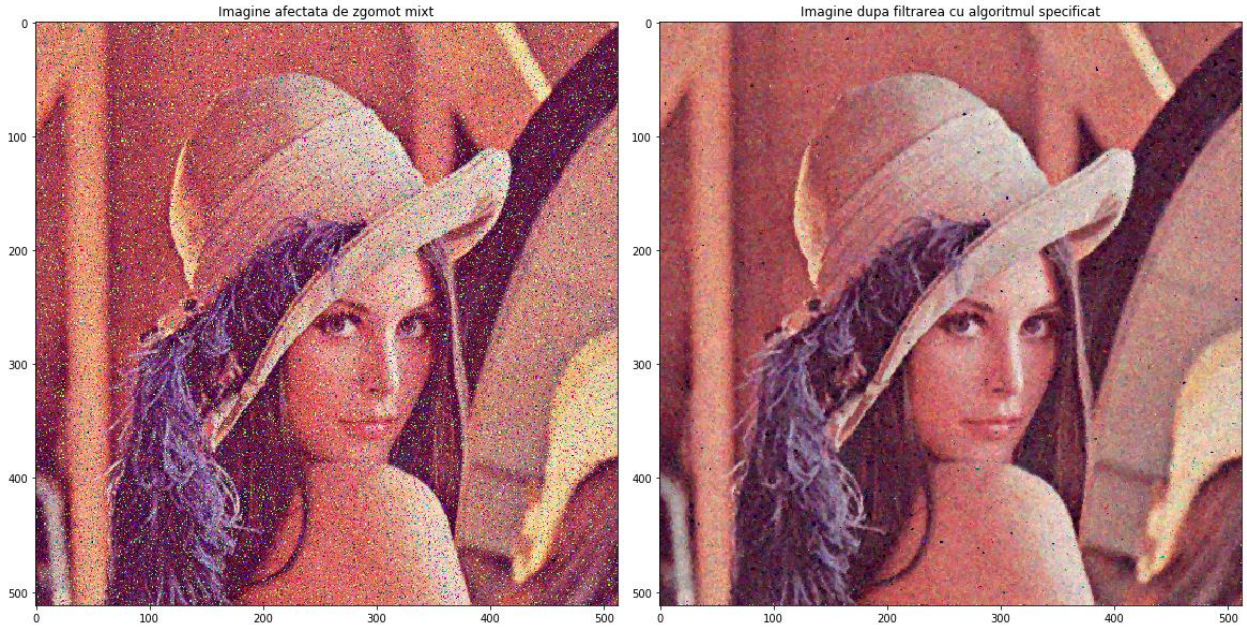

Figura 1.16: Imaginea afectată de zomot impulsiv (30%) și zgomot gaussian (deviație standard 20)

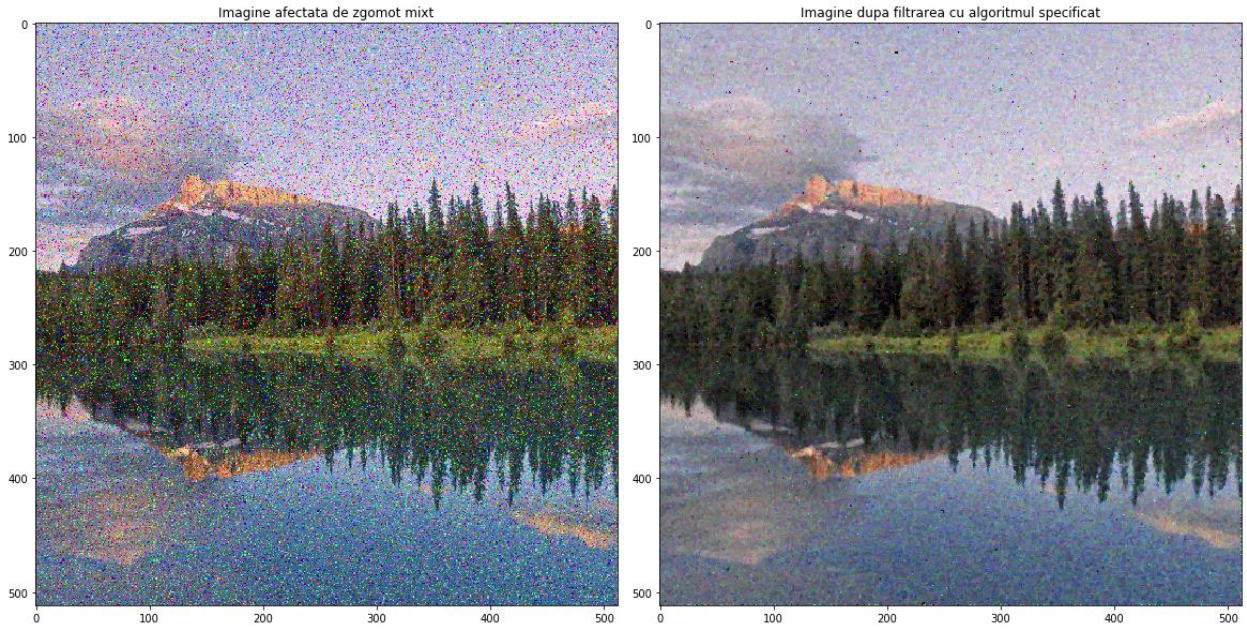

Figura 1.17: Imaginea afectată de zomot impulsiv (30%) și zgomot gaussian (deviație standard 20)

Se observă că performanțele algoritmului de filtrare sunt acceptabile până într-o anumită limită de zgomot. În comparație cu cele două filtre clasice, filtrul prezentat oferă rezultate comparabile, în plus elimină ambele tipuri de zgomot într-o singură iterație. Principalul neajuns este că, în cazul ferestrelor unde pixelul central este afectat de zgomot impulsiv, deși acesta este detectat și eliminat, reducerea zgomotului gaussian este slabă, grupul de similaritate fiind redus, uneori format doar dintr-un singur pixel.

Bibliografie

[1] V. G. a. A. H. Samuel Morillas, "Fuzzy Peer Groups for Reducing Mixed Gaussian-Impulse Noise From Color Images," *IEEE TRANSACTIONS ON IMAGE PROCESSING,* 2009.

[2] D. R. Srivastava, "Performance Measurement of Image Processing Algorithms," Varanasi, ITBHU.