Algoritm de filtrare de zgomot în imagini color

1. Descrierea filtrului implementat (algoritmul)

Algoritmul propus este un filtru median bazat pe cei mai apropiați K vecini de tip M, adaptive, fără parametri, pe mai multe canale folosit pentru eliminarea zgomotului impulsiv la imagini color. [1]

Se combină de fapt 2 filtre: unul pe mai multe canale fără parametri și unul bazat pe cei mai apropiați K vecini de tip M. [1]

Astfel se au în vedere următoarele 5 formule pe baza căruia se contruiește filtrul dorit:

$$\hat{x}(y)_{AMN-MMkNN} = \sum_{l=1}^{n} x_l^{VMMKNN} \left(\frac{h_l^{-M} K(y - y_l / h_l)}{\sum_{l=1}^{n} h_l^{-M} K(y - y_l / h_l)} \right) \quad (1)$$

unde y = punctul curent afectat de zgomot ce trebuie estimat dintr-un set dat y_N[1], y_l = vectorul de culoare(valoare) care se procesează, h_l = parametru de netezire[1], $\hat{x}$ = ieșirea filtrului(imaginea filtrată), n=N (adică numărul de puncte extrase de vecinătate = dimensiunea ferestrei de filtrare = 25 deoarece în articol se folosește o fereastră de 5x5). Mai exact pentru a calcula ieșirea filtrului propus se combină liniar ponderat valorile de tip median x_l^{VMMKNN} asociate fiecărui pixel, indexat de "l". Asta înseamnă că trebuie să fie calculate valorile x_l^{VMMKNN} pentru toate punctele din vecinătatea folosită (fereastra glisantă) => trebuie să fie calculate aceste valori în fiecare pixel. Aceste valori vor fi calculate cu formula (3).

Se continuă cu formula (2) care arată cum se calculează h_l :

$$h_l = n^{-p/M} \sum_{j=1}^{n} \left| y_j - y_l \right| \quad (2)$$

unde „$y_j \neq y_l$ pentru $\forall y_j$, j=1,2,…,N, $\left| y_j - y_l \right|$ = distanța absolută dintre cei 2 vectori"[1](în acest caz se dorește suma diferențelor de valoare de la vectorul de valoare curent y_l la toți ceilalți din fereastra de filtrare, iar asta se calculează pentru fiecare pixel din fereastră, adică l=1,…,n, j=1,…,n), p = „parametru ce se alege în plaja de valori [0,0.5]"[1] (în situația de față s-a ales p=0.3), M = „dimeniunea spațiului de măsurare(M=3 pentru imagini color)"[1]. „Funcția K(y) este funcția nucleu ce are forma exponențială K(y) = exp(-|y|) pentru zgomot impulsiv"[1]. Dar în situația de față se va alege funcția nucleu = 1 pentru ușurința calculelor și deoarece reprezintă cea simplă formă de calcul pe baza tabelului 1 din articol(VMMKNN (simplest)).

Se continuă cu formula (3) menționată anterior:

$$x_l^{VMMKNN} = \hat{e}_{VMMKNN}^{q} = MED\left\{ g^{(q)} \right\} \quad (3)$$

unde „$g^{(q)}$ = set de valori K_c ale pixelilor care sunt ponderate în conformitate cu funcția de influență $\widetilde{\Psi}_{\left(y_m \right)}$ utilizată într-o fereastră de filtrare și sunt cele mai apropriate ca valoare de estimarea obținută la pasul anterior $\hat{e}_{VMMKNN}^{(0)}$"[1](adică K_c = „numărul celor mai apropiați

pixeli”[1] = numărul de valori care se selectează din fereastra de filtrare dată și se folosesc pentru a găsi medianul pentru noua iterație. Cele K_c valori selectate sunt cele mai apropriate de precendentul median calculat la iterația anterioară q-1), „y_N = set de pixeli care provin din imaginea cu zgomot y_m, m=1,…,N într-o fereastră glisantă; $\hat{e}^{(0)}_{VMMKNN}$ = $y_{(N+1)/2}$ = estimatul inițial(valoarea de la care se pornește) care este egal cu pixelul central din fereastra de filtrare; q = indexul iterației curente.”[1]

Se continuă cu formula (4) care arată cum se determină K_c:

$$K_c = \left[K_{\min} + a \cdot D_S(y_{(N+1)/2}) \right] \le K_{\max} \quad (4)$$

unde „a = controlează conservarea detaliilor”[1] (iar în situația de față deoarece în referința [10] care de fapt în articol constituie referința [4], s-a ales a=4 ca valoare optimă, iar în implementare se va folosi aceeași valoare); „K_{min} = numărul minim de vecini pentru eliminarea zgomotului”[1] (iar în situația de față deoarece în referința [10] s-a ales K_{min}=5 ca valoare optimă, în implementare se va folosi aceeași valoare); „K_{max} = numărul maxim de vecini pentru restricționarea marginilor și netezirea detaliilor”[1] (K_c se va calcula astfel încât să fie între 1 și N=25, din acest motiv folosindu-se în formula (5) K_{min}, dar K_{max} < N); „$D_S(y_{(N+1)/2})$ = detectorul de vârf.”[1]

Se continuă cu ultima formulă, și anume (5) care arată cum se calculează $D_S(y_{(N+1)/2})$:

$$D_S(y_{(N+1)/2}) = \left[\frac{MED\left\{ \left| y_{(N+1)/2} - y_m \right| \right\}}{MAD} \right] + \left[\frac{1}{2} \cdot \frac{MAD}{MED\left\{ y_m \right\}} \right] \quad (5)$$

Altfel spus se folosește o fereastră pătrată, de dimensiune impară, în care s-au selectat N valori $y_1,\ldots,y_N$, adică mulțimea de valori $\{y_m\}$; $y_{(N+1)/2}$ corespunde valorii pixelului current (adică pixelul din centrul ferestrei de filtrare). Operatorii folosiți în această formulă sunt MED = median (acest lucru înseamnă că pe setul de valori se face o operație de ordonare a valorilor, după care se ia valoarea din mijloc), MAD = median al diferențelor absolute față de median (adică se face medianul pe setul de numere; se scade medianul din fiecare valoare din set, iar această operație se face în modul; după care se face medianul valorilor care au rezultat la punctul anterior). În cazul de față $MED\{y_m\}$ = median din setul de valori din fereastră, $MED\left\{ y_{(N+1)/2} - y_m \right\}$ = median din setul diferențelor absolute dintre valorile din fereastră și valoarea din punctul central al ferestrei, $MAD = MED\left\{ \left| y_m - MED\{y_m\} \right| \right\}$ = median din diferențele absolute față de median, numărul D_S este asociat punctului curent prelucrat.

„Algoritmul se termină când $\hat{e}^{(q)}_{VMMKNN} = \hat{e}^{(q-1)}_{VMMKNN}$.”[1]

2. Descrierea modului de implementare

Algoritmul la nivel de ansamblu ar arăta astfel: se începe prin importarea bibliotecilor, declararea parametrilor globali folosiți, citirea imaginilor care se vor prelucra, crearea variabilelor pentru imaginile afectate de zgomot, crearea zgomotului gaussian și impulsiv conform referinței [7], aplicarea zgomotului pe imagini, crearea imaginilor cu padding pentru imaginile afectate de zgomot, crearea imaginii care va fi folosită la finalul funcției filtrului propus de articol, definirea funcțiilor ce calculează PSNR și SSIM preluate de la referința [2], definirea funcțiilor pentru MED, MAD, detectorul de vârf D_S, parametrul de netezire h_l, funcția nucleu K(y) și filtrul AMN-MMKNN, afișarea imaginii originale,

imaginilor afectate de zgomotul gaussian și impulsiv, împreună cu cele filtrate(pentru filtrare s-au folosit funcțiile cv2.GaussianBlur() preluată din referința [3] și funcția cv2.medianBlur() preluată din referința [4], ambele aplicate pe vecinătăți de 3x3) și afișarea valorilor de PSNR și SSIM pentru imagile folosite.

O analiză mai detaliată se va aplica pentru funcțiile principale folosite în cadrul algoritmului:

- functia **med**

```
def med(window, scalar=-1): #scalar = -1 => optional argument
    if scalar != -1:
        for i in range(len(window)):
            window[i] = abs(window[i] - scalar)
    window.sort()
    values = window[:]
    while len(values) != 0 and values[int(np.floor(len(values)/2))] == 0:#check to see if 0 exist
s in window
        values.pop(int(np.floor(len(values)/2))) # eliminate values of 0
    if len(values) != 0:# if the are non-zero elements
        return (values[int(np.floor(len(values)/2))])#return the middle value
    else:
        return (1)
```

S-a ales crearea funcției **med** cu un parametru obligatoriu(fereastra de filtrare) și unul opțional(scalar) pentru a se putea implementa mai ușor formula (5) care în prima jumătate folosește 2 argumente la numărător, iar în a doua jumătate folosește un singur argument la numitor. În cazul în care scalar este obligatoriu se calculează medianul conform $MED\left\{\left|y_{(N+1)/2}-y_m\right|\right\}$, după care se sortează fereastra de filtrare, se creează o copie a acesteia, se verifică dacă există valori de 0 în fereastră pentru a evita o eventuala împărțire la 0, iar dacă există se elimină din fereastră. În cazul în care există valori nenule în fereastră se va returna valoarea din mijloc, altfel se va returna 1.

- funcția **mad**

```
def mad(window):
    return med(window, scalar=med(window))
```

Funcția **mad** doar returnează medianul folosind fereastra de filtrare și medianul ferestrei de filtrare(valoarea din centrul ferestrei de filtrare) ca argumente.

- funcția **spike_detector**

```
def spike_detector(window, scalar): # window = ym , scalar = y_(N+1)/2 from the article
    med result = med(window,scalar)
    mad_result = mad(window)
    first_half = np.floor(np.divide(med_result,(mad_result)))
    second_half = np.floor(0.5*(np.divide(mad(window),med(window))))
    return first_half + second_half
```

Funcția **spike_detector** primește ca argumente fereastra de filtrare și pixelul central din fereastra de filtrare. Se calculează inițial $MED\left\{\left|y_{(N+1)/2}-y_m\right|\right\}$ din formula (5), după care

MAD, urmând în continuare să se calculeze prima jumătate și a doua jumătate a formulei (5), iar la final se va returna suma celor 2 jumătăți.

- funcția **smooth_parameter**

```
def smooth_parameter(window,scalar): # window = yj , scalar = yl from the article
    suma = 0
    for i in range(len(window)):
        suma = suma + abs(window[i] - scalar)
    return n**(-p/M) * suma
```

Funcția **smooth_parameter** primește ca argumente fereastra de filtrare și scalar care reprezintă y_l din articol. Se calculează practic suma diferențelor de valoare de la vectorul de valoare curent la toți ceilalți din fereastra de filtrare, iar asta se calculează pentru fiecare pixel din fereastră. După care se returnează la final suma înmulțită cu $n^{-p/M}$.

- Funcția **kernel_function**

```
def kernel_function(window, current_pixel, neighbour_pixel):
    #return math.exp(-
abs(current_pixel - neighbour_pixel/smooth_parameter(window,neighbour_pixel)))
    return 1
```

Funcția **kernel_function** va returna 1 pentru a simplifica calculul și a scădea complexitatea algoritmului.

- funcția **amn_mmknn_filter**

```
def amn_mmknn_filter(image):
  for ch in range(image.shape[2]): # iterate through channels
      channel = image[:,:,ch] # get each channel of the image
      vmmknn = np.zeros(shape=channel.shape) # vmmknn values from the article
      for i in range(2, len(channel) - 2): # iterate rows
          for j in range(2, len(channel[0]) - 2): # iterate columns
              stopCond = False # break condition for while loop
              window=[] # 5x5 window array
              for length in range(i-2,i+3):
                  for width in range(j-2,j+3):
                      window.append(channel[length][width]) # get neighbours pixels of
 current pixel
              idx = 12 # y (N+1)/2 from article
              idx_list = [] # store the past indexes from which we iterated through
              counter = 0
              while stopCond == False: # while e^VMMKNN(q) != e^VMMKNN(q+1)
                  Kc dict = {} # key = index , value = result of the number of the nea
rest neighbour pixels
                  e_dict = {} # key = e^VMMKNN(q) , value = number of the nearest neig
hbour pixels for e^VMMKNN(q)
                  e index = idx # e^VMMKNN(q)
                  for k in range(len(window)): # iterate through window pixels
```

```
                    result = K_min + a * spike_detector(window,window[k]) # Kc from
the article
                    if k == e_index: # check if pixel index is the same as e^VMMKNN(
q)
                        e_dict[e_index] = result
                    elif result <= K_max: # check if result for pixel index is small
er than Kmax
                        Kc_dict[k] = result
                diff_dict = {} # dictionary for the differences between the Kc value
s
                for key in Kc_dict.keys():
                    diff = abs(Kc_dict[key] - e_dict[e_index])
                    diff_dict[key] = diff
                diff_dict = dict(sorted(diff_dict.items(), key=lambda item: item[1])
) # sort dictionary by values
                idx = int(len(diff_dict)/2) # get central index of difference dictio
nary
                counter += 1
                if idx == e_index or e_index in idx_list: # check if e^VMMKNN(q) = e
^VMMKNN(q+1)
                    stopCond = True
                idx_list.append(idx)
            vmmknn[i][j] = e_dict[e_index]
    filter_output = np.zeros(shape=channel.shape).astype(np.uint8) # filtered image
    for i in range(2, len(channel) - 2): # iterate rows
        for j in range(2, len(channel[0]) - 2): # iterate columns
            vmmknn_window = [] # vmmknn for the valid pixels(non-padded pixels)
            window = [] # valid neighbour pixels
            for length in range(i-2,i+3):
                for width in range(j-2,j+3):
                    if length >= 2 and length <= len(channel) - 3: # check for valid
 pixels
                        if width >= 2 and width <= len(channel[0]) - 3:
                            vmmknn_window.append(vmmknn[length][width])
                            window.append(channel[length][width])
            ###### Compute the (1) equation from the article######
            filter_out = 0
            result_jos = 0
            for l2 in range(len(window)):
              result_jos = result_jos + smooth_parameter(window,window[l2]) * kernel
_function(window, channel[i][j], window[l2])
            for l1 in range(len(window)):
              filter_out = filter_out + vmmknn_window[l1] * (smooth_parameter(window
, window[l1]) * kernel_function(window, channel[i][j], window[l1])) / result_jos
            filter_output[i][j] = int(filter_out)
```

```
            ##########################################################
    new_image[:,:,ch] = filter_output # filter for each channel
return new_image
```

Funcția **amn_mmknn_filter** primește ca argument imaginea care se dorește a fi filtrată. Se începe prin parcurgerea canalelor imaginii. Pentru fiecare canal se creează o matrice *vmmknn* în care se vor stoca valorile x_l^{VMMKNN} calculate. Se continuă prin parcurgerea liniilor și coloanelor canalului curent procesat, se creează o variabilă de tip boolean pentru a fi folosită în condiția de stop din bucla while, se creează o variabilă pentru fereastra de filtrare de 5x5 care se populează cu vecinii pixelului curent, se creează o variabilă *idx* care se inițializează cu 12 deoarece este valoarea din mijlocul ferestrei de 5x5, se creează o listă *idx_list* în care se vor stoca indexii care au fost deja parcurși. Se începe bucla while în care condiția de stop este $\hat{e}_{VMMKNN}^{(q)} = \hat{e}_{VMMKNN}^{(q-1)}$, în interiorul buclei while se creează două dicționare *Kc_dict* și *e_dict*, unde pentru primul cheia este reprezentată de index și valoarea este dată de rezultatul numărului de cei mai apropiați pixeli vecini, iar pentru al doilea cheia este dată de $\hat{e}_{VMMKNN}^{(q)}$ iar valoarea este dată de numărul celor mai apropiați pixeli vecini pentru $\hat{e}_{VMMKNN}^{(q)}$. Se continuă cu o variabilă *e_index* care este inițializată cu *idx* descris anterior, se parcurge fereastra de pixeli unde în variabila *result* se calculează valoarea K_c din articol pentru fiecare pixel din fereastră, se verifică dacă indexul pixelului curent este la fel cu $\hat{e}_{VMMKNN}^{(q)}$, iar dacă da se va memora în *e_dict* la indexul *e_index* valoarea *result*, altfel se compară ce s-a calculat(*result*) cu K_{max} în așa fel încât să fie mai mic decât K_{max}, iar dacă da, în dicționarul *Kc_dict* la poziția care corespunde indexului pixelului curent se va stoca *result*. Mai departe se creează un dicționar *diff_dict* pentru a memora diferențele dintre valorile K_c, se parcurge *Kc_dict* pe baza cheilor unde se fac diferențele absolute dintre valorile K_c iar rezultatele se pun în *diff_dict*, urmând ca după să se sorteze *diff_dict* după valori pentru a lua pixelul central din dicționarul de diferențe, se incrementează contorul, se verifică dacă $\hat{e}_{VMMKNN}^{(q)} = \hat{e}_{VMMKNN}^{(q-1)}$, iar dacă da variabila booleană creată anterior din False devine True, se memorează *idx* la care s-a ajuns în *idx_list*, iar la ieșirea din while se memorează în fereastra *vmmknn* pixelul al cărui index este același cu $\hat{e}_{VMMKNN}^{(q)}$.

Mai departe se creează o matrice numită *filter_output* pentru imaginea de la ieșirea filtrului, se parcurg liniile și coloanele canalului curent prelucrat unde se creează două liste: *vmmknn_window* pentru pixelii valizi(cei care nu au padding) și *window* pentru pixelii vecini valizi. În continuare se parcurge fereastra de 5x5 și se verifică atât pe linie cât și pe coloană că nu sunt pixeli ce provin din padding, iar în caz pozitiv se populează cele două liste. Urmează să se calculeze ecuația (1) din articol: pentru asta se creează două variabile *filter_out* care va fi folosită pentru a popula *filter_output* și *result*_jos care va fi folosită pentru a calcula $\sum_{l=1}^{n} h_l^{-M} K(y - y_l / h_l)$ din formula (1) de la numitor. Se începe mai întâi prin a calcula $\sum_{l=1}^{n} h_l^{-M} K(y - y_l / h_l)$ folosindu-se pentru asta ciclul for cu *l2*, iar pe urmă calculează restul formulei cu ciclul for cu *l1* unde pentru fiecare

pixel din fereastra de filtrare de 5x5 se stochează rezultatul în *filter_out* iar mai departe cu acest rezultat se populează *filter_output*, urmând ca la final de tot să se pună în imaginea finală pe fiecare canal corespunzător imaginii filtrate.

3. Experimente

Pentru experimente s-au folosit imagini cu dimensiunea de 64x64 deoarece acest lucru a fost dictat de complexitatea algoritmului care este foarte mare, și pentru a avea un timp de execuție rezonabil. Dimensiunea mică a imaginilor poate duce la o degradare a calității imaginilor, dar acesta este compromisul pentru un timp de rulare al algoritmului cât mai mic. De asemenea, pe cele 5 imagini de test nu se va putea adăuga un zgomot aditiv de deviație standard 10 deoarece zgomotul aplicat ar fi prea puternic și nu s-ar mai putea distinge nimic în imagine.

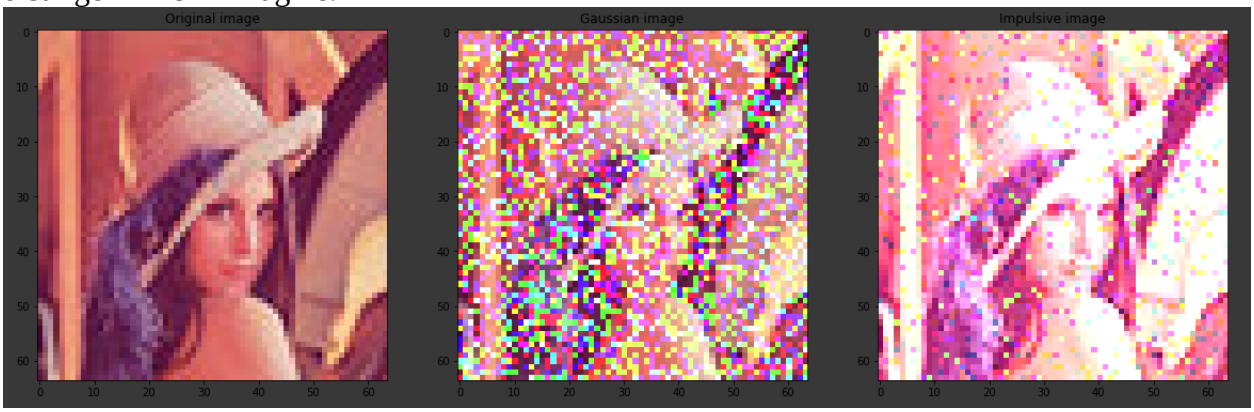

Figura 3.1: Imaginea Lena originală, zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

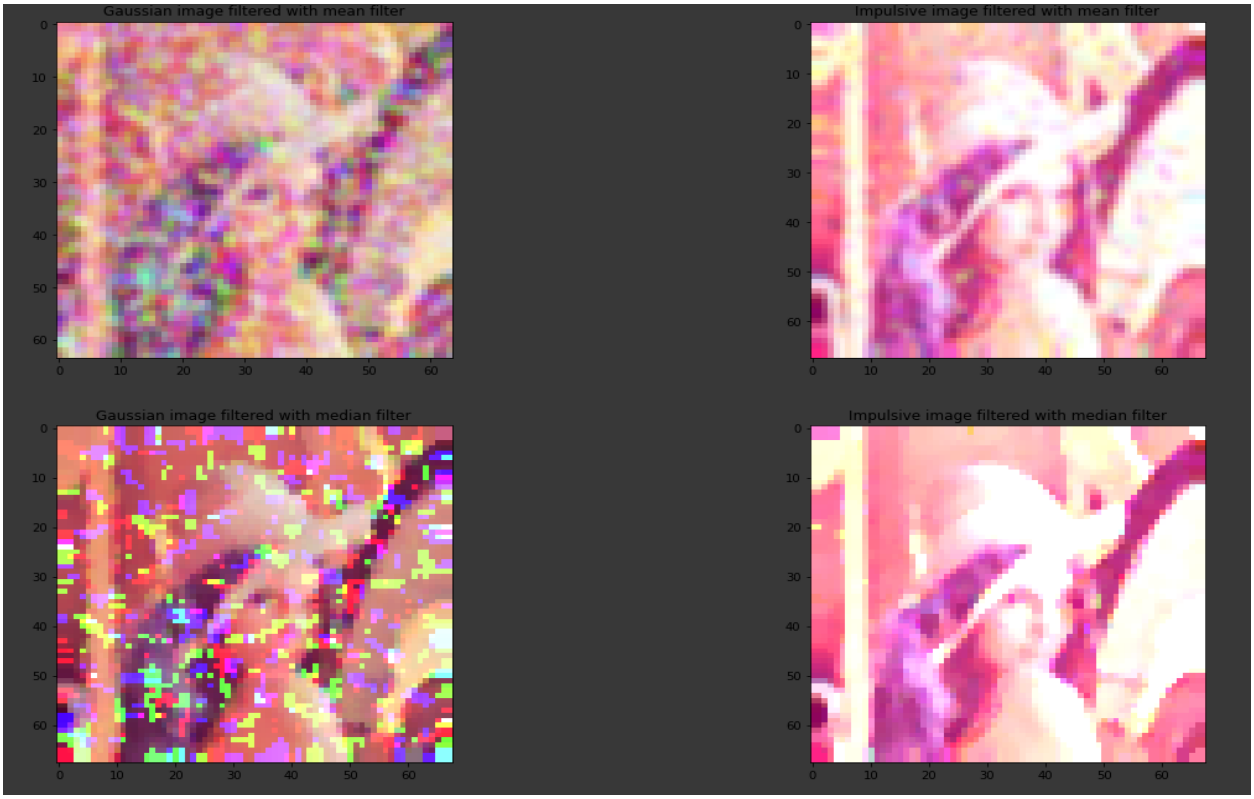

Figura 3.2: Imaginea Lena cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul de medie aritmetică și median

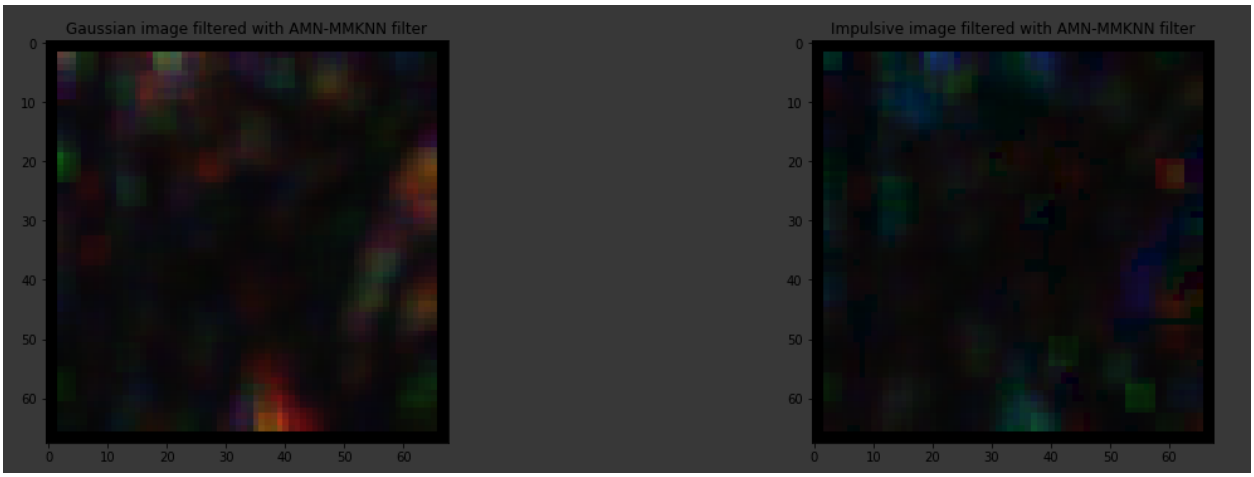

Figura 3.3: Imaginea Lena cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	PSNR	SSIM
Lena (img1)	medie aritmetică	gaussian	13.5924	0.4584
		impulsiv	9.7445	0.5505
	median	gaussian	13.2765	0.3847
		impulsiv	9.2749	0.5262
	amn-mmknn	gaussian	6.1591	0.0518
		impulsiv	6.1591	0.0518

Tabel 3.1: Imaginea Lena 64x64 cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

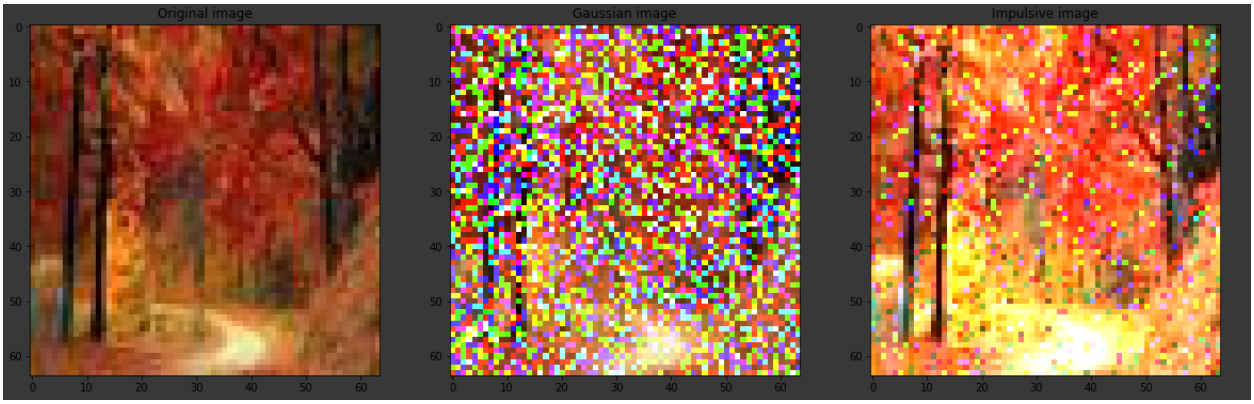

Figura 3.4: Imaginea Autumn originală, cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

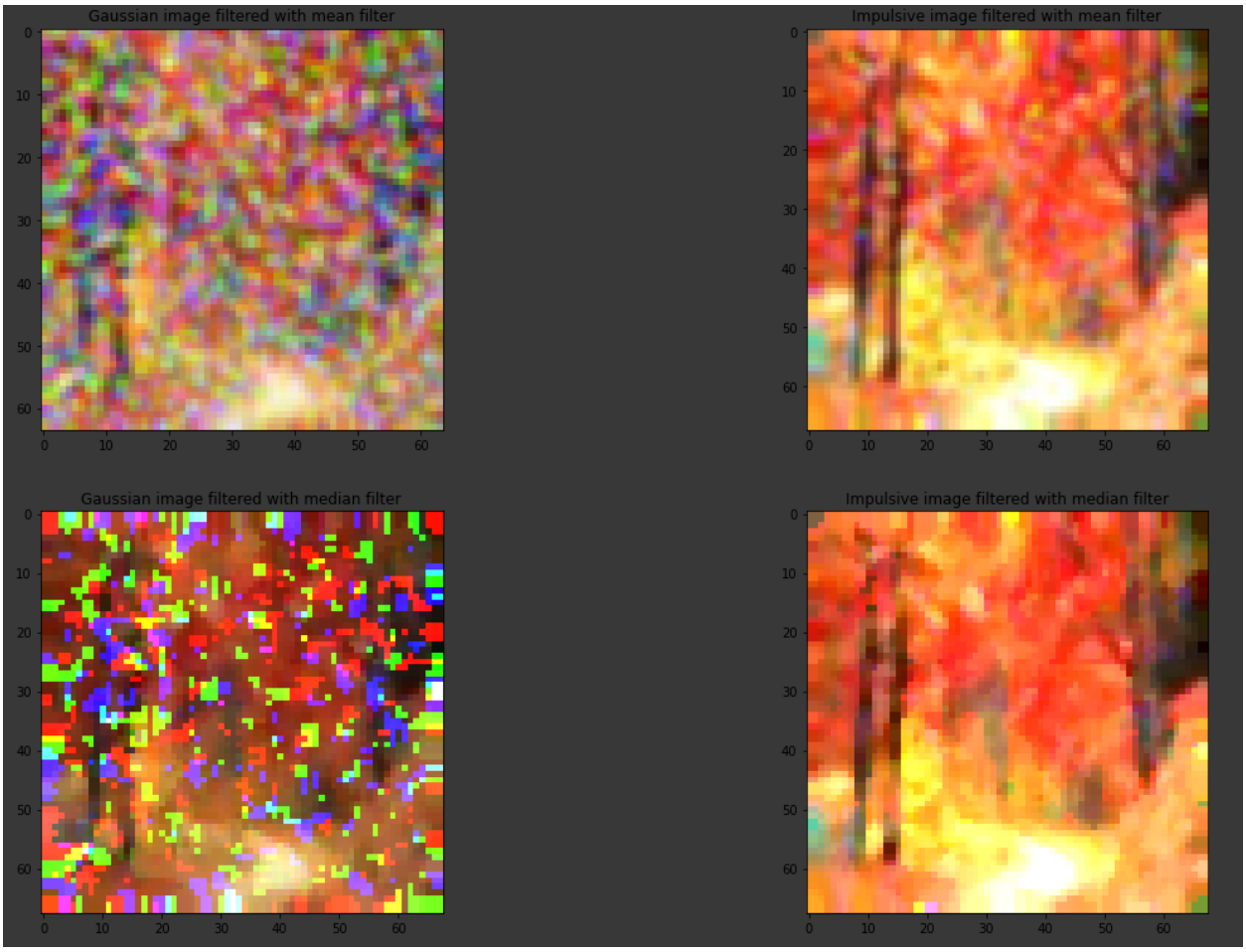

Figura 3.5: Imaginea Autumn cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul de medie aritmetică și median

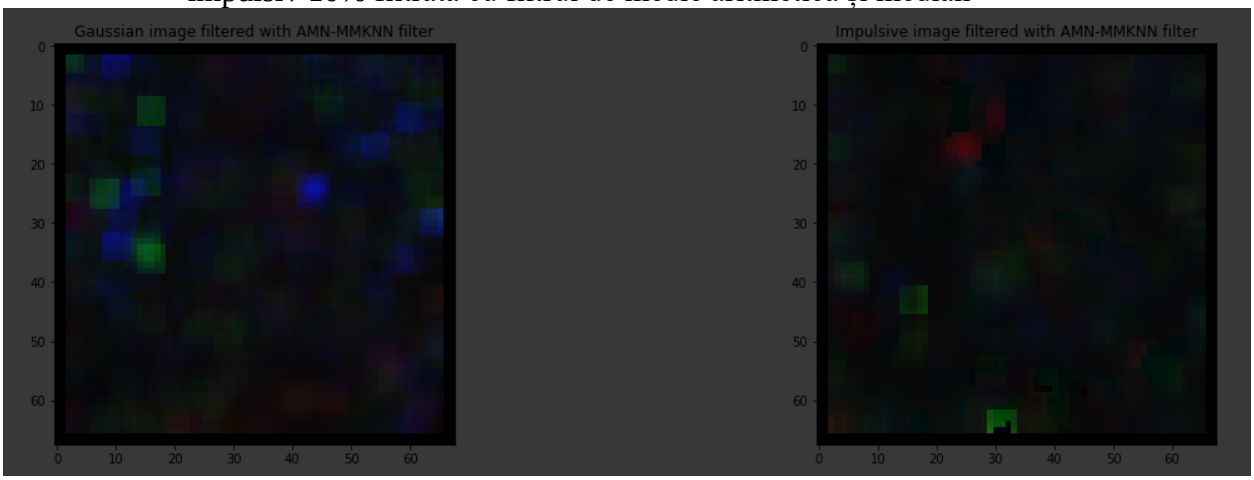

Figura 3.6: Imaginea Autumn cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	PSNR	SSIM
Autumn (img2)	medie aritmetică	gaussian	11.2779	0.2550
		impulsiv	11.1353	0.5212
	median	gaussian	11.3670	0.2523
		impulsiv	10.9515	0.5116
	amn-mmknn	gaussian	9.3831	0.0800
		impulsiv	9.3831	0.0800

Tabel 3.2: Imaginea Autumn 64x64 cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

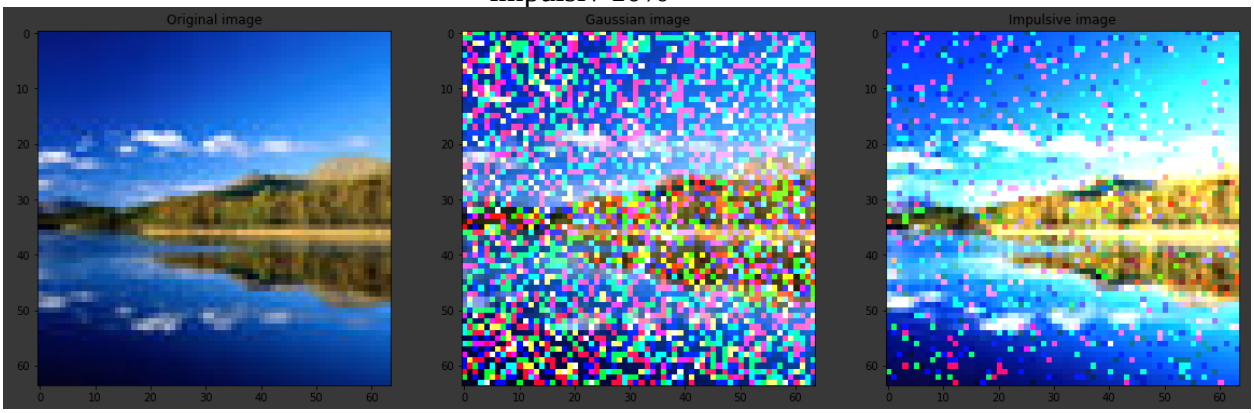

Figura 3.7: Imaginea Lake originală, cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

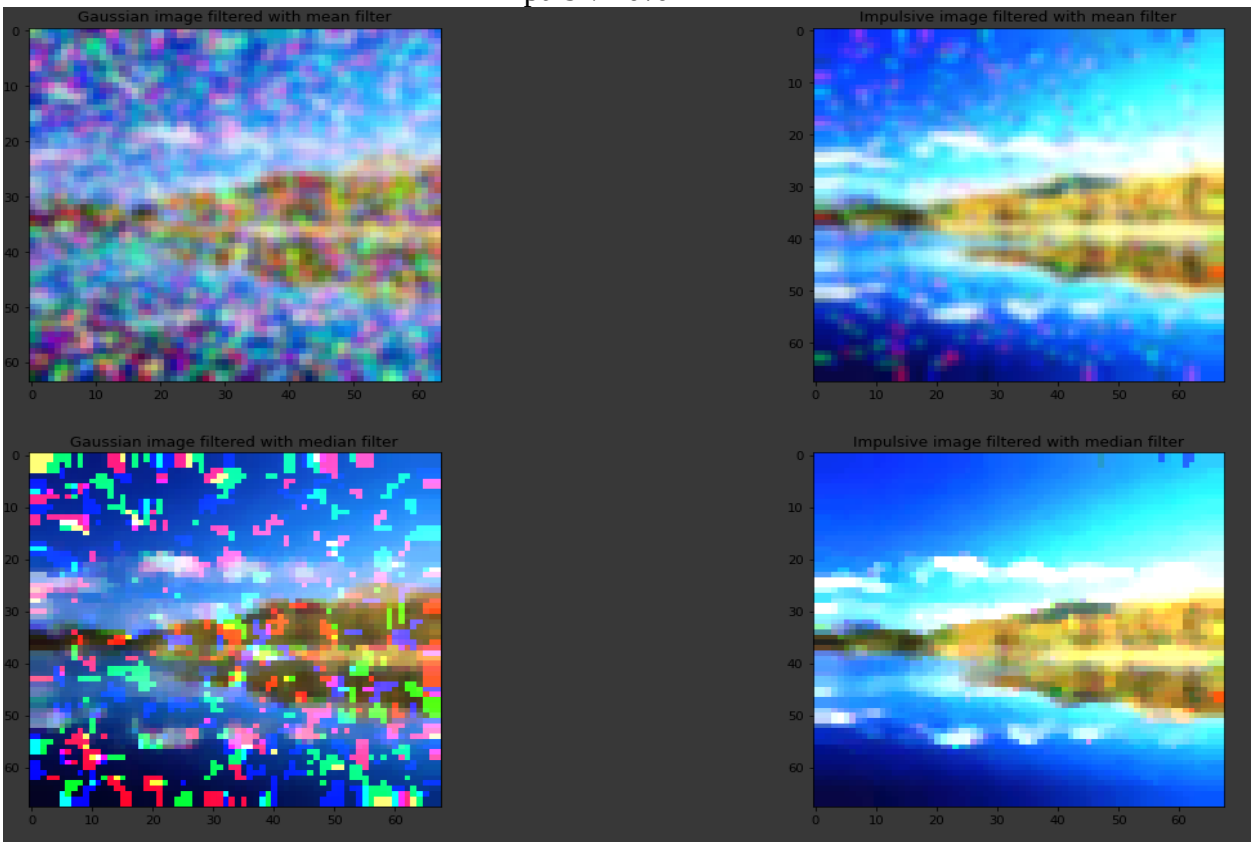

Figura 3.8: Imaginea Lake cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul de medie aritmetică și median

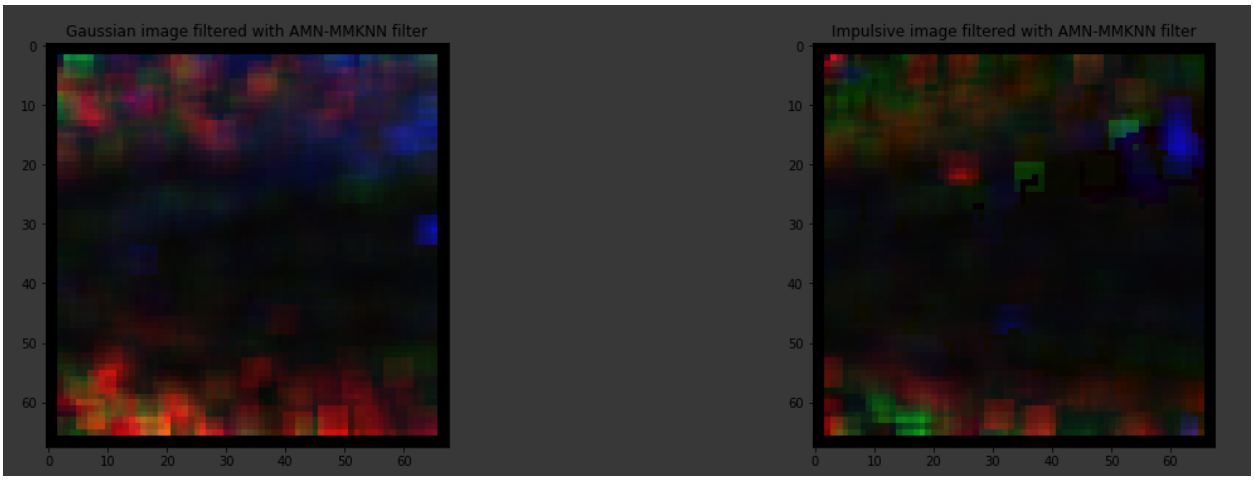

Figura 3.9: Imaginea Lake cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	PSNR	SSIM
Lake (img3)	medie aritmetică	gaussian	11.8910	0.3834
		impulsiv	11.1591	0.5324
	median	gaussian	11.9351	0.3742
		impulsiv	11.0679	0.6022
	amn-mmknn	gaussian	7.6839	0.0448
		impulsiv	7.6839	0.0448

Tabel 3.3: Imaginea Lake 64x64 cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

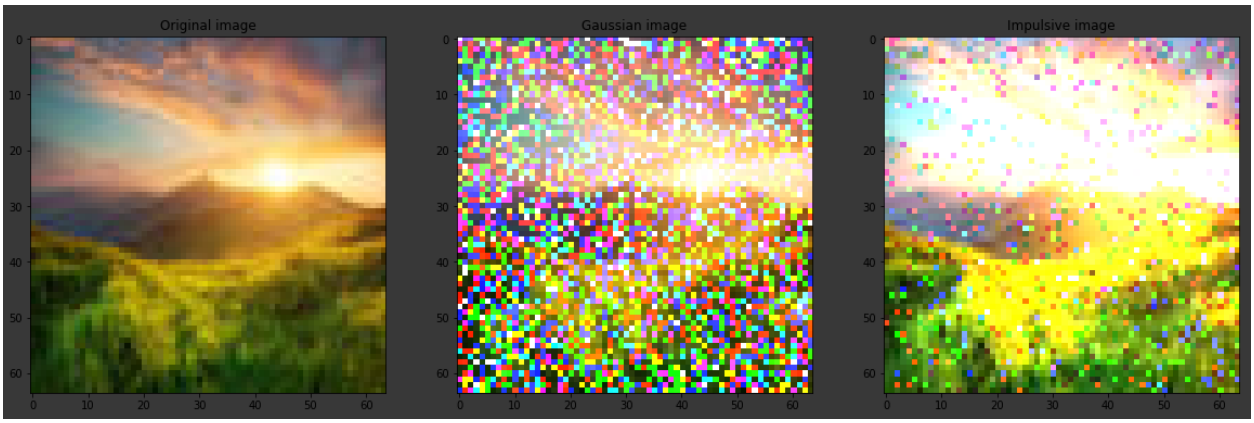

Figura 3.10: Imaginea Sunset originală, cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

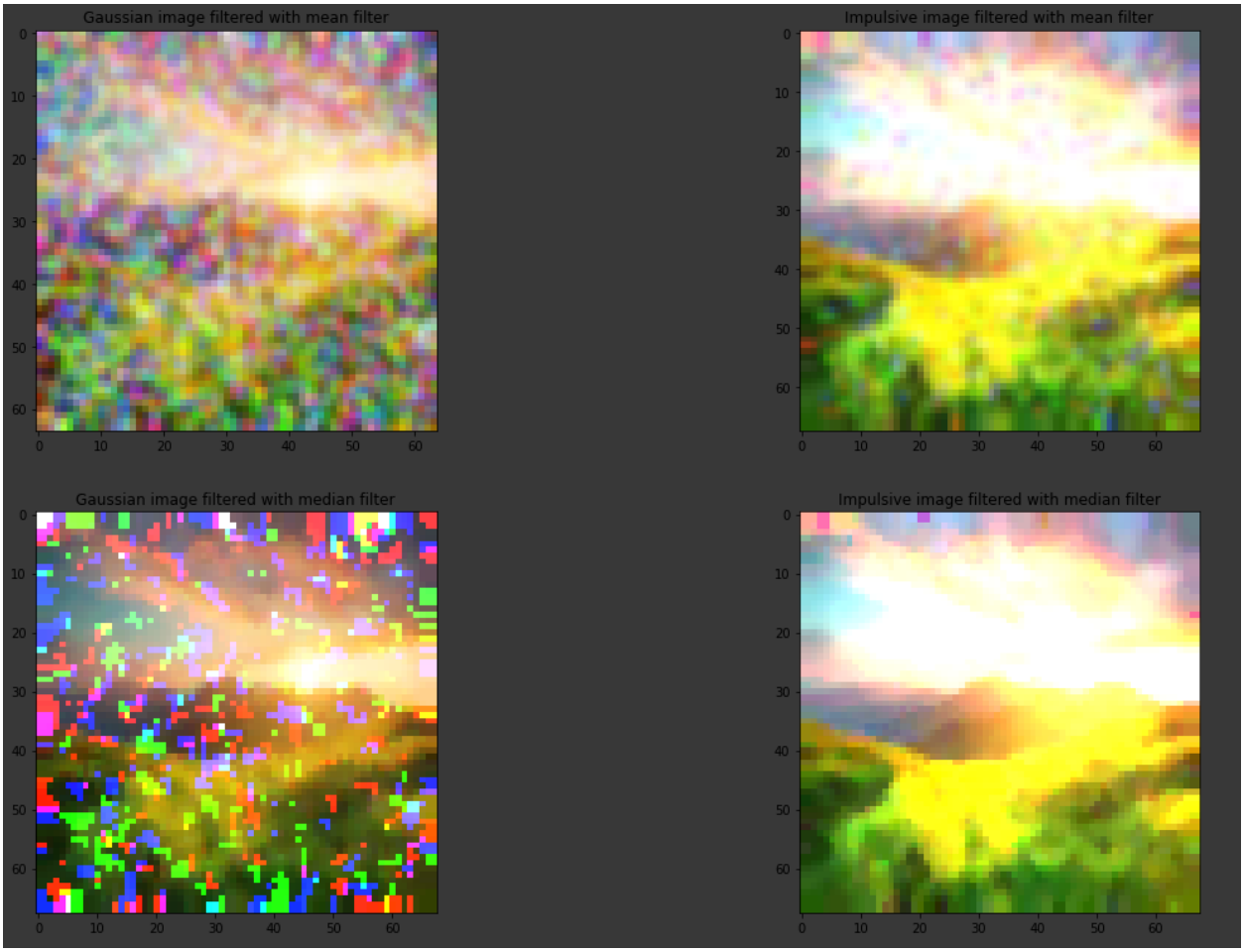

Figura 3.11: Imaginea Sunset cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul de medie aritmetică și median

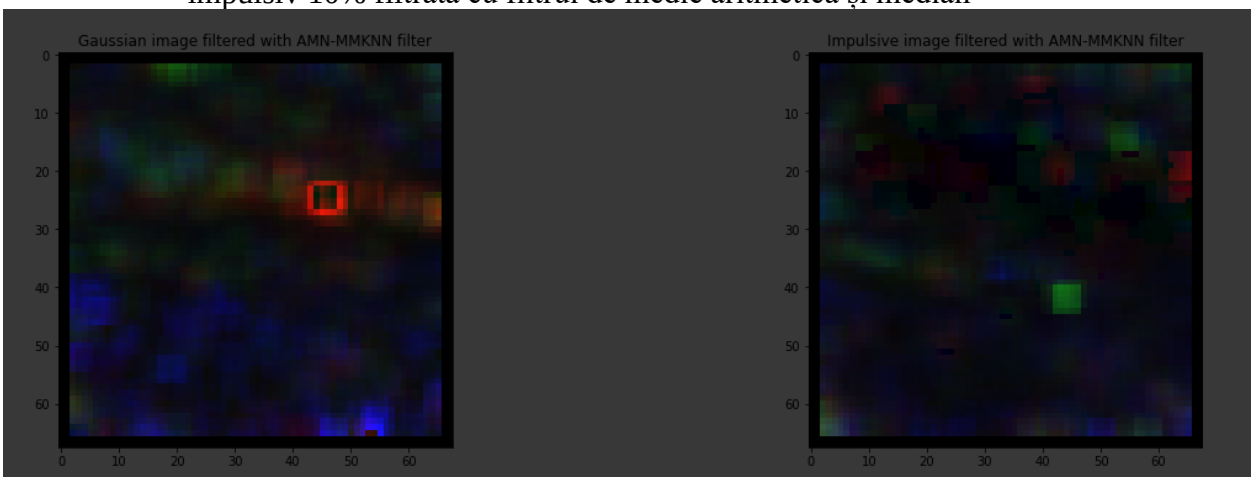

Figura 3.12: Imaginea Sunset cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	PSNR	SSIM
Sunset (img4)	medie aritmetică	gaussian	12.3172	0.3959
		impulsiv	10.3554	0.4919
	median	gaussian	12.4625	0.3929
		impulsiv	10.2029	0.5198
	amn-mmknn	gaussian	7.5264	0.0762
		impulsiv	7.5264	0.0762

Tabel 3.4: Imaginea Sunset 64x64 cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

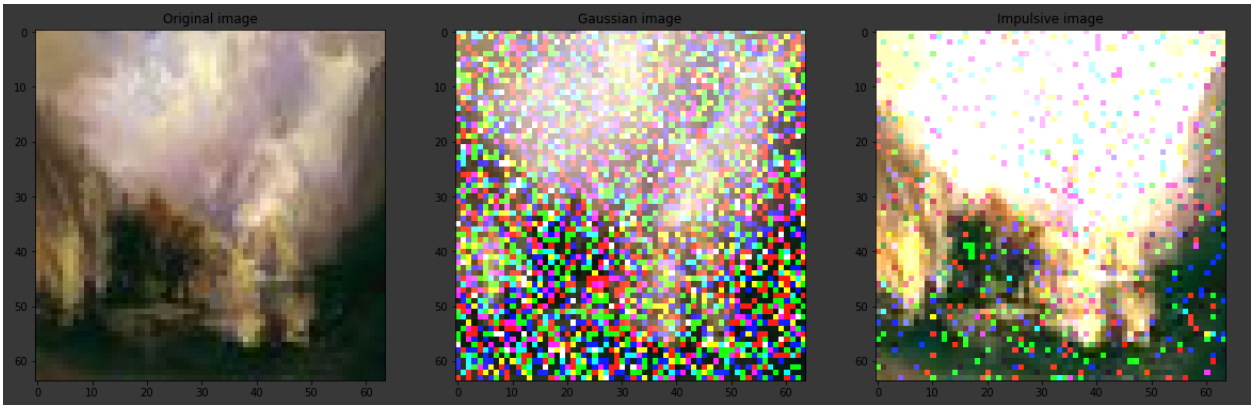

Figura 3.13: Imaginea Valley originală, cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

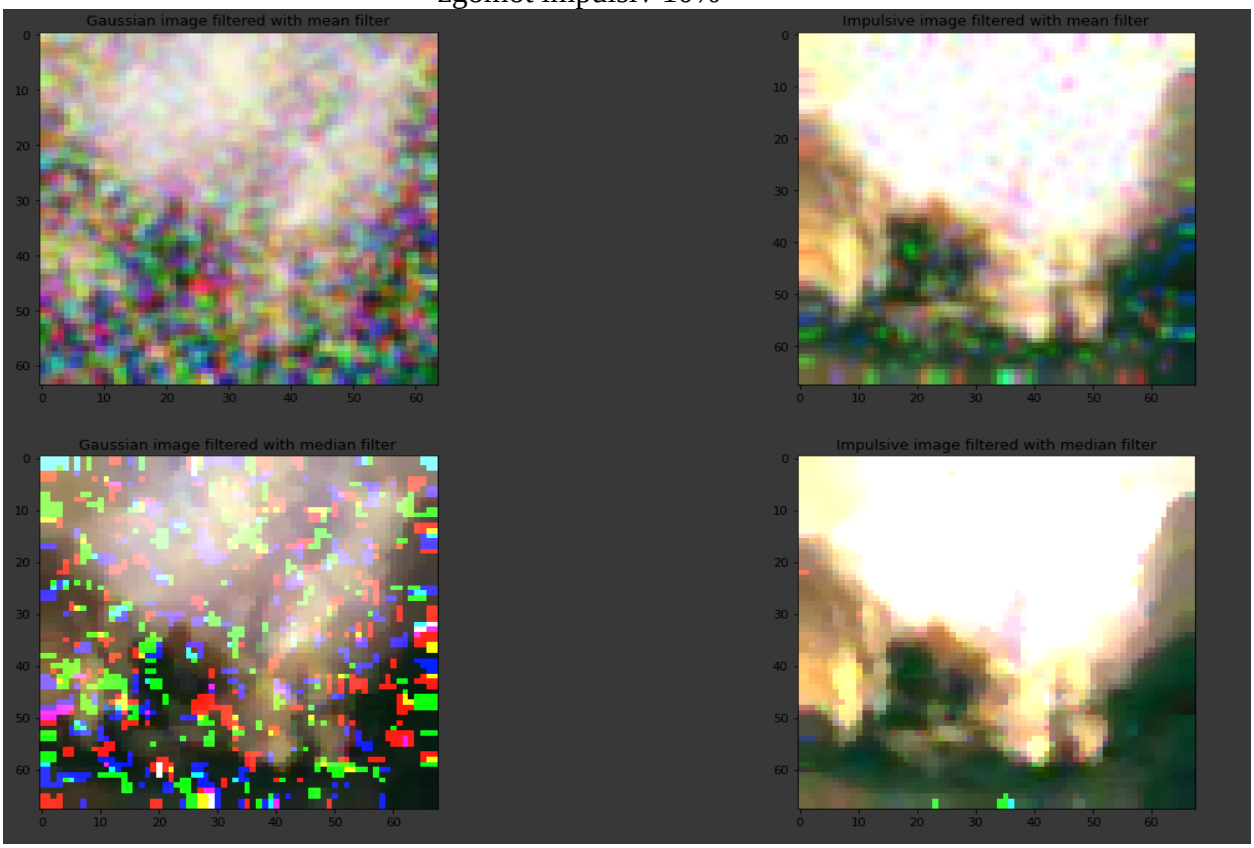

Figura 3.14: Imaginea Valley zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul de medie aritmetică și median

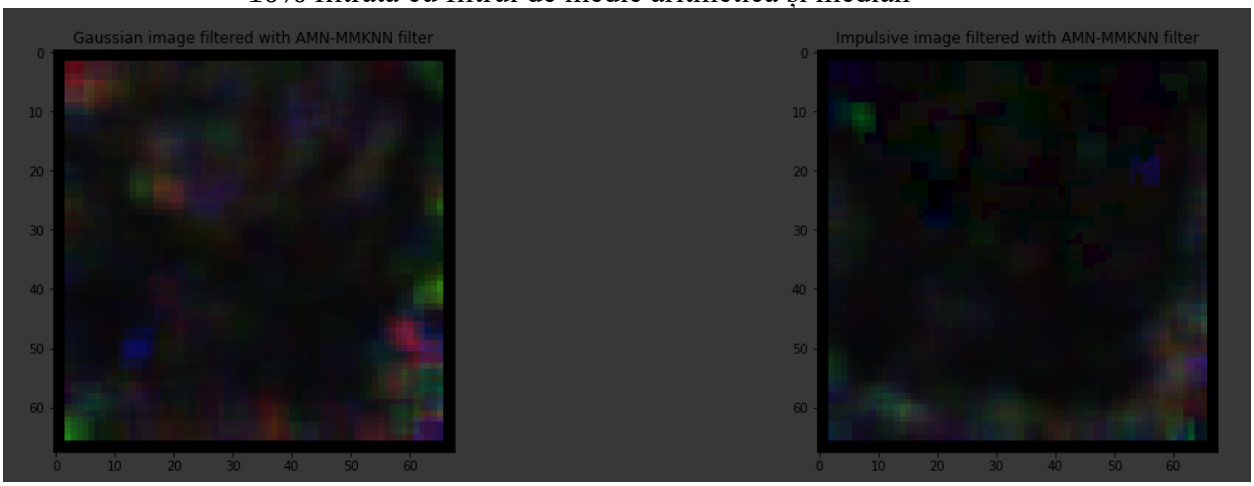

Figura 3.15: Imaginea Valley zgomot gaussian de deviație standard 2 și zgomot impulsiv 10% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	PSNR	SSIM
Valley (img5)	medie aritmetică	gaussian	12.5365	0.3820
		impulsiv	10.7307	0.4740
	median	gaussian	12.9310	0.3744
		impulsiv	10.6593	0.5144
	amn-mmknn	gaussian	7.5117	0.0782
		impulsiv	7.5117	0.0782

Tabel 3.5: Imaginea Valley 64x64 cu zgomot gaussian de deviație standard 2 și zgomot impulsiv 10%

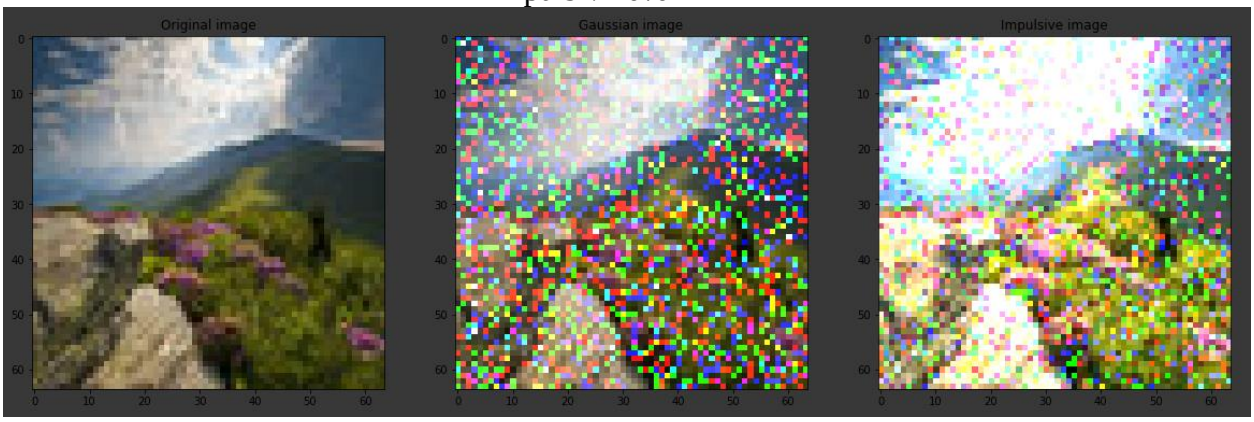

Figura 3.16: Imaginea Mountain originală, cu zgomot gaussian de medie 2 deviație standard 3 și zgomot impulsiv 20%

Figura 3.17: Imaginea Mountain cu zgomot gaussian de medie 2 deviație standard 3 și impulsiv 20% filtrată cu filtrul de medie aritmetică și median

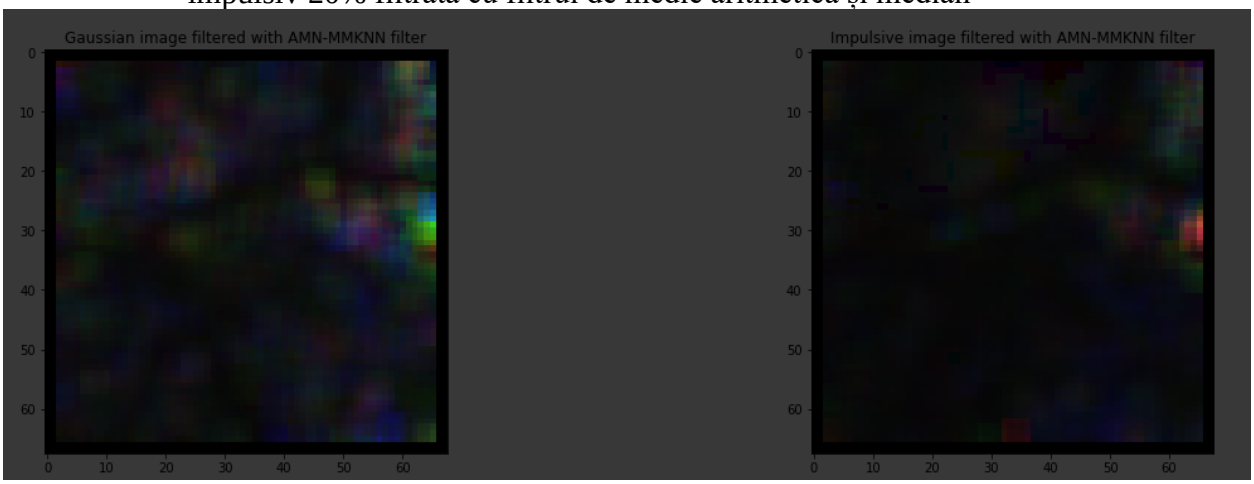

Figura 3.18: Imaginea Mountain cu zgomot gaussian de medie 2 deviație standard 3 și impulsiv 20% filtrată cu filtrul AMN-MMKNN

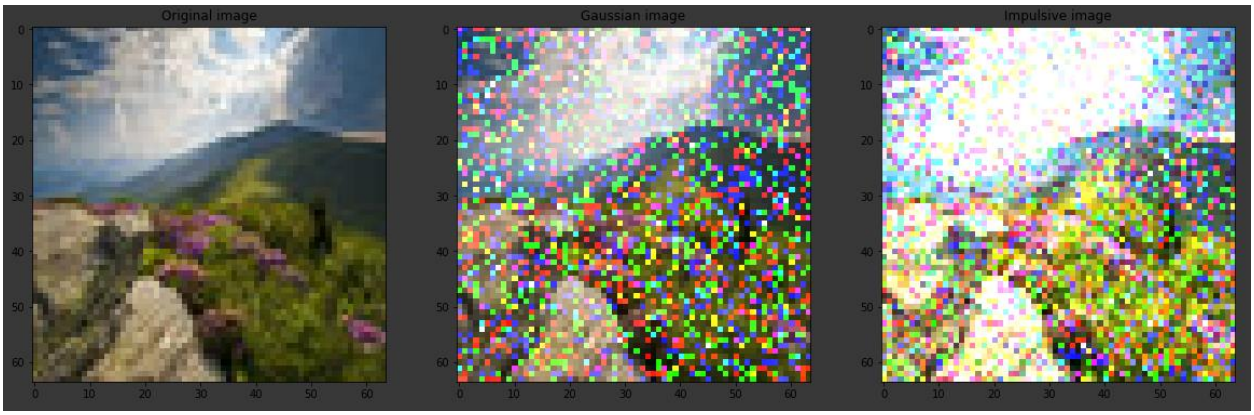

Figura 3.19: Imaginea Mountain originală, cu zgomot gaussian de medie 3 deviație standard 4 și zgomot impulsiv 30%

Figura 3.20: Imaginea Mountain cu zgomot gaussian de medie 3 deviație standard 4 și impulsiv 30% filtrată cu filtrul de medie aritmetică și median

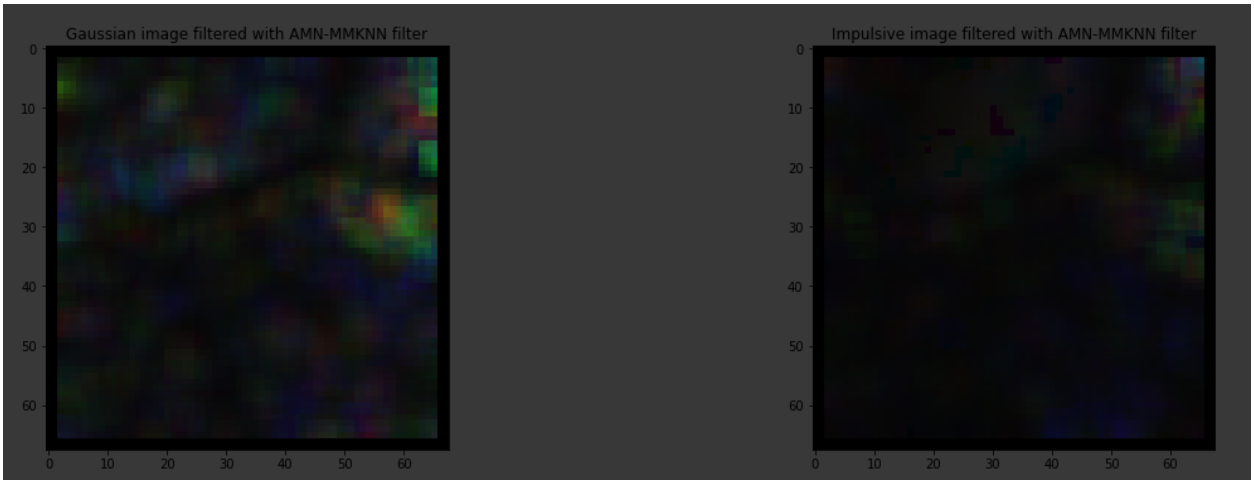

Figura 3.21: Imaginea Mountain cu zgomot gaussian de medie 3 deviație standard 4 și impulsiv 30% filtrată cu filtrul AMN-MMKNN

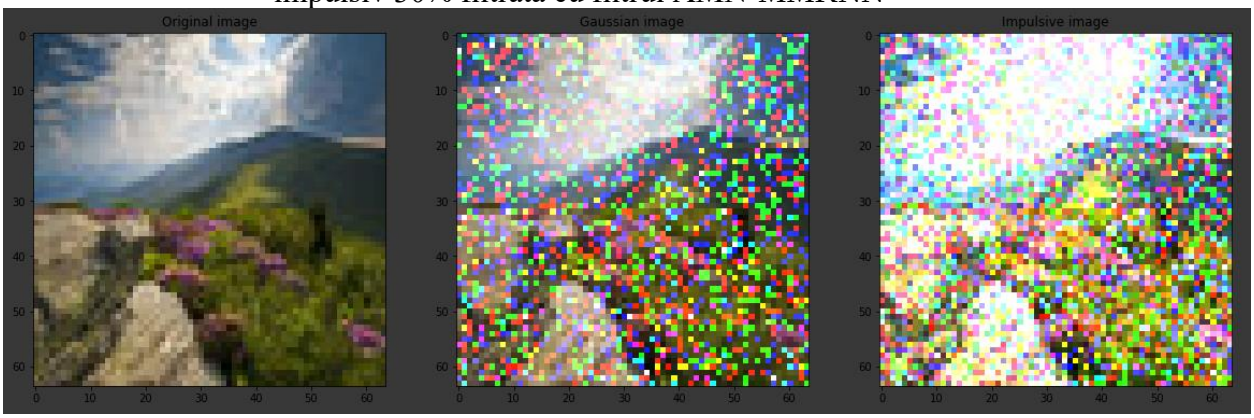

Figura 3.22: Imaginea Mountain originală, cu zgomot gaussian de medie 4 deviație standard 5 și zgomot impulsiv 40%

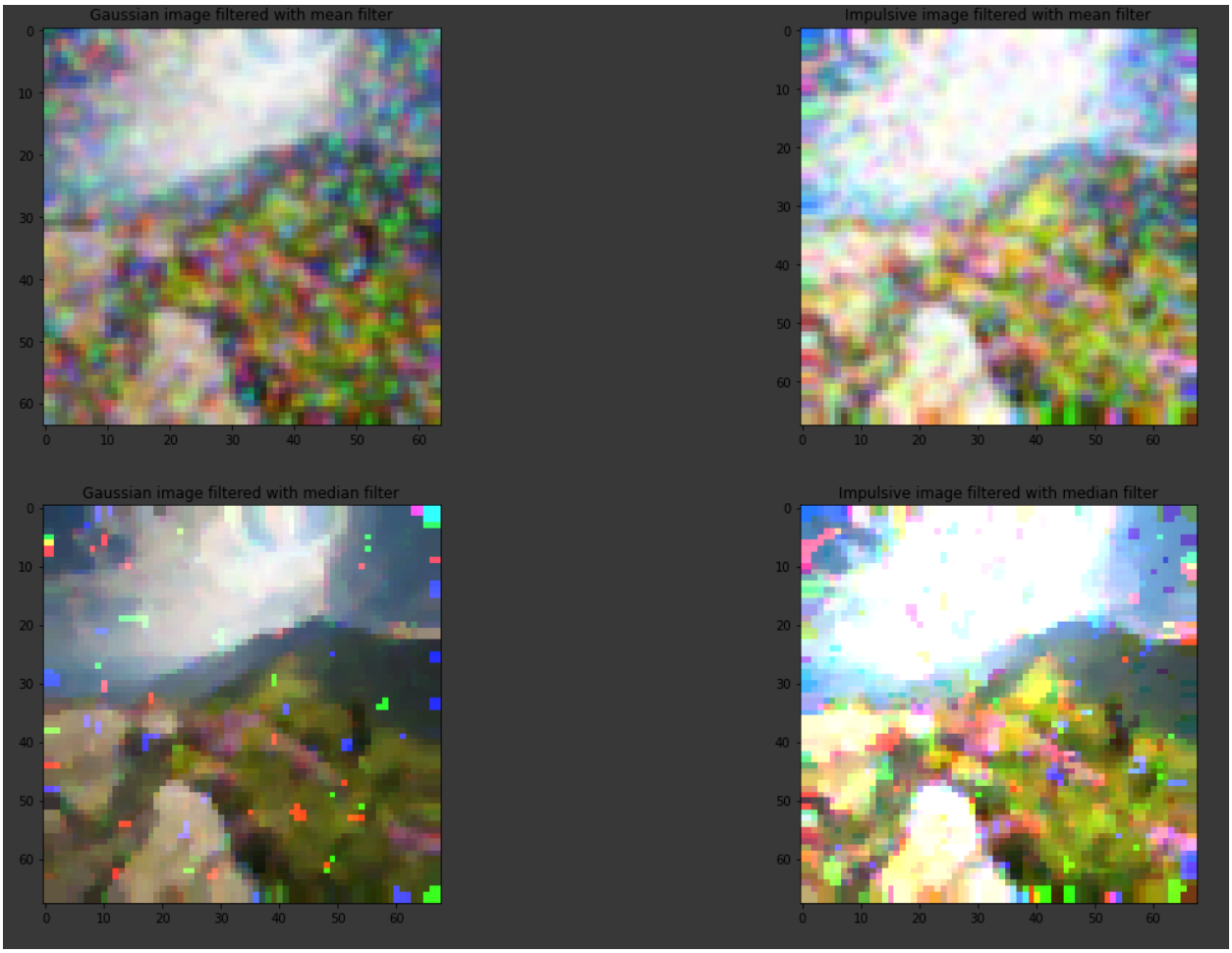

Figura 3.23: Imaginea Mountain cu zgomot gaussian de medie 4 deviație standard 5 și impulsiv 40% filtrată cu filtrul de medie aritmetică și median

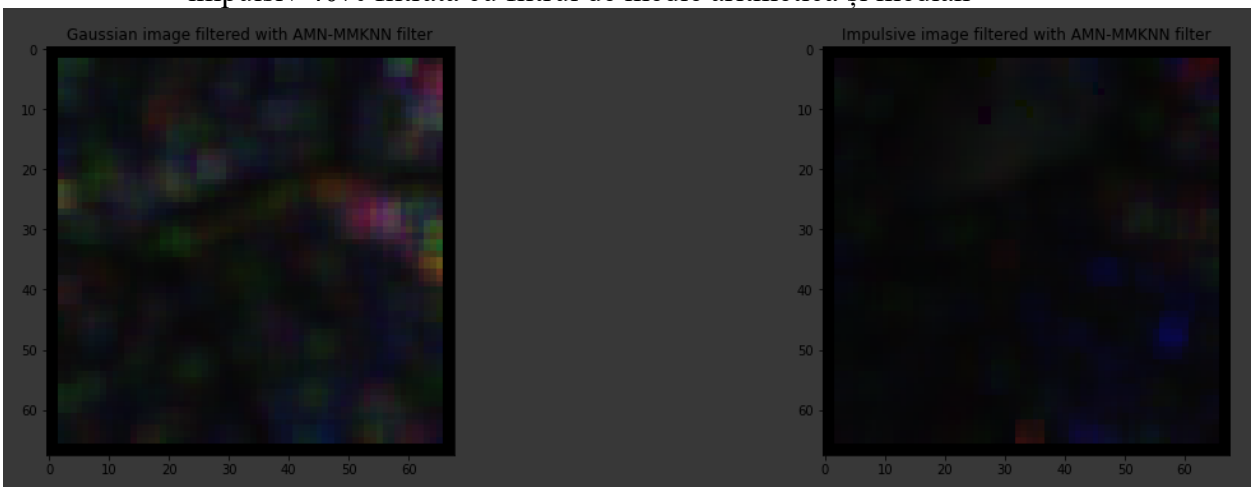

Figura 3.24: Imaginea Mountain cu zgomot gaussian de medie 4 deviație standard 5 și impulsiv 40% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	Intensitate zgomot	PSNR	SSIM
Mountain (img6)	medie aritmetică	gaussian	Medie = 2 Deviație standard = 3	16.2445	0.4861
			Medie = 3 Deviație standard = 4	15.9171	0.4805
			Medie = 4 Deviație standard = 5	15.9736	0.4814
		impulsiv	20%	10.2435	0.4810
			30%	10.0556	0.4477
			40%	10.0654	0.4184
	median	gaussian	Medie = 2 Deviație standard = 3	19.6655	0.6928
			Medie = 3 Deviație standard = 4	19.0231	0.6783
			Medie = 4 Deviație standard = 5	19.9910	0.6857
		impulsiv	20%	10.0309	0.5079
			30%	9.7535	0.4708
			40%	9.6331	0.4097
	amn-mmknn	gaussian	Medie = 2 Deviație standard = 3	7.8635	0.0621
			Medie = 3 Deviație standard = 4	7.8419	0.0618
			Medie = 4 Deviație standard = 5	7.8633	0.0626
		impulsiv	20%	7.8635	0.0621
			30%	7.8419	0.0618
			40%	7.8633	0.0626

Tabel 3.6: Imaginea Mountain 64x64 cu diferite intensități pentru zgomot gaussian și zgomot impulsiv

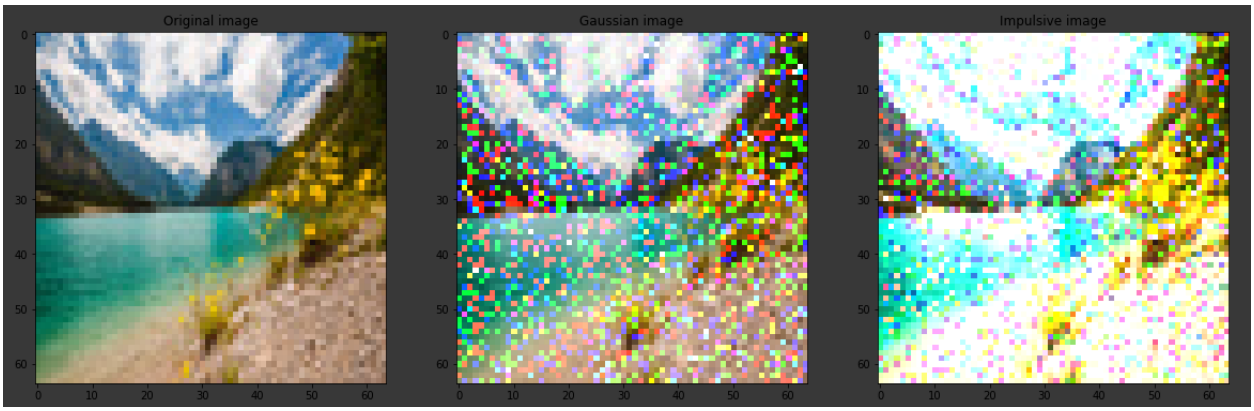

Figura 3.25: Imaginea Landscape originală, cu zgomot gaussian de medie 2 deviație standard 3 și zgomot impulsiv 20%

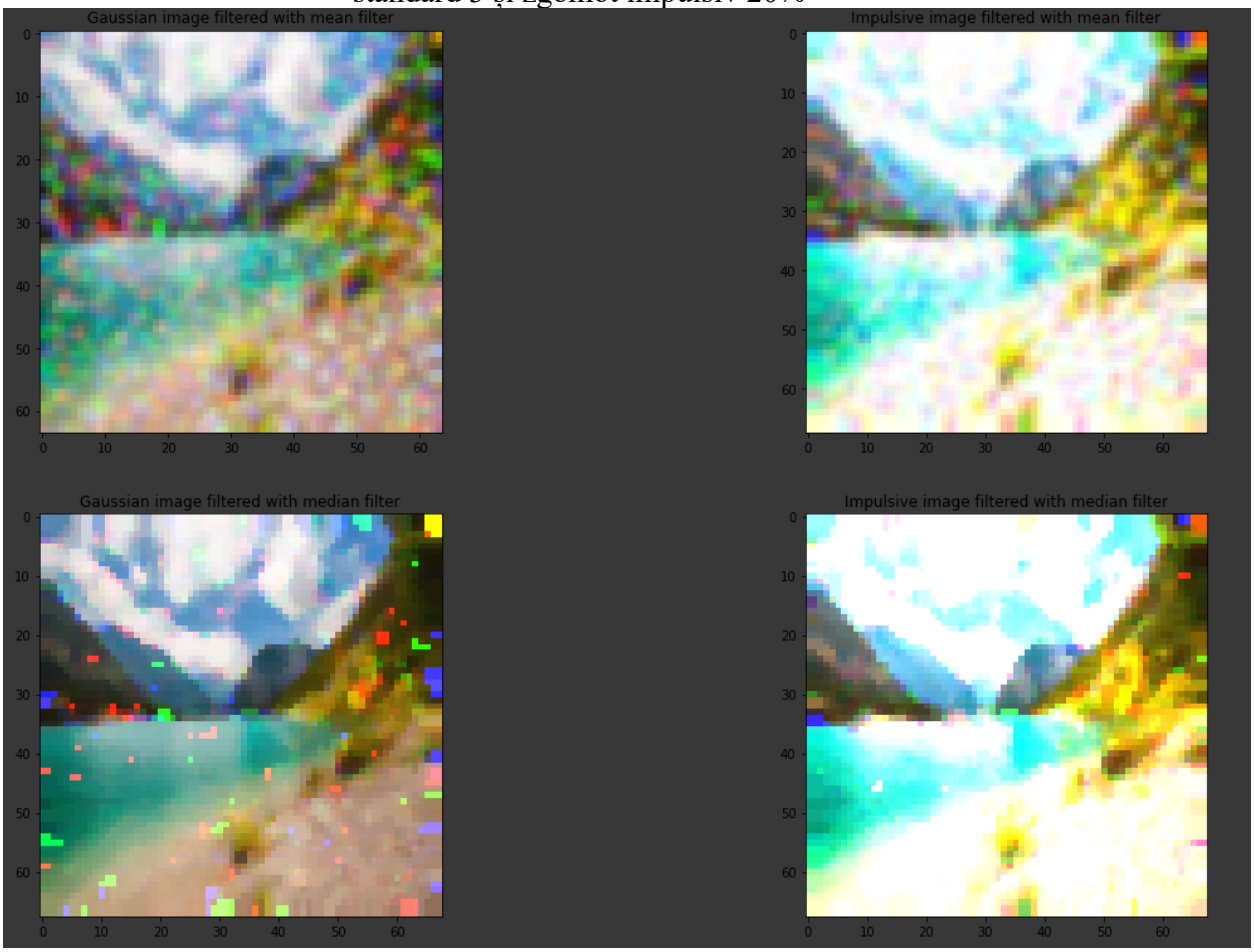

Figura 3.26: Imaginea Landscape cu zgomot gaussian de medie 2 deviație standard 3 și impulsiv 20% filtrată cu filtrul de medie aritmetică și median

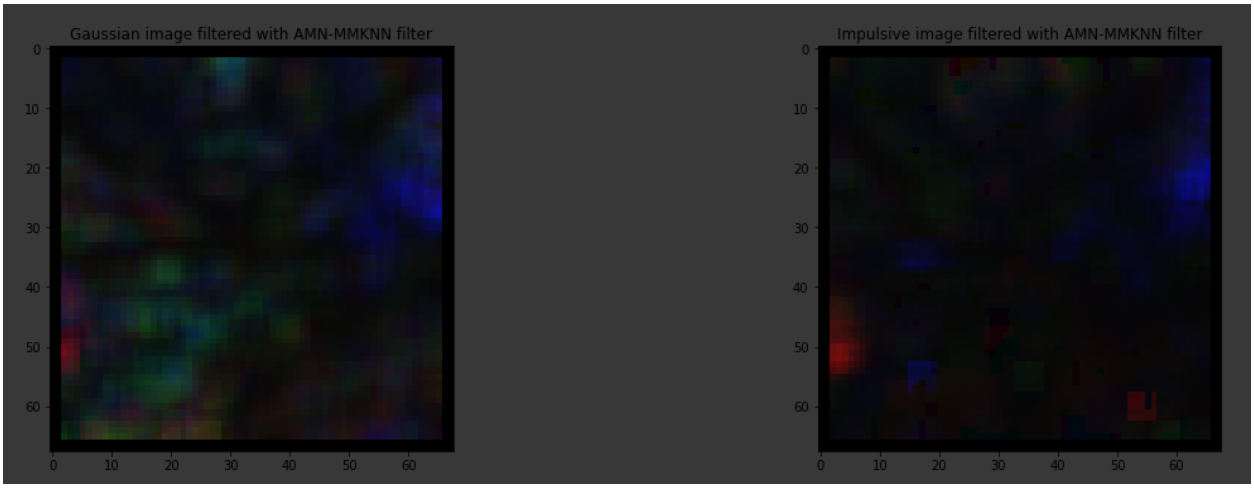

Figura 3.27: Imaginea Landscape cu zgomot gaussian de medie 2 deviație standard 3 și impulsiv 20% filtrată cu filtrul AMN-MMKNN

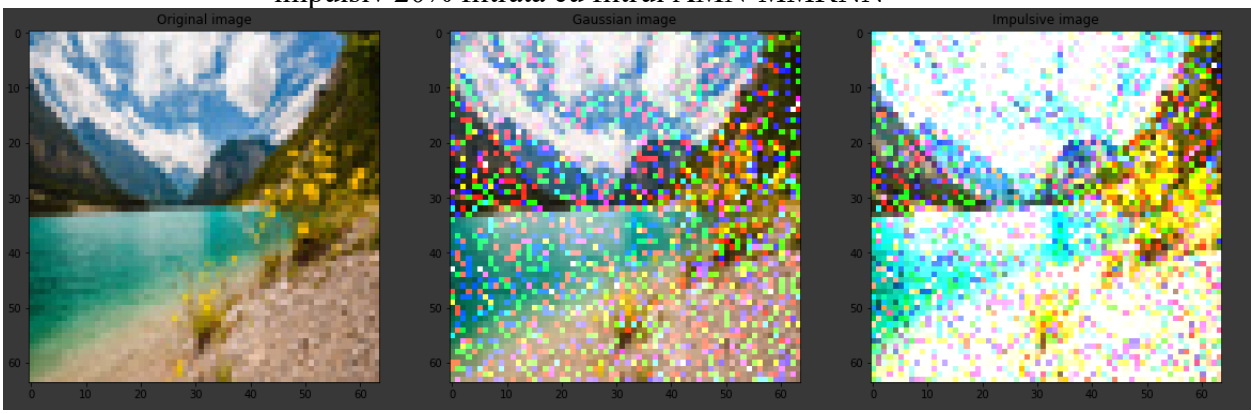

Figura 3.28: Imaginea Landscape originală, cu zgomot gaussian de medie 3 deviație standard 4 și zgomot impulsiv 30%

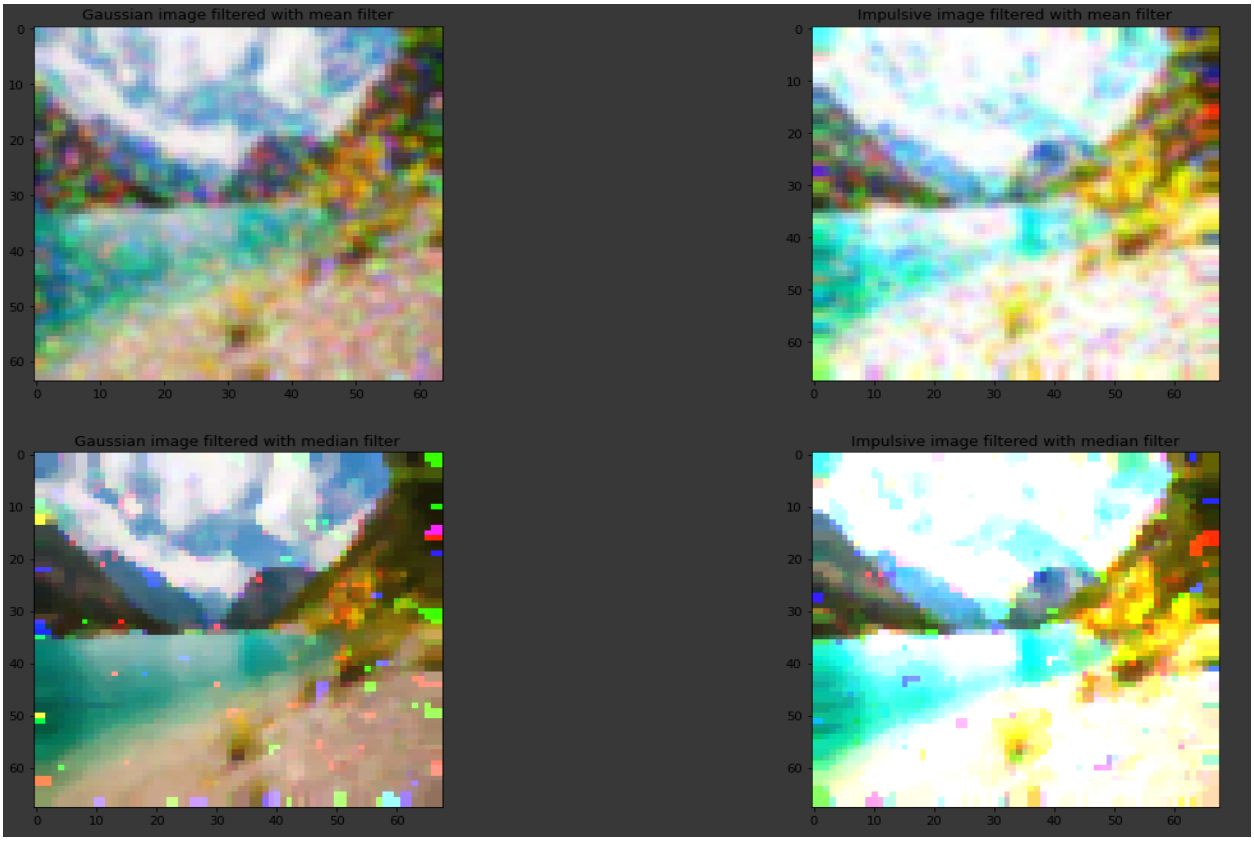

Figura 3.29: Imaginea Landscape cu zgomot gaussian de medie 3 deviație standard 4 și impulsiv 30% filtrată cu filtrul de medie aritmetică și median

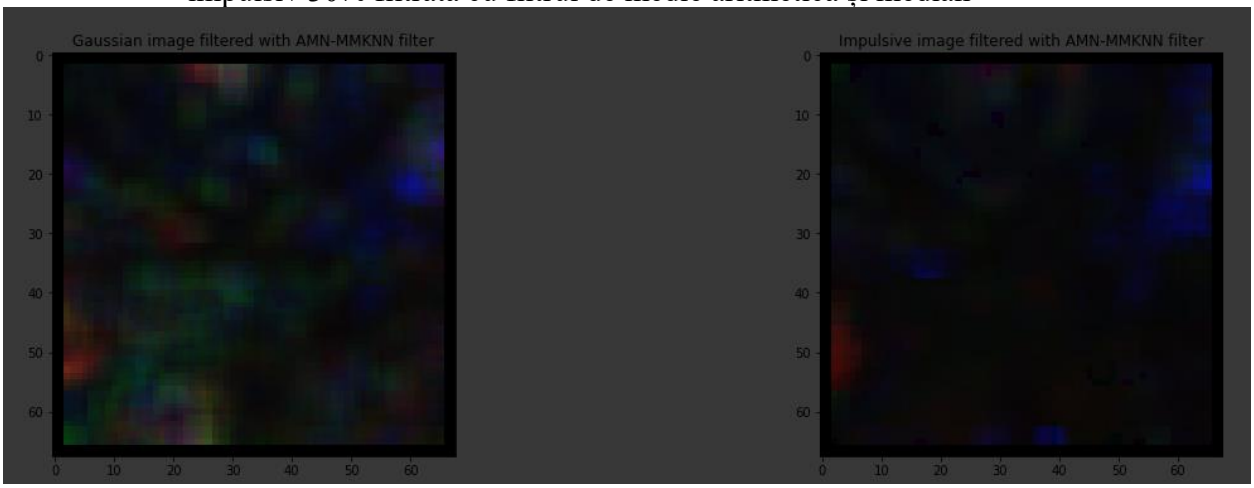

Figura 3.30: Imaginea Landscape cu zgomot gaussian de medie 3 deviație standard 4 și impulsiv 30% filtrată cu filtrul AMN-MMKNN

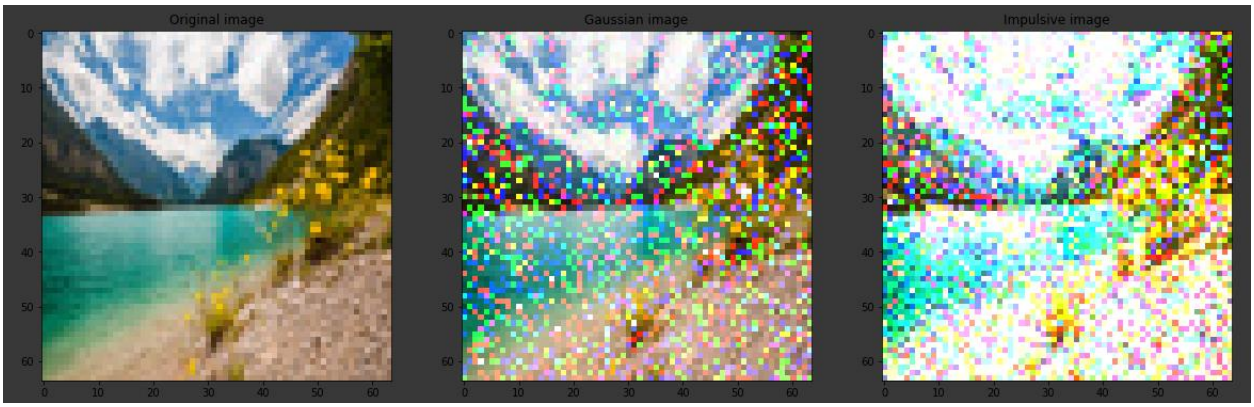

Figura 3.31: Imaginea Landscape originală, cu zgomot gaussian de medie 4 deviație standard 5 și zgomot impulsiv 40%

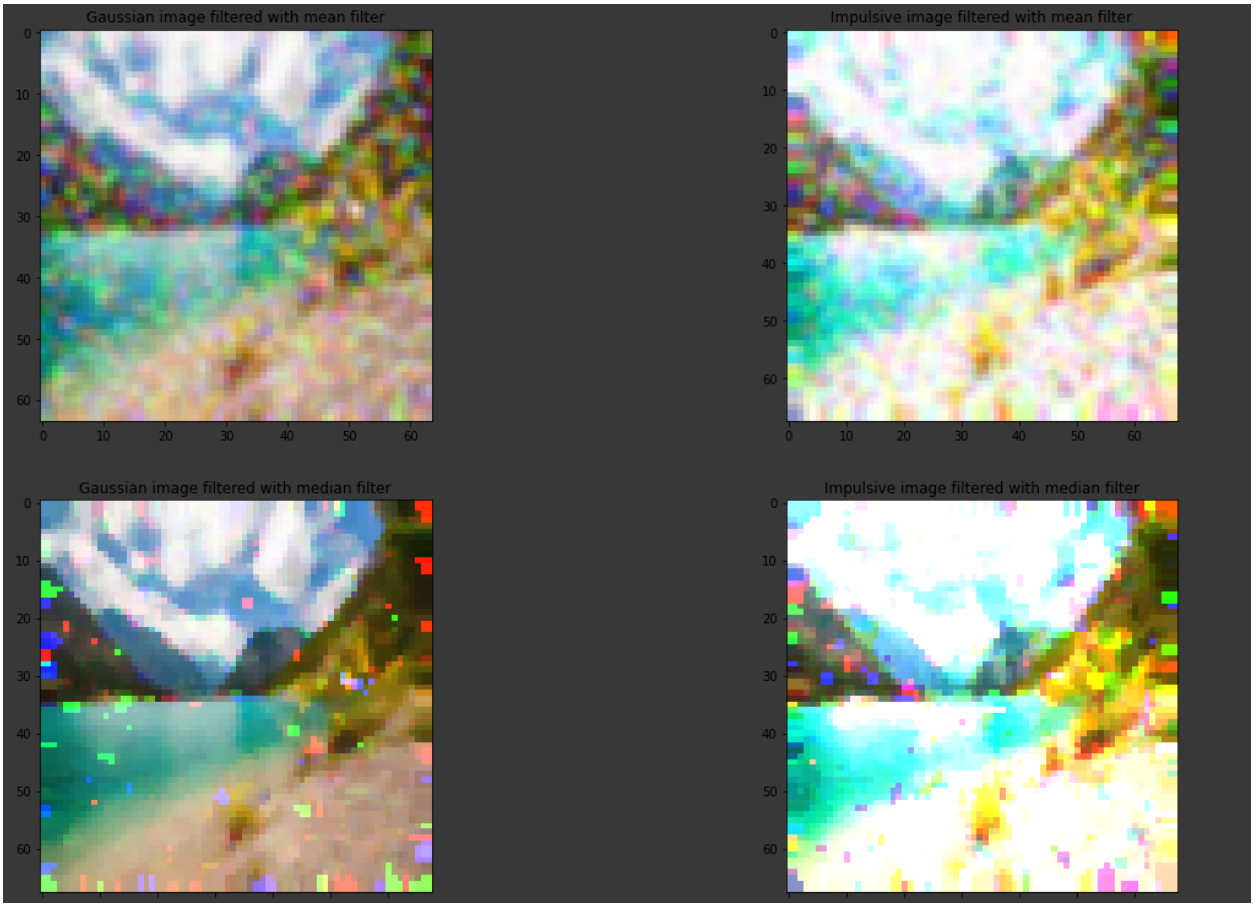

Figura 3.32: Imaginea Landscape cu zgomot gaussian de medie 4 deviație standard 5 și impulsiv 40% filtrată cu filtrul de medie aritmetică și median

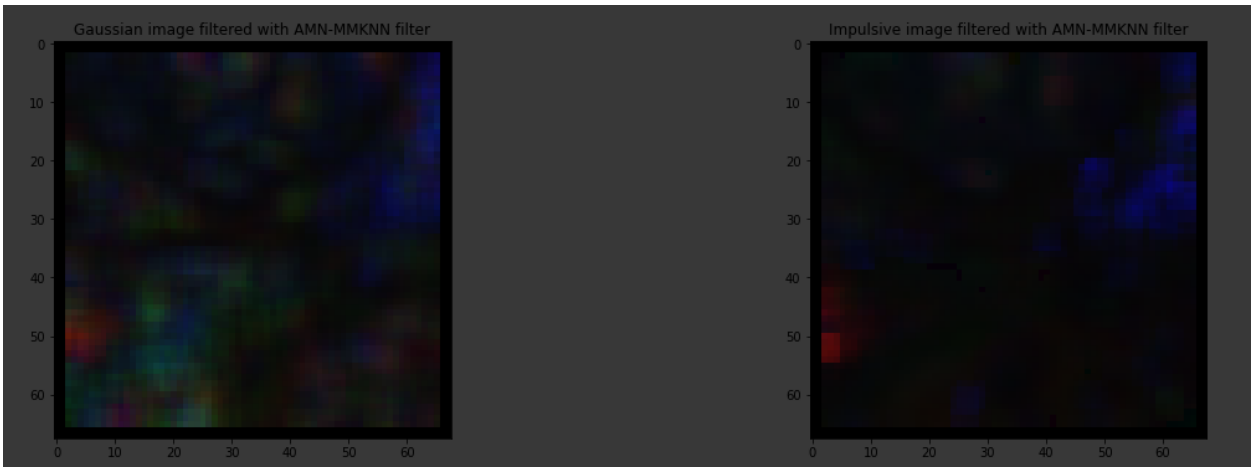

Figura 3.33: Imaginea Landscape cu zgomot gaussian de medie 4 deviație standard 5 și impulsiv 40% filtrată cu filtrul AMN-MMKNN

Imagine	Filtru	Zgomot	Intensitate zgomot	PSNR	SSIM
Landscape (img7)	medie aritmetică	gaussian	Medie = 2 Deviație standard = 3	17.0075	0.5892
			Medie = 3 Deviație standard = 4	17.0641	0.5981
			Medie = 4 Deviație standard = 5	16.579	0.5853
		impulsiv	20%	10.1140	0.4621
			30%	10.3461	0.4382
			40%	10.4366	0.4285
	median	gaussian	Medie = 2 Deviație standard = 3	19.2926	0.6985
			Medie = 3 Deviație standard = 4	19.1205	0.7077
			Medie = 4 Deviație standard = 5	18.6278	0.6891
		impulsiv	20%	9.4955	0.4432
			30%	9.4689	0.4082
			40%	9.4336	0.3846

	amn-mmknn	gaussian	Medie = 2 Deviație standard = 3	5.9072	0.0360
			Medie = 3 Deviație standard = 4	5.9058	0.0393
			Medie = 4 Deviație standard = 5	5.8774	0.0403
		impulsiv	20%	5.9072	0.0360
			30%	5.9058	0.0393
			40%	5.8774	0.0403

Tabel 3.7: Imaginea Landscape 64x64 cu diferite intensități pentru zgomot gaussian și zgomot impulsiv

Din figurile afișate se pot vedea următoarele:

- calitatea imaginii originale este afectată datorită utilizării dimensiunii 64x64, dimensiune care este una destul de mică și care nu poate păstra detaliile la un nivel foarte bun
- zgomotul gaussian afectează foarte mult imaginea originală
- zgomotul impulsiv afectează într-o măsură mai mică imaginea originală
- prima dată se folosește filtrul de medie aritmetică(mediere) care este un filtru de netezire(adică efectuează o filtrare liniară), prin urmare are un efect de încețoșare a imaginii iar acest lucru se observă pe ambele imagini afectate de zgomotul gaussian și impulsiv. De asemenea, în general pentru reducerea zgomotului gaussian se folosește filtrul de mediere, iar pentru zgomotul impulsiv se folosește filtrul median, doar că în cazul de față din cauza dimensiunii mici a imaginilor, zgomotul impulsiv a fost curățat mai bine decât zgomotul gaussian.
- după se aplică filtrul median(care efectuează o filtrare neliniară) pe aceleași imagini afectate de zgomotul gaussian și impulsiv. Putem spune că efectele filtrării sunt evidente în sensul că pe imaginea cu zgomot gaussian încă se observă artefacte care nu au fost curățate complet, în schimb la cealaltă imagine zgomotul impulsiv a fost eliminat aproape complet. Se mai observă totuși existența unor puncte de zgomot care nu au putut fi eliminate, iar în acest caz putem spune că filtrul a fost străpuns de impulsuri.
- la final se aplică filtrul implementat pe baza articolului dar din nefericire efectele sale asupra imaginilor este unul puternic în sensul că imaginile sunt aproape în întregime întunecate, și nu se poate face o distincție clară.

Înainte de a trece la analiza tabelelor trebuie menționat faptul că în general PSNR-ul (Peak signal-to-noise ratio) indică calitatea imaginii în funcție de valoarea sa. Mai exact ce se află sub 20 dB înseamnă o imagine cu calitate slabă spre foarte slabă, ce se află în intervalul [20,30] dB înseamnă o calitate acceptabilă a imaginii iar ce se află în intervalul [30,40] dB sau peste 40 dB arată o calitate foarte bună a imaginii. Pe de altă parte SSIM(Structural Similarity Index Measure) este folosit să arate cât de

asemănătoare sunt 2 imagini între ele. Intervalul de valori utilizat este [0,1], unde 1=imaginile sunt perfect identice, 0=imaginile nu seamănă deloc.[9]

Din tabelele afișate se pot vedea următoarele:

- pentru tabelul 3.1 cea mai bună măsurătoare pentru PSNR(13.5924) se găsește la imaginea care a fost afectată de zgomotul gaussian și filtrată folosind filtrul de medie aritmetică, lucru de așteptat din moment ce filtrul de medie aritmetică este construit special pentru eliminarea acestui tip de zgomot. Cea mai bună performanță pentru SSIM(0.5505) se găsește la imaginea care a fost afectată de zgomotul gaussian și filtrată folosind filtrul median, ceea ce în mod normal nu se întâmplă deoarece filtrul median este construit special pentru eliminarea zgomotului impulsiv. Cele mai slabe performanțe pentru PSNR(6.1591) și SSIM(0.0518) se obțin pentru filtrul AMN-MMKNN implementat atât pentru zgomotul gaussian cât și pentru cel impulsiv.
- pentru tabelul 3.2 cea mai bună performanță pentru PSNR(11.3670) este la imaginea afectată de zgomotul gaussian și filtrată median, lucru neașteptat deoarece filtrul median este construit special pentru eliminarea zgomotului impulsiv. Pentru SSIM(0.5212) cel mai bun rezultat se obține la imaginea afectată de zgomotul impulsiv și filtrată cu filtrul de mediere, lucru opus față de ce se întâmplă normal deoarece filtrul de mediere este construit pentru eliminarea zgomotului gaussian. Cele mai slabe performanțe pentru PSNR(9.3831) și SSIM(0.0800) se obțin pentru filtrul AMN-MMKNN implementat atât pentru zgomotul gaussian cât și pentru cel impulsiv.
- pentru tabelul 3.3 cea mai bună performanță pentru PSNR(11.9351) este la imaginea afectată de zgomotul gaussian și filtrată median, lucru neașteptat deoarece filtrul median este construit special pentru eliminarea zgomotului impulsiv. Pentru SSIM(0.6022) cel mai bun rezultat se obține la imaginea afectată de zgomotul impulsiv și filtrată median, ceea ce se aștepta să se întâmple. Cele mai slabe performanțe pentru PSNR(7.6839) și SSIM(0.0448) se obțin pentru filtrul AMN-MMKNN implementat atât pentru zgomotul gaussian cât și pentru cel impulsiv.
- pentru tabelul 3.4 comportamentul este același ca la tabelul 3.3 atât pentru PSNR(12.4625) cât și pentru SSIM(0.5198) în ceea ce privește performanțele cele mai bune. Cele mai slabe performanțe pentru PSNR(7.5264) și SSIM(0.0762) se obțin pentru filtrul AMN-MMKNN implementat atât pentru zgomotul gaussian cât și pentru cel impulsiv.
- pentru tabelul 3.5 se întâmplă același lucru ca și tabelul 3.3 atât pentru PSNR(12.9310) cât și pentru SSIM(0.3744) în ceea ce privește performanțele cele mai bune. Cele mai slabe performanțe pentru PSNR(7.5117) și SSIM(0.0782) se obțin pentru filtrul AMN-MMKNN implementat atât pentru zgomotul gaussian cât și pentru cel impulsiv.
- pentru tabelul 3.6 dintre cele 3 intensități pentru zgomotul gaussian filtrat de filtrul de medie aritmetică, pentru cel de medie 2 și deviație standard 3 se obțin performanțele cele mai bune atât pentru PSNR(16.2445) cât și pentru SSIM(0.4861), lucru dorit deoarece este cea mai mică intensitate dintre cele 3 testate. Cele mai slabe performanțe pentru PSNR(15.9171) cât și pentru

SSIM(0.4805) sunt pentru zgomotul gaussian de medie 3 și deviație standard 4, lucru puțin surprinzător deoarece reprezintă intensitatea medie de zgomot. În ceea ce privește zgomotul impulsiv filtrat de filtrul de medie aritmetică cea mai bună performanță se obține pentru intensitate 20% (PSNR=10.2435, SSIM=0.4810), iar cea mai slabă performanță pentru intensitate 40% la SSIM=0.4184 si pentru intensitate 30% la PSNR=10.0556. Pentru filtrul median la zgomotul gaussian cea mai bună la PSNR = 19.9910 (medie 4, deviație standard 5), la SSIM = 0.6928 (medie 2, deviație standard 3). La capătul opus se află PSNR = 19.0231 (medie 3, deviație standard 4) și SSIM = 0.6783 (medie 3, deviație standard 4), iar pentru zgomotul impulsiv se obține PSNR-ul cel mai mare cu valoarea 10.0309 și SSIM cel mai mare cu valoarea 0.5079 la intensitate 20%, precum și cele mai slabe perfomanțe la PSNR= 9.6331 și SSIM = 0.4097 la intensitate 40%, lucru de așteptat. Pentru filtrul amn-mmknn pentru zgomotul gaussian se obțin cele mai bune valori pentru PSNR = 7.8635 la medie 2, deviație standard 3, SSIM = 0.0626 la medie 4, deviație standard 5. Cele mai slabe valori pentru PSNR = 7.8419, SSIM = 0.0618 la medie 3, deviație standard 4. Pentru zgomotul impulsiv la același filtru se obțin cele mai bune valori pentru PSNR = 7,8635 la intensitate 20%, SSIM = 0.0626 la intensitate 40%; cele mai slabe valori pentru PSNR = 7.8419, SSIM = 0.0618la intensitate 30%

- pentru tabelul 3.7 la filtrul de medie aritmetică pentru zgomot gaussian cele mai bune valori pentru PSNR = 17.0641, SSIM = 0.5981 la medie 3, deviație standard 4; cele mai slabe performanțe pentru PSNR = 16.579, SSIM = 0.5853 la medie 4, deviație standard 5. Pentru zgomotul impulsiv se obțin astfel: cele mai bune valori pentru PSNR = 10.4366 la intensitate 40%, SSIM = 0.4621 la intensitate 20%; cele mai slabe performanțe pentru PSNR = 10.1140 la intensitate 20%, SSIM = 0.4285 la intensitate 40%. La filtrul median pentru zgomotul gaussian se obțin: cele mai bune rezultate pentru PSNR = 19.2926 la medie 2, deviație standard 3, SSIM = 0.7077 la medie 3, deviație standard 4; cele mai slabe rezultate pentru PSNR = 18.6278, SSIM = 0.6891 la medie 4, deviație standard 5. Pentru zgomotul impulsiv se obțin cele mai bune valori pentru PSNR = 9.4955, SSIM = 0.4432 la intensitate 20%; cele mai slabe performanțe pentru PSNR = 9.4336, SSIM = 0.3846 la intensitate 40%, lucru de așteptat. La filtrul amn-mmknn pentru zgomotul gaussian cele mai bune performanțe pentru PSNR = 5.9072 la medie 2, deviație standard 3, SSIM = 0.0403 la medie 4, deviație standard 5, cele mai slabe performanțe pentru PSNR = 5.8774 la medie 4, deviație standard 5, SSIM = 0.0360 la medie 2, deviație standard 3. Pentru zgomotul impulsiv cele mai bune performanțe pentru PSNR = 5.9072 la intensitate 20%, SSIM = 0.0403 la intensitate 40%, cele mai slabe performanțe pentru PSNR = 5.8774 la intensitate 40%, SSIM = 0.0360 la intensitate 20%.

4. Concluzii

Filtrul descris de articol este unul folositor deoarece este o combinație de 2 filtre: unul cu mai multe canale fără parametri ce prezintă importanță pentru domeniul procesării imaginilor deoarece se poate aplica pe toate cele 3 canale de culoare RGB și unul bazat pe cei mai apropiați K vecini de tip M care a fost adaptat pentru imagini color pentru a se păstra detaliile.

Din punct de vedere teoretic ce ar fi trebuit să se obțină în urma implementării algoritmului pe baza articolului ar fi imagini filtrate rezonabil. Totuși implementarea furnizată și descrisă în această documentație nu se ridică la nivelul așteptărilor deoarece rezultatele obținute sunt mai slabe față de cele din articol.

Dar metoda discutată în articol contribuie totuși la dezvoltarea procesării imaginilor, în special pe partea de reducere a zgomotului impulsiv pe imagini color.

5. Comentarii personale

Din punctul meu de vedere ar fi fost util un pseudocod ajutător așa cum se mai oferă și în alte articole deoarece ciclul while (ciclul care se oprește atunci când $\hat{e}_{VMMKNN}^{(q)} = \hat{e}_{VMMKNN}^{(q-1)}$) care este practic nucleul algoritmului pentru că acolo se calculează valorile K_c care servesc mai departe în calculele necesare filtrării imaginilor, nu a fost explicat pe întelesul meu, iar acest lucru a influențat în mod clar implementarea algoritmului.

6. Erori

De-a lungul implementării algoritmului s-au întâlnit mai multe erori după cum urmează:

- eroarea menționată la referința [5] în care se încerca împărțirea la 0. Acest lucru s-a rezolvat prin modificarea ciclului while din funcția **med** astfel

```
while len(values) != 0 and values[int(np.floor(len(values)/2))] == 0:#check to see if
0 exists in window
        values.pop(int(np.floor(len(values)/2))) # eliminate values of 0
```

- eroarea cu mesajul : “OpenCV(4.1.2) /io/opencv/modules/core/src/arithm.cpp:687: error: (-5:Bad argument) When the input arrays in add/subtract/multiply/divide functions have different types, the output array type must be explicitly specified in function 'arithm_op' “. Aceasta s-a rezolvat conform [6] adăugând .astype(`uint8`) la variabila *gaussian* deoarece se lucrează cu o imagine floating-point în intervalul [0,1], dar este nevoie să se lucreze în intervalul [0,255], iar uint8 se asigură că se întâmplă acest lucru.
- se obține o imagine albă dacă se aplică zgomotul folosind următoarea sintaxă *noisy_image_gaussian = img1+gaussian* așa cum se observă și în figura următoare:

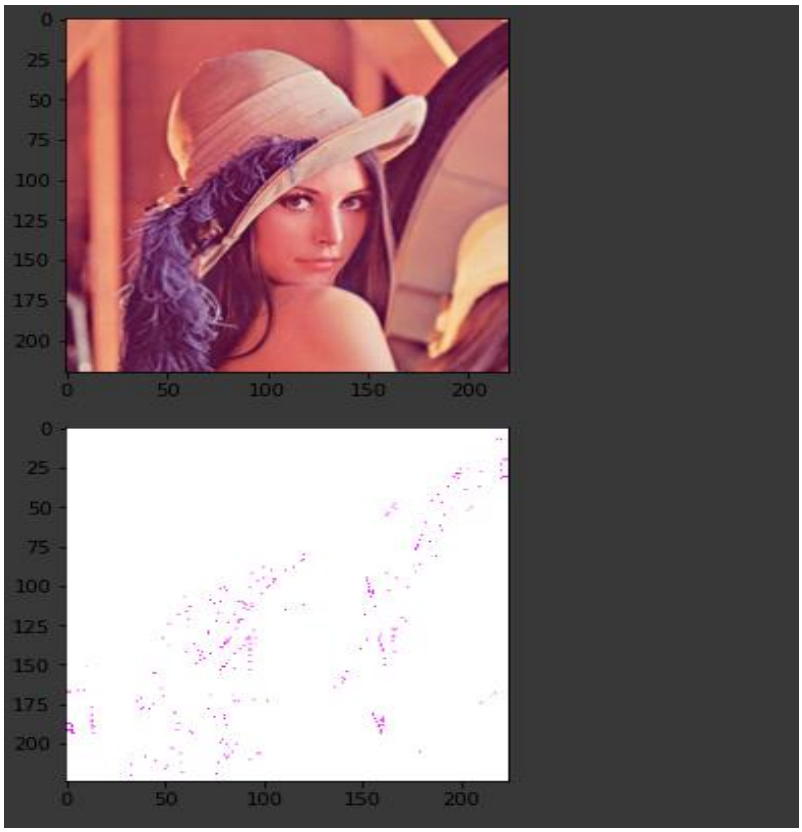

Figura 6.1: Imagine albă cu zgomot gaussian

O soluție care face ca imaginea să nu mai fie albă pe majoritatea suprafeței ar fi referința [7] unde se menționează la final că *img+noise* ar da rezultate nedorite deoarece OpenCV tratează 250+10 ca 255, în timp ce Numpy tratează (250+10)%255=5 și în loc de *img+noise* să se utilizeze *cv2.add(img, noise)*. Chiar și adaptând *noisy_image_gaussian = img1 + gaussian* la *noisy_image_gaussian = cv2.add(img1,gaussian)* se obține următoarea figură**:**

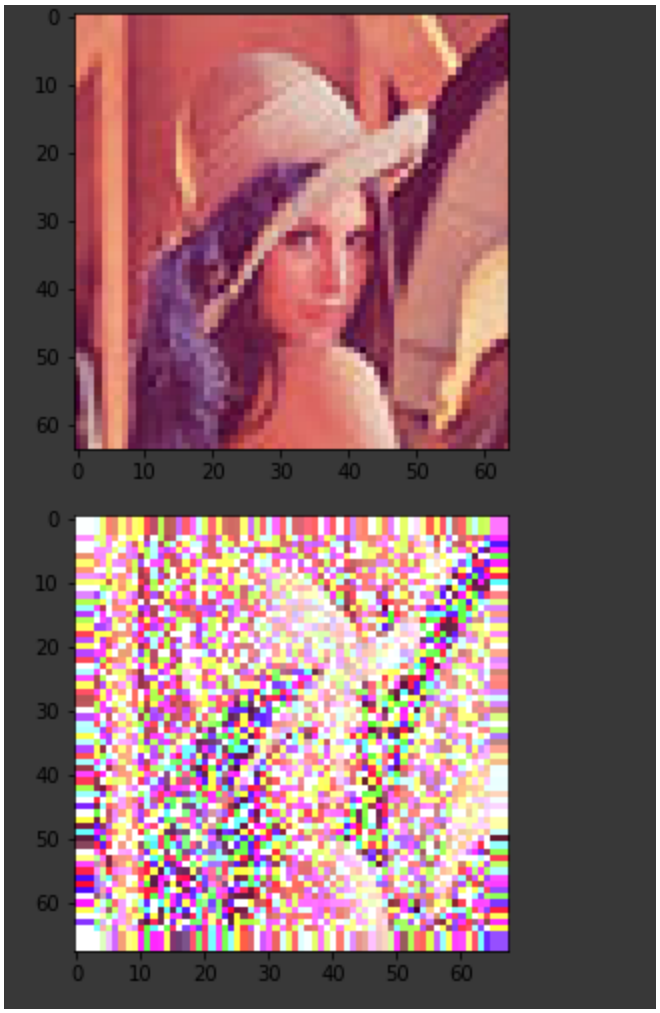

Figura 6.2: Imagine cu zgomot gaussian

- se încearcă iar o împărțire la 0 în figura următoare:

```
ValueError                                Traceback (most recent call last)
<ipython-input-19-107ebe1d5664> in <module>()
----> 1 plt.figure(), plt.imshow(amn_mmknn_filter(padded_noisy_image_impulsive))

<ipython-input-18-b6b7f3ed7553> in amn_mmknn_filter(image)
    215                 for l1 in range(len(window)):
    216                   filter_out = filter_out + vmmknn_window[l1] * (smooth_parameter(window, window[l1]) * kernel_function(window, channel[i][j], window[l1])) / result_jos
--> 217                 filter_output[i][j] = int(filter_out)
    218                 ##########################################################
    219       new_image[:,:,ch] = filter_output # filter for each channel

ValueError: cannot convert float NaN to integer

 SEARCH STACK OVERFLOW
<Figure size 432x288 with 0 Axes>
```

Figura 6.3: Împărțire la 0

Problema s-a remediat prin adăugarea următoarelor 2 linii la for-ul cu *l2* în calculul ecuației (1) din articol după ce se calculează *result_jos*:

```
if result_jos == 0:
                    result_jos += 0.1
```

- eroarea cu mesajul: “ only length-1 arrays can be converted to Python scalars “. Practic se încearcă să se ia mai mult de 1 argument; *math.floor* nu acceptă un array cu dimensiunea >= 2 ci doar un scalar așa cum se observă în figură. Eroarea s-a rezolvat folosind în schimb *np.floor*.

```
<ipython-input-18-8332392bc5c9>:92: RuntimeWarning: overflow encountered in ushort_scalars
  window[i][j] = abs(window[i][j] - scalar[j])
<ipython-input-18-8332392bc5c9>:108: RuntimeWarning: divide by zero encountered in true_divide
  return math.floor(np.divide(med(window,scalar,scalar_index),(mad(window,scalar_index)))) + math.floor(0.5*(np.divide(mad(wind
ow,scalar_index),med(window,scalar_index))))

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-8332392bc5c9> in <module>
    117         K_c_results=[]
    118         for k in range(len(window)):
--> 119             result=K_min + a * spike_detector(window,window[k],k)
    120             if result <= K_max:
    121                 K_c_indexes.append(k)

<ipython-input-18-8332392bc5c9> in spike_detector(window, scalar, scalar_index)
    106
    107 def spike_detector(window, scalar, scalar_index):
--> 108     return math.floor(np.divide(med(window,scalar,scalar_index),(mad(window,scalar_index)))) + math.floor(0.5*(np.divid
e(mad(window,scalar_index),med(window,scalar_index))))
    109
    110 for i in range(2, len(norm_image) - 2):

TypeError: only size-1 arrays can be converted to Python scalars
```

Figura 6.4: Accesarea funcției math.floor cu mai mult de 1 argument

- eroarea cu mesajul menționat în referința [8]

7. Bibliografie

[1] V. Ponomaryov, F. Gallegos-Funes, A. Rosales-Silva, *Adaptive multichannel non-parametric median M-type K-nearest neighbour (AMNMMKNN) filter to remove impulsive noise from colour images*, 2004

[2] *PSNR and SSIM Metric: Python Implementation*, https://cvnote.ddlee.cc/2019/09/12/psnr-ssim-python, accesat la data: 30.01.2021

[3] *OpenCV Python Image Smoothing – Gaussian Blur*, https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/, accesat la data: 30.01.2021

[4] *Smoothing Images*, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html, accesat la data: 30.01.2021

[5] *RuntimeWarning: invalid value encountered in divide*, https://stackoverflow.com/questions/14861891/runtimewarning-invalid-value-encountered-in-divide, accesat la data: 31.01.2021

[6] *Add different noise to an image*, https://theailearner.com/2019/05/07/add-different-noise-to-an-image/, accesat la data: 31.01.2021

[7] *How to add noise (Gaussian/salt and pepper etc) to image in Python with OpenCV [duplicate]*, https://stackoverflow.com/questions/22937589/how-to-add-noise-gaussian-salt-and-pepper-etc-to-image-in-python-with-opencv, accesat la data: 31.01.2021

[8] *ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()*, https://stackoverflow.com/questions/10062954/valueerror-the-truth-value-of-an-array-with-more-than-one-element-is-ambiguous, accesat la data: 31.01.2021

[9] *What is the Maximum PSNR and SSIM values for HR images ?*, https://www.mathworks.com/matlabcentral/answers/460468-what-is-the-maximum-psnr-and-ssim-values-for-hr-images, accesat la data: 31.01.2021
[10] Gallegos-Funes, F.J., Ponomaryov, V., Sadovnychiy, S., and Nino-de-Rivera, L.: 'Median M-type K-nearest neighbour (MMKNN) filter to remove impulse noise from corrupted images', Electron. Lett., 2002, 38, (15), pp. 786–787