

Efficient Block Matching for Removing Impulse Noise

Cuprins

Introducere	3
Explicarea metodei folosite	4
Explicarea termenilor importanți.....	4
Identificarea pixelilor zgomotoși.....	4
Construirea unei matrici de indicare pentru găsirea blocurilor de potrivire.....	5
Eliminarea zgomotului	5
Ștergerea zgomotului din blocurile ţintă	5
Ștergerea zgomotului din blocurile de zgomot.....	6
Extinderea metodei pe imagini color.....	8
Explicarea codului	9
Matricea de co-apariție pentru nivele de gri.....	9
Intervalul de omogenitate	9
Crearea unei matrici de zgomot	10
Crearea seturilor de blocuri de potrivire, ţintă și zgomotoase.....	11
Ștergerea zgomotului din blocuri ţintă	12
Ștergerea zgomotului din blocuri zgomotoase	15
Salvarea canalului filtrat	16
Funcții ajutătoare.....	16
Rezultate ale metodei + comparații cu metode uzuale	18
airplane.png	20
baboon.png	22
fruits.png	24
goldmedal.jpg	26
horse.jpg	28
lena.png (2 tipuri de zgomot * 3 intensități diferite).....	30
gaussian cu mean = 0, sigma =5; impulsiv 5%.....	30
gaussian cu mean = 0, sigma =10; impulsiv 10%	32
gaussian cu mean = 30, sigma =10; impulsiv 20%	34
sarah.jpg (2 tipuri de zgomot * 3 intensități diferite).....	36
gaussian cu mean = 0, sigma =5; impulsiv 5%.....	36

Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

gaussian cu mean = 0, sigma =10; impulsiv 10%	38
gaussian cu mean = 30, sigma =10; impulsiv 20%	40
Concluzii și comentarii personale	42
Bibliografie	43

Introducere

Acest document explică o nouă metodă de reducere a zgomotului din imagini bazată pe blocuri, prezentă în articolul din [1].

Această metodă este folosită pentru a scăpa într-un mod eficient de zgomotul impulsiv care poate apărea într-o imagine.

Metoda expusă va fi extinsă să funcționeze atât pe imagini gri cât și pe imagini color și se va afișa o posibilă implementare a algoritmului.

De asemenea, se va încerca compararea algoritmului implementat cu alte metode cunoscute de filtrare a zgomotului din imagine și se vor trage concluzii în legătură cu performanța și utilitatea acestei idei.

Explicarea metodei folosite

Conceptul principal de înțeles în această metodă este de a potrivi blocuri similare folosind un tabel de căutare. În termeni cât mai simpli, metoda bazată pe potrivirea blocurilor similare se referă la căutarea unor sub-secțiuni din imagine care conțin pixeli zgomotoși și la eliminarea acestor pixeli folosind alte sub-secțiuni similare din imagine. Filtrarea acestor blocuri, găsirea pixelilor zgomotoși și procesul de a determina similaritatea între blocuri este explicat mai jos.

Acest algoritm funcționează pentru imagini cu un singur canal de culoare, aceste imagini fiind grayscale. În continuare, algoritmul va fi explicat pentru imaginile grayscale dar poate fi extins și pentru imagini color.

Explicarea termenilor importanți

- Tabel de căutare (lookup table) – matrice care conține informații pre-calculate pentru a salva timpul de rulare al unui algoritm, în acest caz matricea este folosită pentru a salva locațiile blocurilor de potrivire.
- Sub-secțiuni – blocuri de mărime $n \times n$ care pot fi considerate ca niște ferestre glisante care parcurg întreaga imagine în identificarea tipurilor de bloc.
- Pixel zgomotos – un pixel care este identificat ca fiind zgomotos în cadrul unei imagini cu zgomot, procesul de identificare va fi explicat mai jos.
- Bloc de potrivire – bloc care conține un număr mai mic de pixeli zgomotoși decât un prag predefinit P , numărul de pixeli zgomotoși poate să fie 0.
- Bloc zgomotos – bloc care conține un număr mai mare sau egal de pixeli zgomotoși decât un prag predefinit P .
- Bloc țintă – bloc care nu este un bloc zgomotos dar are cel puțin un pixel zgomotos, numărul de pixeli zgomotoși se află în intervalul $[1, P-1]$, poate fi de asemenea un bloc de potrivire.

Identificarea pixelilor zgomotoși

Mai întâi, se creează o matrice H de co-apariție pentru nivele de gri pentru imaginea cu zgomot. Această matrice de 256×256 elemente conține următoarea informație: $H(i, j)$ este egal cu numărul de pixeli vecini cu valoarea j care se află în jurul pixelului cu valoarea i . De exemplu, imaginea de jos:

1	1	1	5
2	9	2	5
3	3	9	5
4	4	4	5

dacă se consideră că pixelii vecini se pot afla cu o poziție mai sus sau mai jos și/sau mai în stânga sau mai în dreapta $\rightarrow H(9, 1) = 3 ; H(9, 2) = 3 ; H(9, 3) = 3 ; H(9, 4) = 2 ; H(9, 5) = 3 ; H(9, 9) = 2$

Cu această matrice generată, mai departe se va calcula un interval de omogenitate $[low_{\delta}(i) ; high_{\delta}(i)]$ pentru toate valorile i de la 0 la 255 din matricea de co-apariție. Cele două extreme ale intervalului sunt calculate cu formulele (3) și (4) adaptate din articolul [2] :

$$low_{\delta}(i) = \arg \min_j \left(\frac{1}{3} \sum_{k=-1}^1 P(I, j+k) \geq \tau \right)$$

$$high_{\delta}(i) = \arg \max_j \left(\frac{1}{3} \sum_{k=-1}^1 P(I, j+k) \geq \tau \right)$$

Ecuația 1. Formulele pentru capetele intervalului de omogenitate. [2]

unde τ este un prag. „Acesta ecuații caută pur și simplu slotul histogramei în care media mobilă în 3 puncte este mai mică decât τ , ceea ce indică faptul că majoritatea sloturilor din jurul său au valori mici.“ [2]

„Pentru fiecare pixel q adiacent lui p , dacă p se află în intervalul de omogenitate a lui q , atunci contorul este mărit cu 1. După ce sunt luați în considerare toți cei opt pixeli vecini, dacă contorul este mai mic decât un prag, atunci p este clasificat ca zgomot.“ [1]

Construirea unei matrici de indicare pentru găsirea blocurilor de potrivire

După identificarea pixelilor zgomotoși, se va folosi o fereastră glisantă de 3×3 care va parcurge imaginea pixel cu pixel iar pixelii zgomotoși din acea fereastră vor fi numărați. În funcție de numărul de pixeli zgomotoși din imagine, fereastra va fi ori un bloc zgomot, ori un bloc de potrivire ori un bloc țintă.

Dacă blocul este unul de potrivire, poziția acestuia va fi memorată într-o matrice $A[0...8][0...255][0...MAX_PIX]$ unde:

- prima dimensiune se referă la locația pixelului din fereastra 3×3 , [1]
- a doua dimensiune se referă la valoarea nivelului de gri al pixelului din fereastra respectivă, [1]
- a treia dimensiune se referă la numărul de pixeli care au aceeași valoare a nivelului de gri și aceeași locație în contextul ferestrei 3×3 , unde MAX_PIX este acel număr, [1]
- iar rezultatul de la sfârșit este coordonata (r, c) a blocului care este convertită într-o valoare scalară folosind expresia $r * w + c$ unde w este lățimea imaginii, iar r și c sunt linia și coloana la care se află pixelul din centrul ferestrei, în imagine. [1]

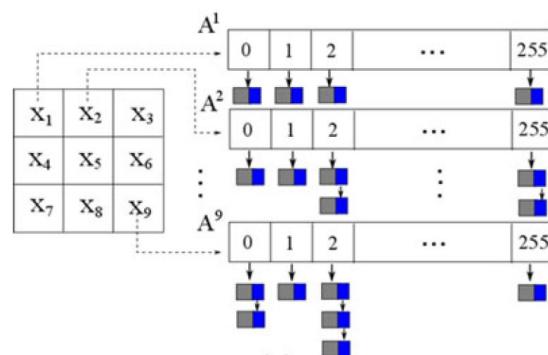


Figura 1. Locația pixelilor X_i dintr-o fereastră 3×3 și cum corespund aceștia elementelor matricei A_i .
[1]

Eliminarea zgomotului

Ștergerea zgomotului din blocurile țintă

Pentru fiecare bloc țintă găsit, se vor parcurge toți pixelii din acel bloc X /acea fereastră X de 3×3 . Pentru toți pixelii X_i care au valoarea p și care nu sunt zgomotoși, se va căuta în matricea $A[i][p-d...p+d]$ blocuri de potrivire, unde d este o valoare de toleranță.

Valoarea de toleranță este folosită pentru a căuta pixeli în blocuri țintă care au valori de gri similare cu valoarea p pentru a crește şansele de a găsi candidați buni pentru filtrarea zgomotului.

Atunci când se găsește un bloc de potrivire, se va utiliza o funcție de similaritate între acel bloc și blocul țintă pentru a determina dacă blocul de potrivire curent este un candidat potrivit pentru filtrarea zgomotului din blocul țintă.

Funcția de similaritate este descrisă în felul următor:

$$\text{simil}(T, M) = \sum_{i=1}^9 \text{sgn}(t_i) \cdot \text{sgn}(p_i) \cdot \text{tol}_d(t_i, p_i)$$

Ecuăția 2. Funcția de similaritate între un bloc țintă și bloc de potrivire [1]

unde

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x \notin I_{\text{noise}} \\ 0, & \text{otherwise} \end{cases} \text{ și } \text{tol}_d(x, y) = \begin{cases} 1, & \text{if } |x - y| \leq d \text{ and } x \notin I_{\text{noise}}, y \notin I_{\text{noise}} \\ 0, & \text{otherwise} \end{cases}$$

Ecuăția 3. Funcția de semn și funcția de toleranță [1]

unde t_i este pixelul i din blocul țintă T , p_i este pixelul i din blocul de potrivire M iar I_{noise} este imaginea care descrie dacă anumiți pixeli sunt zgomotoși sau nu (exemplu: $x \notin I_{\text{noise}}$ înseamnă că pixelul x nu este unul zgomots).

Funcția de similaritate ne spune la sfârșit câte perechi de pixeli între blocul țintă și blocul de potrivire sunt similare, rezultatul fiind în intervalul $[0, 9]$.

Dacă acest rezultat este mai mare decât un anumit prag τ , atunci blocul de potrivire va fi memorat într-un set $\Sigma(T)$:

$$\Sigma(T) = \{M \mid \text{simil}(T, M) \geq \tau\}$$

Ecuăția 4. Setul de blocuri de potrivire care îndeplinește o anumită condiție [1]

Dacă lungimea setului ajunge la o valoare egală sau mai mare decât $\#S$, procesul de căutare se oprește pentru acel bloc țintă. În acest caz, „eliminarea zgomotului din blocul țintă T se face prin înlocuirea pixelilor zgomotoși din T cu media tuturor pixelilor curați la locația corespunzătoare a tuturor blocurilor de potrivire din $\Sigma(T)$.“ [1]

Dacă lungimea setului nu ajunge la valoarea $\#S$, nivelul toleranței d va crește cu Δd și procesul de căutare pornește din nou de la început până se va găsi un număr suficient de blocuri de potrivire.

Ștergerea zgomotului din blocurile de zgomot

Se vor trece prin toate blocurile de zgomot iar fiecare bloc de zgomot B va fi împărțit în 4 blocuri înconjurătoare.

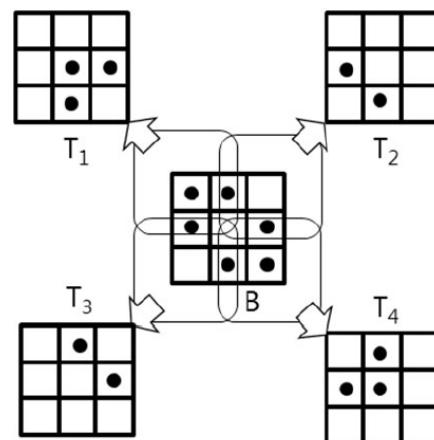


Figura 2. Un bloc zgomotos B este împărțit în 4 blocuri înconjurătoare. [1]

Pentru aceste blocuri înconjurătoare T_i, se verifică pentru fiecare bloc dacă numărul de pixeli zgomotoși este mai mic decât un anume prag. Dacă această condiție este îndeplinită, blocul zgomotos va fi transformat într-un bloc țintă. După transformarea acestor blocuri, se începe din nou ștergerea pixelilor din blocurile țintă.

Dacă acest proces nu scapă de toate blocurile țintă și de toate blocurile zgomotoase, există soluția de a împărți aceste blocuri în blocuri de mărime 2x2 iar procesul de potrivire al blocurilor continuă în acest mod.

Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

Extinderea metodei pe imagini color

Din moment ce metoda descrisă funcționează doar pentru imagini cu un singur canal de culoare, filtrul poate fi extins să meargă și pe imagini color.

Acest lucru se poate face foarte simplu prin a extrage fiecare canal de culoare din imaginea zgombatoasă. Se filtrează fiecare canal de culoare prin metoda descrisă mai sus iar la final toate canalele de culoare filtrate sunt îmbinate pentru a forma imaginea color filtrată.

Explicarea codului

Pentru implementarea filtrului descris, s-a ales limbajul MATLAB care dispune de o suită de utilități folositoare pentru procesarea imaginilor.

Filtrul funcționează ca o funcție de MATLAB care primește ca parametru de intrare imaginea care se vrea filtrată iar la ieșire, rezultatul funcției va conține imaginea filtrată.

Se verifică câte canale de culoare are imaginea de la intrare și pentru fiecare canal se va face filtrarea descrisă mai sus.

```
%get number of image channels
image_channels = size(I, 3);

%for every channel in the image
for image_channel = 1:1:image_channels
    %get the image channel separately
    img = I(:,:,image_channel);
```

Matricea de co-apariție pentru nivele de gri

Pentru fiecare canal de culoare, se va crea matrice de co-apariție pentru nivele de gri folosind funcția graycomatrix din MATLAB. [3]

```
%create the channel's gray co-occurrence matrix with the closest 8
%neighbours
H = graycomatrix(img, 'NumLevels', gray_levels, 'Offset', neighbours);

%since the function gives the co-occurrence matrix for EVERY
%neighbour SEPARATELY, sum up all the matrices
H = sum(H, 3);
```

Funcția graycomatrix primește ca parametru canalul de culoare separat și i se spune să facă o matrice pentru 256 de nivele de gri, unde offset-ul pentru acest calcul este dat în vectorul neighbours.

```
%set number of gray levels
gray_levels = 256;

%set offset of neighbour pixels array
neighbours = [-1 -1; -1 0; -1 1; 0 -1; 0 1; 1 -1; 1 0; 1 1];
```

Vectorul neighbours conține offset-urile pentru un anumit pixel. De exemplu, dacă trebuie luat pixelul vecin de sus, se vor folosi indecșii $i + \text{neighbours}[2][1]$ și $j + \text{neighbours}[2][2]$ pentru obținerea liniei $(i-1)$ și coloanei $(j+0)$ vecinului respectiv.

Intervalul de omogenitate

Se vor trece prin toate valorile i de la 1 până la 256 (indexarea în MATLAB începe de la 1) și pentru fiecare valoare i vom ține minte valorile j care îndeplinesc condiția prezentă în Ecuația 1.

```
%homogeneity upper bound values
h_up = zeros(1, gray_levels);

%homogeneity lower bound values
h_low = zeros(1, gray_levels);

%going through the gray co-occurrence matrix
for i = 1:1:gray_
```

```
%setting threshold to pass
threshold = 2;
values = [];
for j = 1:1:gray_levels
    result = 0;
    counter = 0;
    %move with a 1x3 window through the matrix
    k_values = [-1:1:1];
    for k = k_values
        if j+k >= 1 && j+k <= 256
            result = result + H(i, j+k);
            counter = counter + 1;
        end
    end
    %get the mean value
    result = result / counter;
    %if the value passed the threshold, save the index where it
    %happened
    if result >= threshold
        values = [values j];
    %otherwise put -1 in its' place to keep track of failures
    else
        values = [values -1];
    end
end
%if for a value i there are no valid indexes, set the lower and
%upper bound at i to itself
if all(values == -1)
    h_up(1, i) = i;
    h_low(1, i) = i;
%otherwise search argmax and argmin
else
    [~, h_up(1,i)] = max(values);
    %here we replace all -1 values with +inf, so the min
    %function doesn't choose an invalid index
    values(values == -1) = Inf;
    [~, h_low(1,i)] = min(values);
end
end
```

Crearea unei matrici de zgomot

În continuare, se va aplica padding peste imaginea originală astfel încât să putem folosi o fereastră glisantă 3x3 care să proceseze toți pixelii importanți.

```
%create padded image so we can process all the meaningful pixels
padded_img = padarray(img, [1 1], 'symmetric');

%create array where we keep track of noisy pixels locations
%if noisy_pixels(i, j) == 1 -> noisy pixel
%else -> not a noisy pixel
noisy_pixels = zeros(size(padded_img));
```

Pentru fiecare pixel din fereastra glisantă, se verifică dacă pixelul este unul "zgomotos", adică dacă valoarea de gri a acestuia este în intervalul de omogenitate al pixelului central din fereastra respectivă. Dacă, în fereastra glisantă, se găsesc 6 sau mai mulți pixeli zgomotoși, atunci pixelul central din fereastră se consideră zgomotos și se notează acest lucru în matricea de zgomot.

```
%move with a 3x3 window and a stride of 1
for i = 2:1:size(padded_img,1)-1
    for j = 2:1:size(padded_img,2)-1
        center_value = padded_img(i, j);
        noise_counter = 0;
        for k = 1:1:size(neighbours,1)
            neighbour_value = padded_img(i+neighbours(k, 1), ours(k,2));
            low_value = h_low(1, center_value + 1);
            up_value = h_up(1, center_value + 1);

            %checking if the neighbour of the center pixel
            %of the window is in the range of homogeneity
            if neighbour_value < low_value || ...
                neighbour_value > up_value
                %if it's not, then we increase the noise counter
                noise_counter = noise_counter + 1;
            end

        end
        %if the noise counter is above a certain threshold
        if noise_counter >= (75/100) * size(neighbours, 1)
            %the center pixel of the moving window is considered
            %noisy
            noisy_pixels(i, j) = 1;
        end
    end
end
```

Crearea seturilor de blocuri de potrivire, țintă și zgomotoase

Se parurge matricea de zgomot cu o fereastră glisantă de 3x3. Dacă majoritatea pixelilor din acea fereastră nu sunt zgomotoși, atunci această fereastră este considerată un bloc de potrivire și se adaugă în setul respectiv. La fiecare 3 ferestre parcurse, se verifică dacă acea fereastră conține cel puțin un pixel zgomots. În acest caz, fereastra este adăugată în setul de blocuri țintă. Iar în cazul în care fereastra conține pixeli zgomotoși în majoritate, fereastra este adăugată în setul de blocuri zgomotoase. Adăugarea se face prin a transforma indecșii pixelului central al ferestrei în contextul imaginii originale într-un singur scalar. $(r,c) \rightarrow r*w+c$ unde w este lățimea imaginii.

```
%construct array for keeping track of matching blocks coordinates
matching_blocks = zeros(size(window,1), gray_levels);
%construct array for keeping track of frequency of pixels regarding
%matching blocks
freqs_matching_blocks = zeros(size(window,1), gray_levels);

%construct vector for keeping track of target blocks coordinates
target_blocks = [];

%construct vector for keeping track of noisy blocks coordinates
noisy_blocks = [];

%move with a 3x3 window with a stride of 1
for i = 2:1:size(noisy_pixels,1)-1
    for j = 2:1:size(noisy_pixels,2)-1
        noise_counter = 0;
        %count the number of noisy pixels
        for k = 1:1:size(window,1)
            if noisy_pixels(i+window(k,1), ...
                j+window(k,2)) == 1
```

```

        noise_counter = noise_counter + 1;
    end
end
%if the number of noisy pixels is smaller than a threshold
if noise_counter < uint8(size(window,1) / 2)
    %then the window is a matching block
    for k = 1:1:size(window, 1)
        freqs_matching_blocks(k, ...
            padded_img(i+window(k,1), ...
                j+window(k,2))+1) = freqs_matching_blocks(k, ...
                    padded_img(i+window(k,1), ...
                        j+window(k,2))+1);
        matching_blocks(k, padded_img(i+window(k,1), ...
            j+window(k,2))+1, freqs_matching_blocks(k, ...
                padded_img(i+window(k,1), ...
                    j+window(k,2))+1)) = get_scalar_from_indexes(i, j,
padded_img);
    end
end
%here we put an extra condition to check as if we moved
%with a 3x3 window and a stride of 3
if mod(i,3) == 2 && mod(j,3) == 2
    %if there is at least 1 noisy pixel
    if noise_counter >= 1
        %remember the window and its'
        %coordinate as a target block
        target_blocks = [target_blocks...
            get_scalar_from_indexes(i, j, padded_img)];
    end
    if noise_counter >= uint8(size(window,1)/2)
        noisy_blocks = [noisy_blocks...
            get_scalar_from_indexes(i, j, padded_img)];
    end
end
end
end

```

Ștergerea zgomotului din blocuri țintă

Parcurgem setul de blocuri țintă și pentru fiecare pixel din fereastra acestui bloc, se verifică dacă există blocuri de potrivire cu o valoare similară de gri în intervalul unei toleranțe predefinite.

Dacă se găsesc aceste blocuri de potrivire, calculăm o valoare de similaritate între blocul de potrivire și blocul țintă și dacă această valoare este mai mare decât un anume prag, se va ține minte blocul de potrivire într-un set diferit.

Odată ce lungimea acestui set atinge o anumite valoare minimă, procesul de căutare poate fi oprit. Pentru fiecare obiect din set, se verifică dacă pixelul X_i din blocul țintă este unul zgomots și dacă același pixel X_i din blocul de potrivire este unul zgomots. Se salvează media aritmetică a valorii de gri a tuturor pixelilor X_i care nu sunt zgomotoși din blocul de potrivire și acest rezultat duce la valoarea de gri nouă a pixelului din imagine, acesta fiind considerat un pixel eliminat de zgomot.

Dacă toți pixelii țintă din fereastră au fost eliminate de zgomot, acel bloc țintă se va transforma într-unul de potrivire.

Căutarea repornește cu o toleranță mai mare dacă setul de blocuri de potrivire nu scade.

```
%define tolerance bounds to move with through the matching blocks
%array
tolerance_lower = 1;
tolerance_step = 2;
tolerance_upper = 9;

%define thresholds
sufficient_number_of_matching_blocks = 5;
similarity_threshold = 3;

%variable to keep track if target blocks are decreasing
decreasing_target_blocks = 1;

%increase the tolerance if needed
for d = tolerance_lower:tolerance_step:tolerance_upper
    %while the target blocks are decreasing
    while decreasing_target_blocks == 1
        %for every target block
        for t = 1:1:size(target_blocks,2)
            current_matching_blocks = [];
            found_enough_blocks = 0;
            %get its' coordinate as indexes
            [target_row, target_col] =
                get_indexes_from_scalar(target_blocks(t), padded_img);
            for k = 1:1:size(window, 1)
                %if target pixel is not noisy
                if noisy_pixels(target_row + window(k,1), target_col +
window(k,2)) == 0
                    %get its' gray value
                    gray_value = padded_img(target_row + window(k,1),
target_col + window(k,2));

                    %get the tolerance-bound lower limit
                    lower_limit = gray_value-d+1;
                    if lower_limit < 1
                        lower_limit = 1;
                    end

                    %get the tolerance-bound upper limit
                    upper_limit = gray_value+d+1;
                    if upper_limit > gray_levels
                        upper_limit = gray_levels;
                    end

                    %get all matching blocks that have the same
                    %gray level that appears in the tolerance-bound
                    %interval
                    temp_list = matching_blocks(k, lower_limit:upper_limit,
:);

                    %for every possible matching block candidate
                    for i = 1:1:size(temp_list, 2)
                        for j = 1:1:size(temp_list, 3)
                            if temp_list(1, i, j) ~= 0
                                [matching_row, matching_col] =
                                    get_indexes_from_scalar(temp_list(1,i,j), padded_img);
                                %check if that matching block is
                                %similar to the target block
                            end
                        end
                    end
                end
            end
        end
    end
end
```

```

        if simil(target_row, target_col,
matching_row, matching_col, padded_img, noisy_pixels, d, window) >=
similarity_threshold
            %if it is, add it to a set
            current_matching_blocks =
[current_matching_blocks, temp_list(1,i,j)];
        end
            %if the set reaches a minimum
            %length, the search can be stopped
            if size(current_matching_blocks, 2) >=
sufficient_number_of_matching_blocks
                found_enough_blocks = 1;
                break
            end
        end
    end
    if found_enough_blocks == 1
        break
    end
end
if found_enough_blocks == 1
    break
end
end
%if the set reached the minimum length
if found_enough_blocks == 1
    %for every pixel in a 3x3 window
    for k = 1:size(window,1)
        %check if the Xi target pixel is noise so it can be
        %denoised
        if noisy_pixels(target_row + window(k,1), target_col +
window(k,2)) == 1
            result = 0;
            counter = 0;
            for idx = 1:size(current_matching_blocks, 2)
                [matching_row, matching_col] =
get_indexes_from_scalar(current_matching_blocks(1,idx), padded_img);
                %if the Xi matching pixel is not noise
                if noisy_pixels(matching_row + window(k,1),
matching_col + window(k,2)) == 0
                    result = result + padded_img(matching_row +
window(k,1), matching_col + window(k,2));
                    counter = counter + 1;
                end
            end
            %save the mean result of all Xi matching
            %pixels that are not noise
            result = result / counter;
            %denoise the target pixels
            if counter > 0
                padded_img(target_row + window(k,1), target_col +
window(k,2)) = result;
                noisy_pixels(target_row + window(k,1),
target_col + window(k,2)) = 0;
            end
        end
    end
    %check if every target pixel from a certain window
    %has been denoised
    everything_denoised = 1;

```

```

        for k = 1:size(window,1)
            if noisy_pixels(target_row + window(k,1), target_col +
window(k,2)) == 1
                everything_denoised = 0;
                break
            end
        end
        %if every target pixel from a certain window was
        %denoised
        if everything_denoised == 1
            %transform the target block into a matching
            %block
            for k = 1:size(window,1)
                freqs_matching_blocks(k,
padded_img(target_row+window(k,1),target_col+window(k,2))+1) =
freqs_matching_blocks(k,
padded_img(target_row+window(k,1),target_col+window(k,2))+1) + 1;
                matching_blocks(k, padded_img(target_row +
window(k,1), target_col + window(k,2))+1, freqs_matching_blocks(k,
padded_img(target_row+window(k,1),target_col+window(k,2))+1)) =
target_blocks(t);
            end
            %remove the target block
            target_blocks(t) = [];
            decreasing_target_blocks = 1;
            break
        end
    else
        %if the number of target blocks is not decreasing
        %anymore, the tolerance will be increased to check
        %a higher range of gray values in the set of
        %matching blocks
        decreasing_target_blocks = 0;
    end
end
%if there are no more target blocks to denoise
if size(target_blocks,2) == 0
    decreasing_target_blocks = 0;
end
end

```

Ștergerea zgomotului din blocuri zgomotoase

Pentru fiecare bloc zgomotos, se vor crea 4 blocuri înconjurătoare. Pentru fiecare bloc înconjurător, se va verifica dacă numărul de pixeli zgomotoși din fereastra respectivă acestuia nu este majoritar. Dacă acesta este cazul, blocul înconjurător va fi înregistrat ca un bloc întă.

```

%denoising of noisy blocks
surrounding = zeros(9, 2, 4);
%get the offsets of every surrounding block
surrounding(:,:,1) = [-2 -2; -2 -1; -2 0; -1 -2; -1 -1; -1 0; 0 -2; 0 -1; 0
0];
surrounding(:,:,2) = [-2 0; -2 1; -2 2; -1 0; -1 1; -1 2; 0 0; 0 1; 0 2];
surrounding(:,:,3) = [0 -2; 0 -1; 0 0; 1 -2; 1 -1; 1 0; 2 -2; 2 -1; 2 0];
surrounding(:,:,4) = [0 0; 0 1; 0 2; 1 0; 1 1; 1 2; 2 0; 2 1; 2 2];
for t = 1:size(noisy_blocks,2)
    [noisy_row, noisy_col] = get_indexes_from_scalar(noisy_blocks(t),
padded_img);
    for k = 1:size(surrounding,3)

```

Proiect PAIC, numărul 15

```

min_row = min(surrounding(:,1,k));
min_col = min(surrounding(:,2,k));
max_row = max(surrounding(:,1,k));
max_col = max(surrounding(:,2,k));
if noisy_row + min_row < 2 || noisy_row + max_row > size(padded_img,
1) - 1 || noisy_col + min_col < 2 || noisy_col + max_col > size(padded_img,
2) - 1
    continue
end
noisy_counter = 0;
for idx = 1:1:size(surrounding(:,:,k), 1)
    if noisy_pixels(noisy_row + surrounding(idx, 1, k), noisy_col +
surrounding(idx, 2, k)) == 1
        noisy_counter = noisy_counter + 1;
    end
end
%if a surrounding block doesn't have enough noisy pixels
if noisy_counter < 5
    offset_i = -1;
    offset_j = -1;
    if k == 1
        offset_i = -1;
        offset_j = -1;
    elseif k == 2
        offset_i = -1;
        offset_j = 1;
    elseif k == 3
        offset_i = 1;
        offset_j = -1;
    elseif k == 4
        offset_i = 1;
        offset_j = 1;
    end
    %turn it into a target block to be denoised later
    target_blocks = [target_blocks get_scalar_from_indexes(i +
offset_i, j + offset_j, padded_img)];
end
end

```

[Salvarea canalului filtrat](#)

Canalul filtrat este salvat în canalul imaginii de ieșire iar filtrarea canalelor următoare continuă aşa cum s-au descris metodele de mai sus.

```
%save the filtered image channel and remove the padding
J(:,:,:,image_channel) = padded_img(2:size(padded_img,1)-1,
2:size(padded_img,2)-1);
```

[Funcții ajutătoare](#)

```

function [row, col] = get_indexes_from_scalar(scalar, fitting_array)
    row = ceil(scalar / size(fitting_array, 2));
    col = mod(scalar, size(fitting_array, 2));
end

function scalar = get_scalar_from_indexes(row, col, fitting_array)
    scalar = (row-1)*size(fitting_array, 2) + col;
end
```

```
function result = simil(T_row, T_col, M_row, M_col, padded_img, ...
    noisy_pixels, tolerance, window)

    %similarity output result
    result = 0;

    for i = 1:size(window,1)
        prod = 1;

        %check if pixel in target block window is noise
        if noisy_pixels(T_row + window(i,1), ...
            T_col + window(i,2)) == 1
            prod = 0;
        end

        %check if pixel in matching block window is noise
        if prod == 1
            if noisy_pixels(M_row + window(i,1), ...
                M_col + window(i,2)) == 1
                prod = 0;
            end
        end

        %check if pixels in matching and target blocks are not noise
        %and if their absolute difference is smaller than a pre-set
        %tolerance
        if prod == 1
            if ~ (noisy_pixels(M_row + window(i,1), ...
                M_col + window(i,2)) == 0 && ...
                noisy_pixels(T_row + window(i,1), ...
                T_col + window(i,2)) == 0 && ...
                abs(padded_img( M_row + window(i,1), ...
                M_col + window(i,2) ) ...
                - padded_img(T_row + window(i,1), ...
                T_col + window(i,2) )) <= tolerance)
                prod = 0;
            end
        end

        result = result + prod;
    end
end
```

Rezultate ale metodei + comparații cu metode uzuale

Setul de test pentru această metodă conține 5 imagini peste care au fost adăugate zgomot aditiv gaussian de deviație standard 10, respectiv separat 10% zgomot impulsiv și 2 imagini pe care s-au testat 3 intensități pentru fiecare tip de zgomot (means, variances = {[0, 5],[0,10],[30,10]}, probabilities = {0.05, 0.1, 0.2}).

Pentru fiecare imagine, rezultatele sunt afișate în felul următor:

- Compararea imaginii originale cu imaginile zgomotoase (original, gaussian, impulsiv)
- Imagineile de la fiecare ieșire de filtru și fiecare tip de zgomot + timpul de rulare al filtrului (gaussian + mean, gaussian + median, gaussian + BMLUT, impulsiv + mean, impulsiv + median, impulsiv + BMLUT)
- Diferențele vizuale între imaginea originală și imaginile de la fiecare ieșire de filtru și fiecare tip de zgomot + metricile de comparare SSIM, PSNR/SNR, MSE (diferențe între imaginea originală și gaussian + mean, gaussian + median, gaussian + BMLUT, impulsiv + mean, impulsiv + median, impulsiv + BMLUT)

În cazul zgomotului gaussian pentru valori mai mari ale deviației standard, pixelii afectați devin din ce în ce mai evidenți iar pentru valori mai mari ale mediei, imaginile par să devină mai luminoase. Pentru valori mai mari ale probabilității în cazul zgomotului impulsiv, imaginile afectate conțin din ce în ce mai mulți pixeli zgomoțiști.

Valoarea SSIM este folosită pentru a cuantifica informații cheie legate de luminanță, contrast și structură. Este o valoare între 0 și 1 iar pentru valori aproape de 1, imaginile comparate sunt foarte similare din punctul de vedere ale celor 3 factori umani menționați mai sus iar pentru valori aproape de 0, imaginile comparate sunt foarte diferite. [4]

Valorile PSNR/SNR și MSE sunt metrii care descriu niște metode puțin mai primitive de a compara calitatea imaginilor deoarece acestea sunt utile în cazul comparării filtrurilor. Cu cât sunt valorile mai mari pentru aceste metrii, cu atât se poate spune că o metodă de filtrare este mai bună decât celealte dar nu neapărat că și imaginea filtrată arată bine din punct de vedere calitativ.

Pentru valorile PSNR/SNR se poate preciza că valorile de 20-30 db descriu o imagine filtrată cu calitate mai mult decât acceptabilă iar pentru valorile de 10-20 db, imaginile filtrate lasă de dorit din punct de vedere calitativ.

Cu cât valorile MSE sunt mai mici, cu atât imaginile comparate sunt mai similare din punctul de vedere al structurii globale al imaginii, neluând în calcul anumite nuanțe subtile.

Diferențele vizuale sunt reprezentate sub formă de hărți SSIM. Pixelii albi reprezintă o similaritate foarte mare între pixelii din aceeași locație din imaginea originală cu cei din imaginea filtrată. Pixelii negri arată opusul, unde pixelii din aceeași locație din imaginea originală sunt foarte diferenți.

Aplicarea zgomotului aditiv gaussian s-a făcut prin funcția MATLAB:

```
function image_out = add_gaussian_noise(image_in, mean_value, dev)
    %get array of random normal random numbers
    noise = normrnd(mean_value, dev, size(image_in)); [5]
    %add the noise array
    image_out = uint8(double(image_in) + noise);
end
```

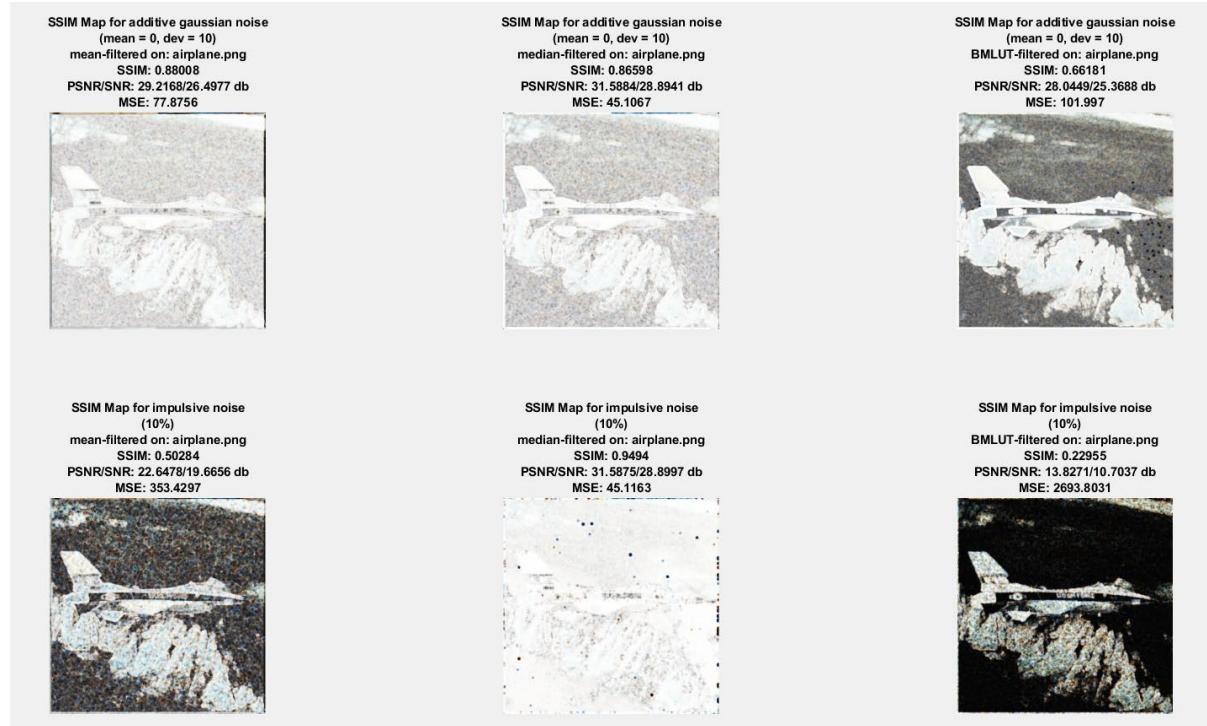
```
function image_out = add_impulsive_noise(image_in, prob)
    %affect a percentage of all the pixels
    image_out = imnoise(image_in, 'salt & pepper', prob); [6]
end
```

```
function image_out = mean_filter(image_in)
    image_out = image_in;
    %the mask for the mean filter (3x3 array where all values are 1/9)
    mask = ones(3)/9;
    channels = size(image_in, 3);
    %for every image channel
    for i = 1:1:channels
        %filter the image using the mask above
        image_out(:,:,:,i) = imfilter(image_in(:,:,:,i), mask); [7]
    end
end
```

```
function image_out = median_filter(image_in)
    image_out = image_in;
    channels = size(image_in, 3);
    %for every image channel
    for i = 1:1:channels
        %use a 3x3 window for the median filter
        image_out(:,:,:,i) = medfilt2(image_in(:,:,:,i)); [8]
    end
end
```

Trăistaru Vlad-Viorel
411-TAID
Proiect PAIC, numărul 15
airplane.png

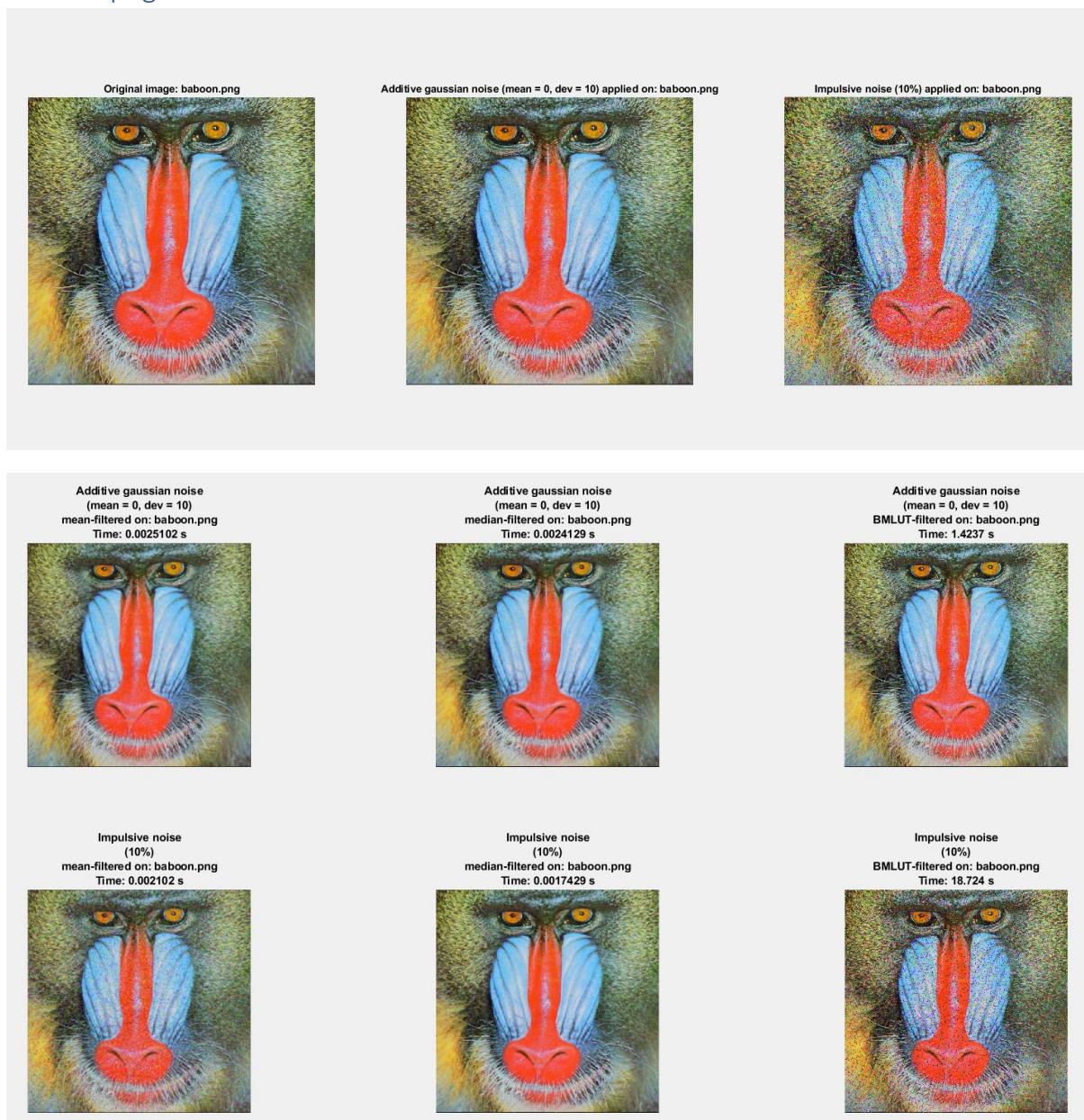


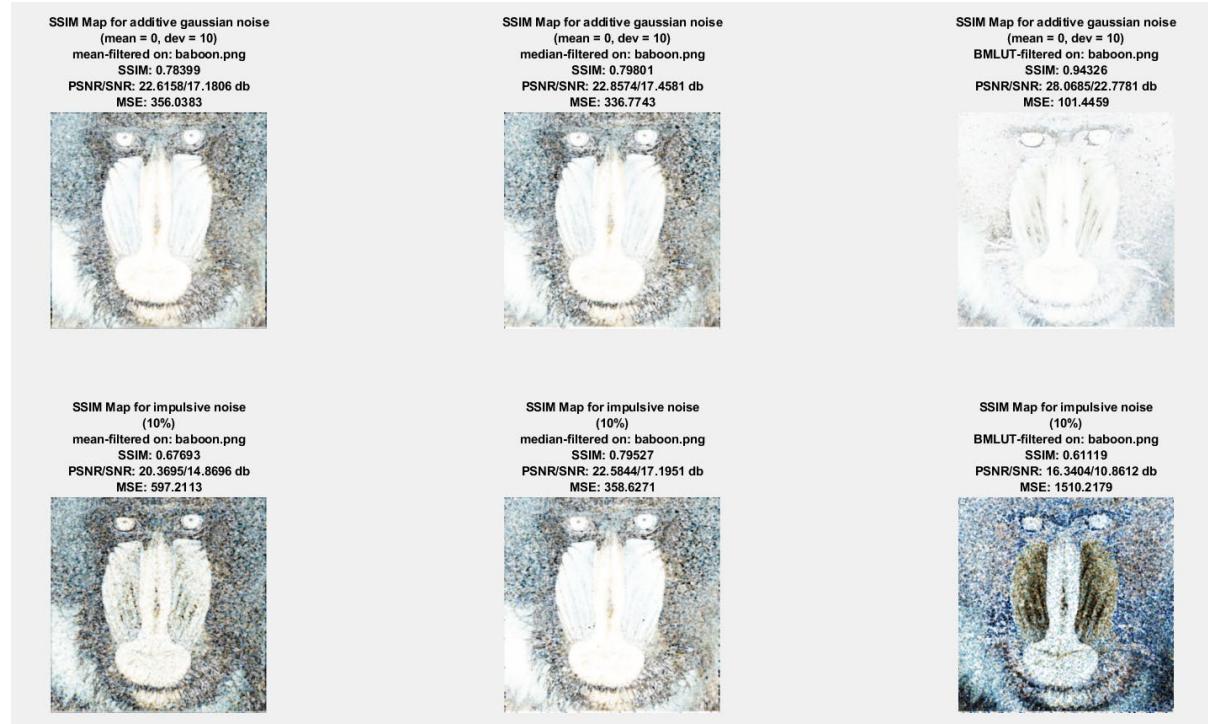


airplane.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.0024	0.88	29.21/26.49	77.87	
	Median	0.0021	0.865	31.58/28.89	45.10	
	BMLUT	1.66	0.66	28.04/25.36	101.99	
Impulsive (10%)	Mean	0.0021	0.50	22.64/19.66	353.42	
	Median	0.0018	0.94	31.58/28.88	45.11	
	BMLUT	25.85	0.229	13.82/10.70	2693.80	

În cazul imaginii cu avionul, filtrarea BMLUT merge cel mai bine pe tipul de zgomot aditiv gaussian iar pentru zgomotul impulsiv această filtrare “ascunde” majoritatea pixelilor zgomoți care apar pe avion dar fundalul este puțin schimbat. Mărimile SSIM și PSNR pentru filtrarea zgomotului gaussian sunt acceptabile ca mărimi de sine stătătoare și ca mărimi comparate cu celelalte tipuri de filtre care par să se descurce mai bine. În cazul filtrării zgomotului impulsiv, diferențele de la ieșirea acestuia sunt minime și se pot vedea atât în calitatea imaginilor de ieșire cât și a mărimilor de comparație SSIM și MSE.

Trăistaru Vlad-Viorel
411-TAID
Proiect PAIC, numărul 15
baboon.png

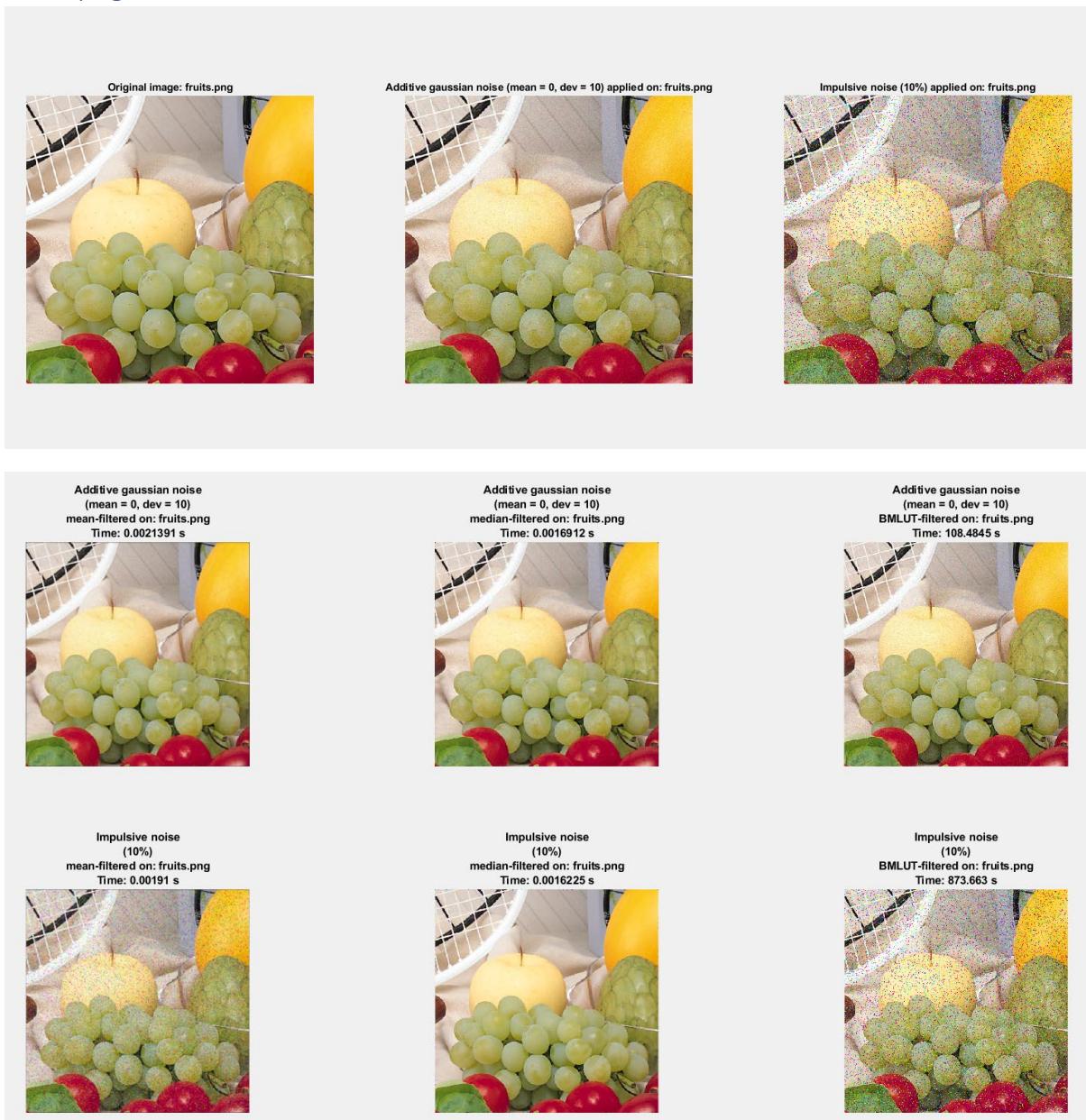


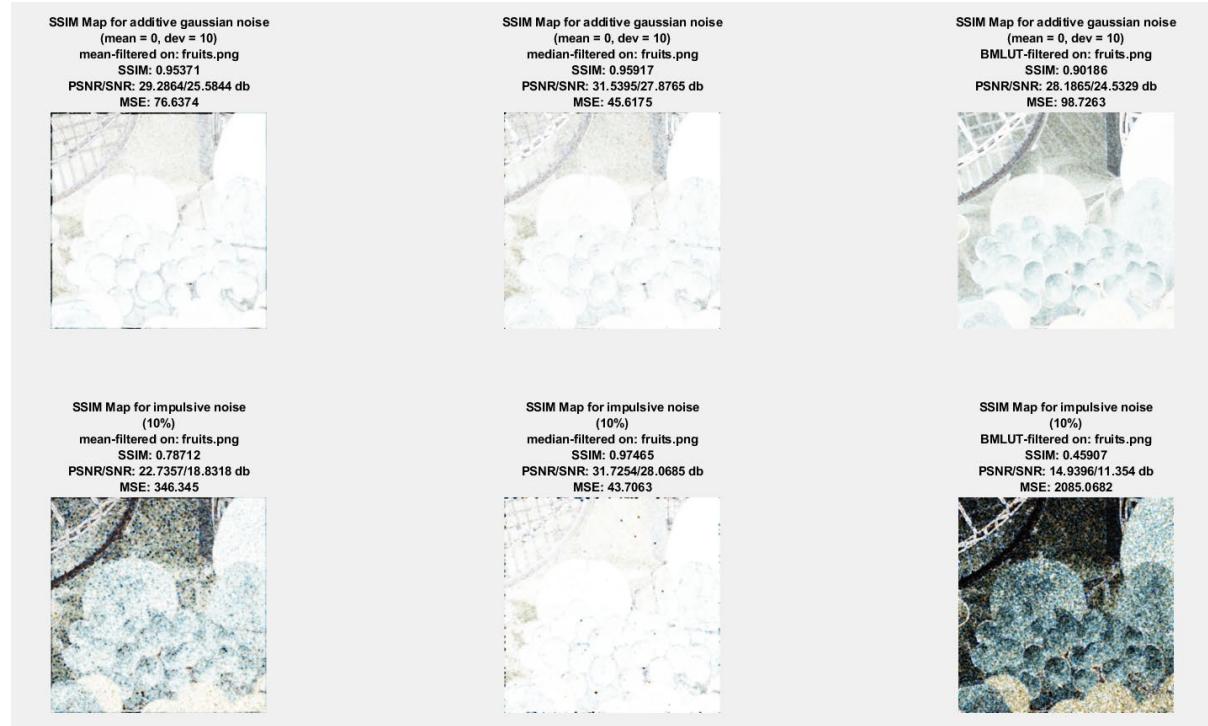


baboon.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.0025	0.783	22.61/17.18	356.03	
	Median	0.0024	0.798	22.85/17.45	336.77	
	BMLUT	1.42	0.943	28.06/22.77	101.44	
Impulsive (10%)	Mean	0.0021	0.676	20.36/14.86	597.21	
	Median	0.0017	0.795	22.58/17.19	358.62	
	BMLUT	18.724	0.611	16.34/10.86	1510.21	

Pentru imaginea cu babuinul, zgomotul gaussian este filtrat cel mai bine folosind BMLUT (valori de SSIM, PSNR, MSE foarte bune) și este foarte apropiat în calitate ca imaginea originală. În schimb pentru zgomotul impulsiv, metricele obiective arată rezultate mai puțin de dorit pentru un timp de executare mai mare. Deși pentru BMLUT, imaginea filtrată cu această metodă cu zgomot impulsiv arată totuși mai bine decât imaginea cu zgomot, nu se poate vorbi despre o calitate superioară a imaginii filtrate în acest caz.

Trăistaru Vlad-Viorel
411-TAID
Proiect PAIC, numărul 15
[fruits.png](#)

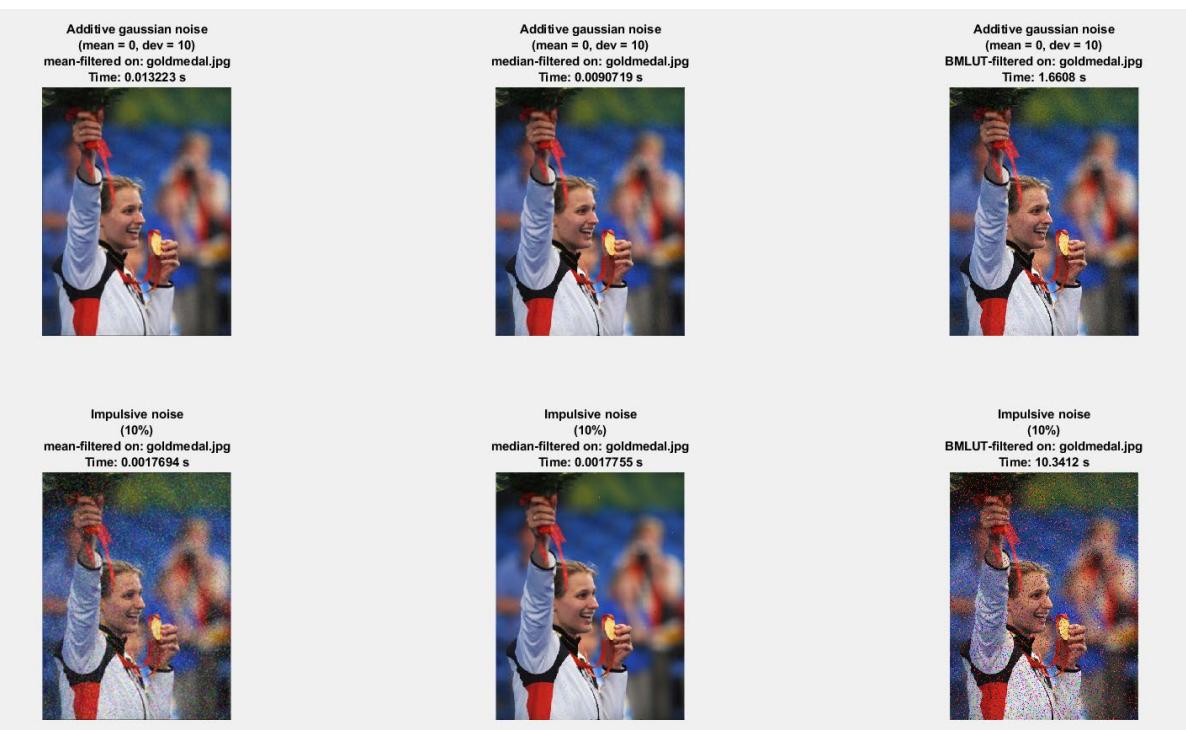
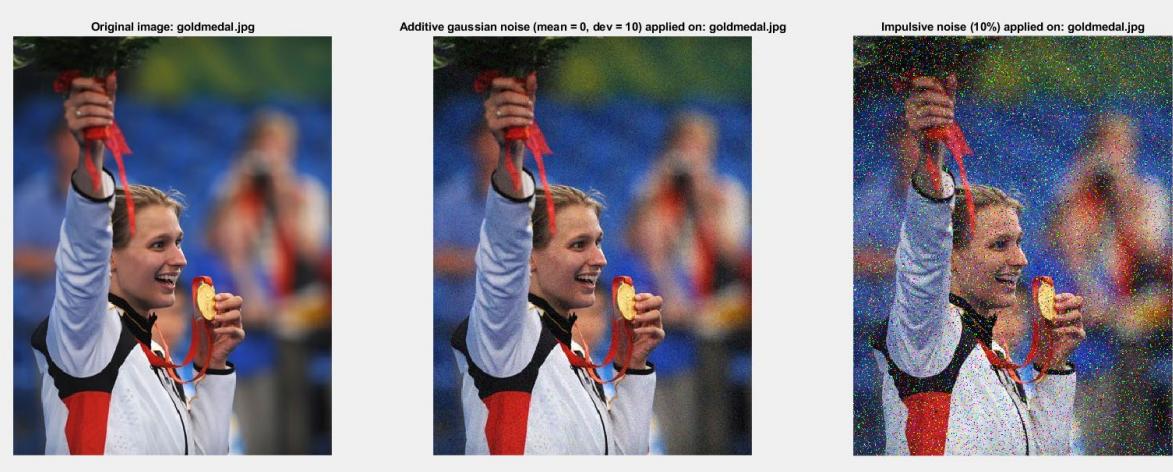




fruits.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.0021	0.953	29.28/25.58	76.63	
	Median	0.0016	0.959	31.53/27.87	45.61	
	BMLUT	108.48	0.901	28.18/24.53	98.72	
Impulsive (10%)	Mean	0.0019	0.787	22.73/18.83	346.34	
	Median	0.0016	0.974	31.72/28.06	43.70	
	BMLUT	873.66	0.459	14.93/11.35	2085.06	

În cazul amestecului de fructe, implementarea de BMLUT a detectat foarte mulți pixeli zgomotoși care algoritmul necesită ca aceștia să fie eliminați și prin urmare timpul de rulare a filtrului este unul foarte mare. Deși calitatea și metricile bune pentru zgomotul gaussian sunt favorabile în acest caz, pentru zgomotul impulsiv, un număr mic de pixeli zgomotoși au fost filtrați iar imaginea la ieșirea filtrului este schimbată minimal.

Trăistaru Vlad-Viorel
411-TAID
Proiect PAIC, numărul 15
goldmedal.jpg

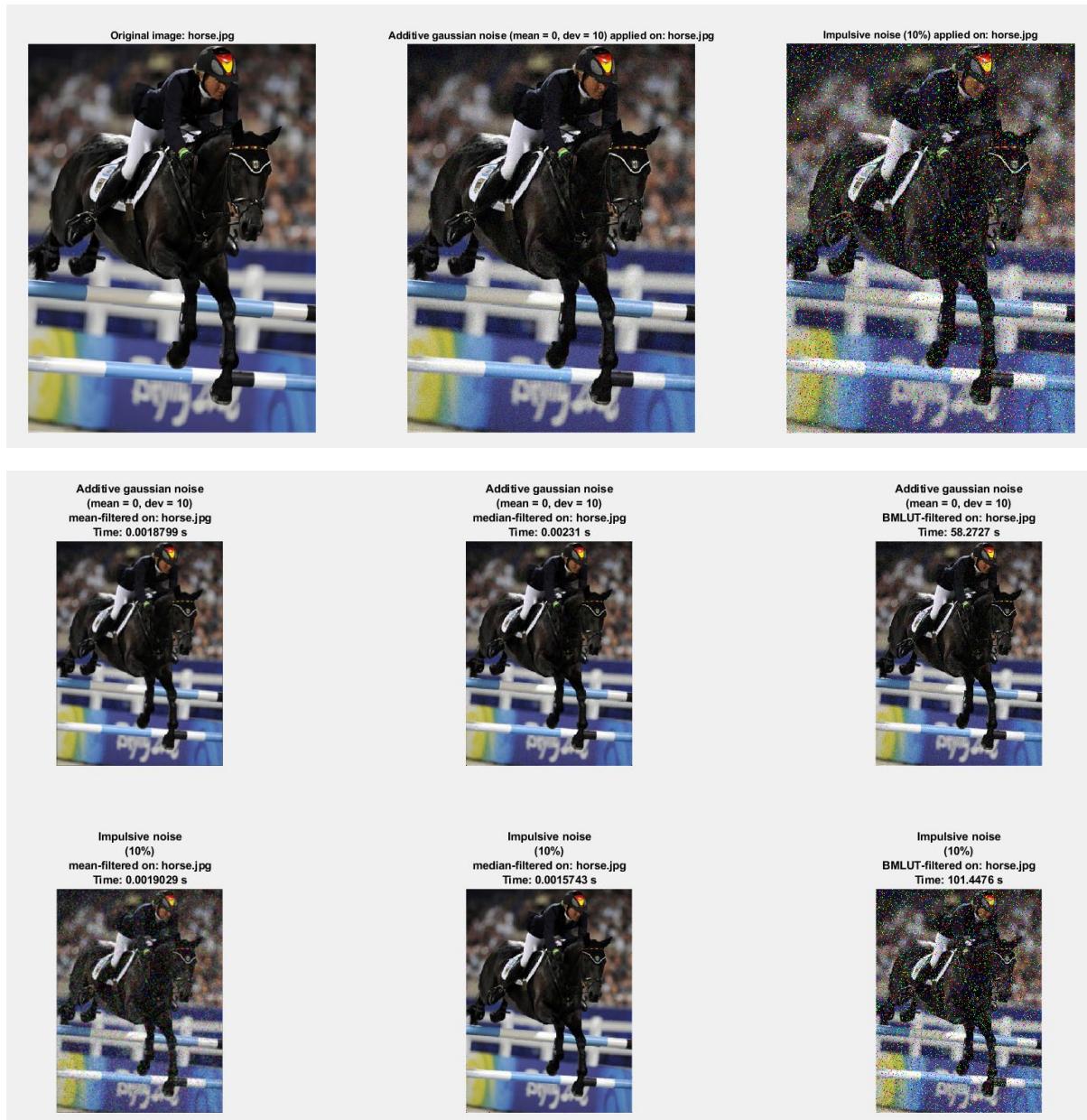


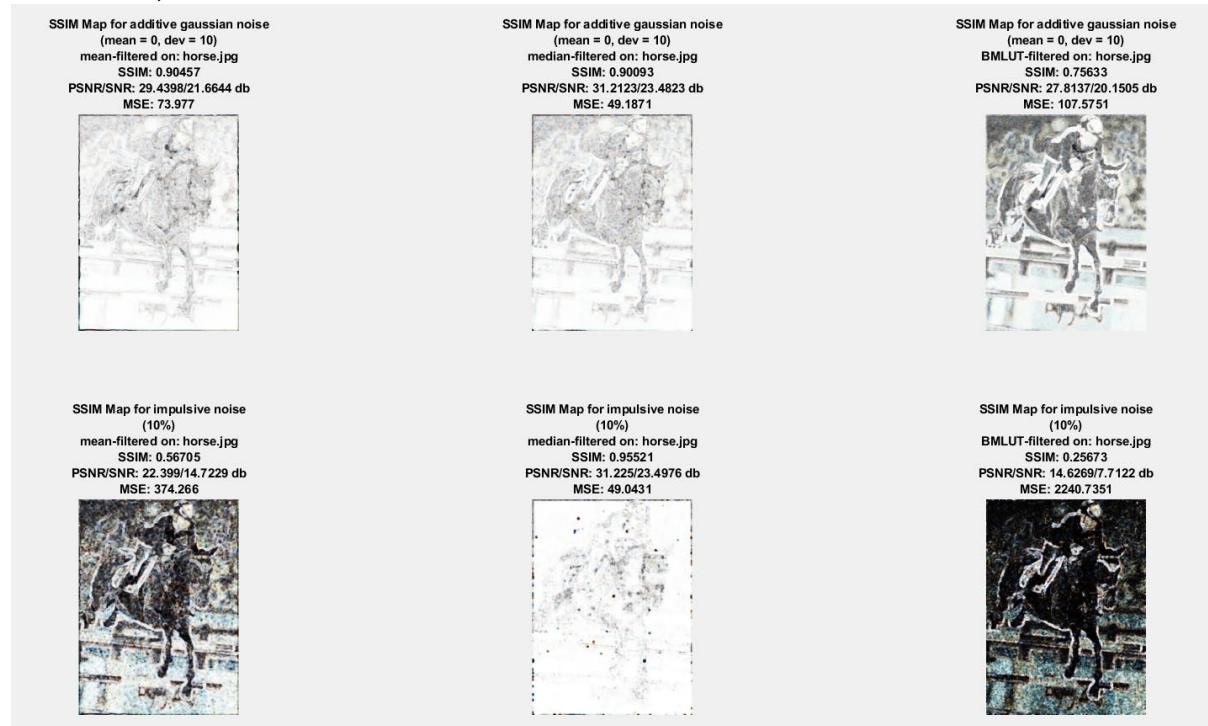


goldmedal.jpg						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.013	0.947	30.85/23.76	53.39	
	Median	0.009	0.944	32.51/25.44	36.48	
	BMLUT	1.66	0.832	27.83/20.81	107.15	
Impulsive (10%)	Mean	0.0017	0.705	23.18/16.15	312.31	
	Median	0.0017	0.978	33.27/26.20	30.59	
	BMLUT	10.34	0.456	17/9.92	1295.74	

În cazul acestei poze, ca și pentru celealte imagini, imaginile cu zgomot gaussian sunt filtrate cu BMLUT într-un mod favorabil dar același lucru nu se poate spune și despre imaginile filtrate cu zgomot impulsiv. De menționat ar fi faptul că majoritatea pixelilor verzi din imaginea cu zgomot impulsiv sunt eliminați, dat prin urmare abundența vizibilă de pixeli roșii la ieșirea filtrului BMLUT.

Trăistaru Vlad-Viorel
411-TAID
Proiect PAIC, numărul 15
horse.jpg





horse.jpg						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.0018	0.904	29.43/21.66	73.977	
	Median	0.0023	0.900	31.22/23.48	49.18	
	BMLUT	58.27	0.756	27.81/20.15	107.57	
Impulsive (10%)	Mean	0.0019	0.567	22.39/14.72	374.26	
	Median	0.0015	0.955	31.22/23.49	49.04	
	BMLUT	101.44	0.256	14.62/7.7	2240.73	

Pentru imaginea cu calul sărind, filtrările BMLUT pentru ambele tipuri de zgomot lasă de dorit din punct de vedere al calității comparative cu celelalte filtre dar și din punctul de vedere al pixelilor zgomotoși care par că doar o parte dintre ei sunt eliberați cu succes.

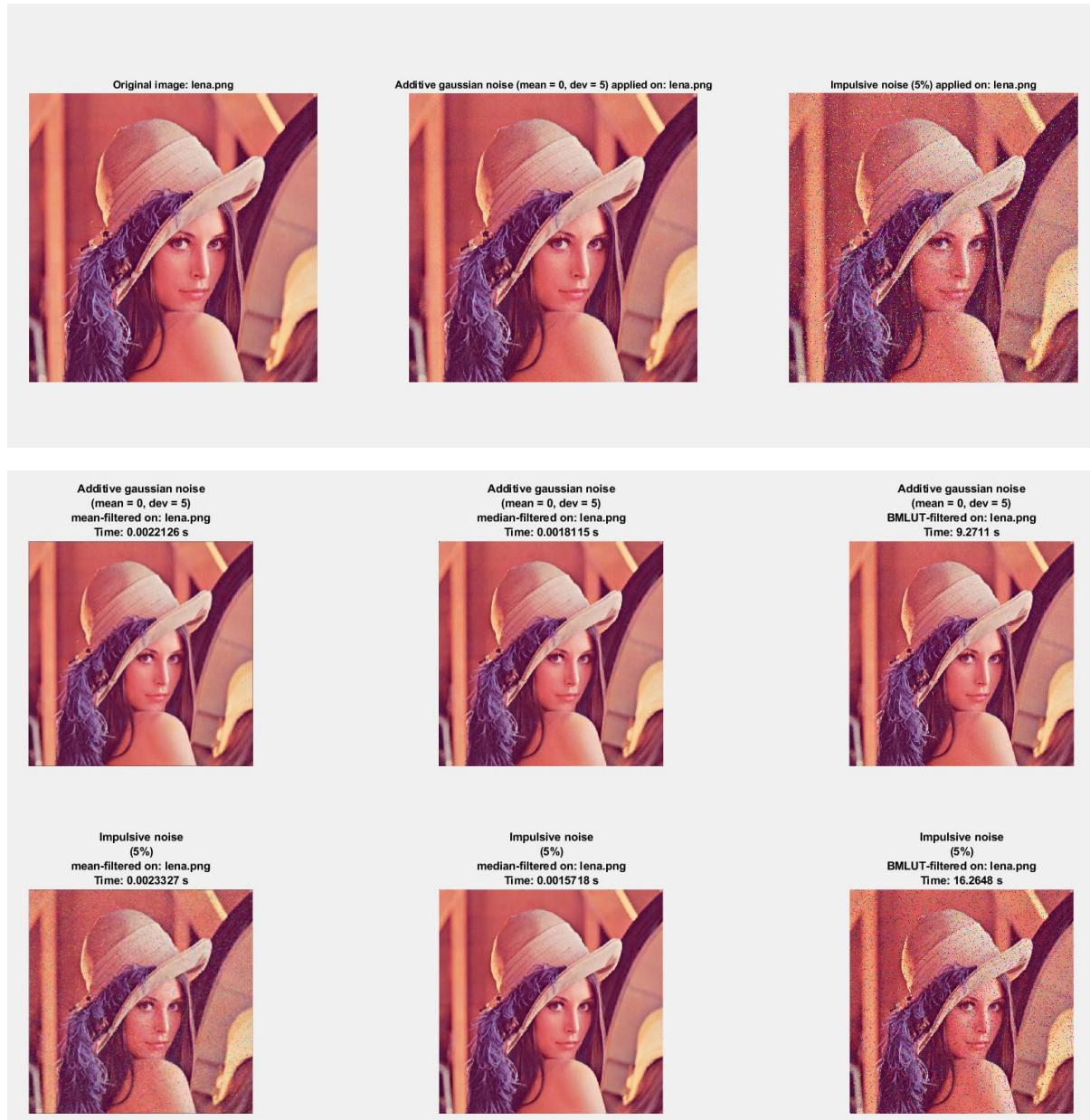
Trăistaru Vlad-Viorel

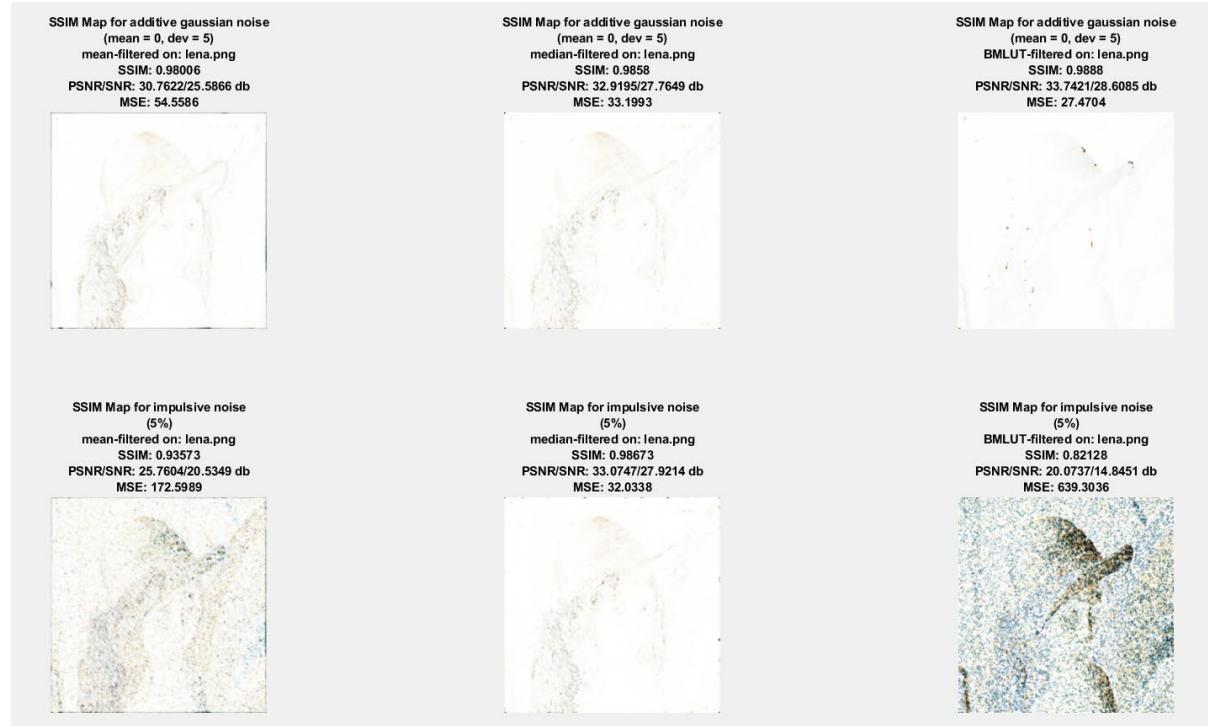
411-TAID

Proiect PAIC, numărul 15

lena.png (2 tipuri de zgomot * 3 intensități diferite)

gaussian cu mean = 0, sigma =5; impulsiv 5%





lena.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 5)	Mean	0.0022	0.98	30.76/25.58	54.55	
	Median	0.0018	0.985	32.91/27.76	33.19	
	BMLUT	9.27	0.988	33.74/28.60	27.47	
Impulsive (5%)	Mean	0.0023	0.935	25.76/20.53	172.59	
	Median	0.0015	0.986	33.07/27.92	32.03	
	BMLUT	16.26	0.821	20.07/14.84	639.30	

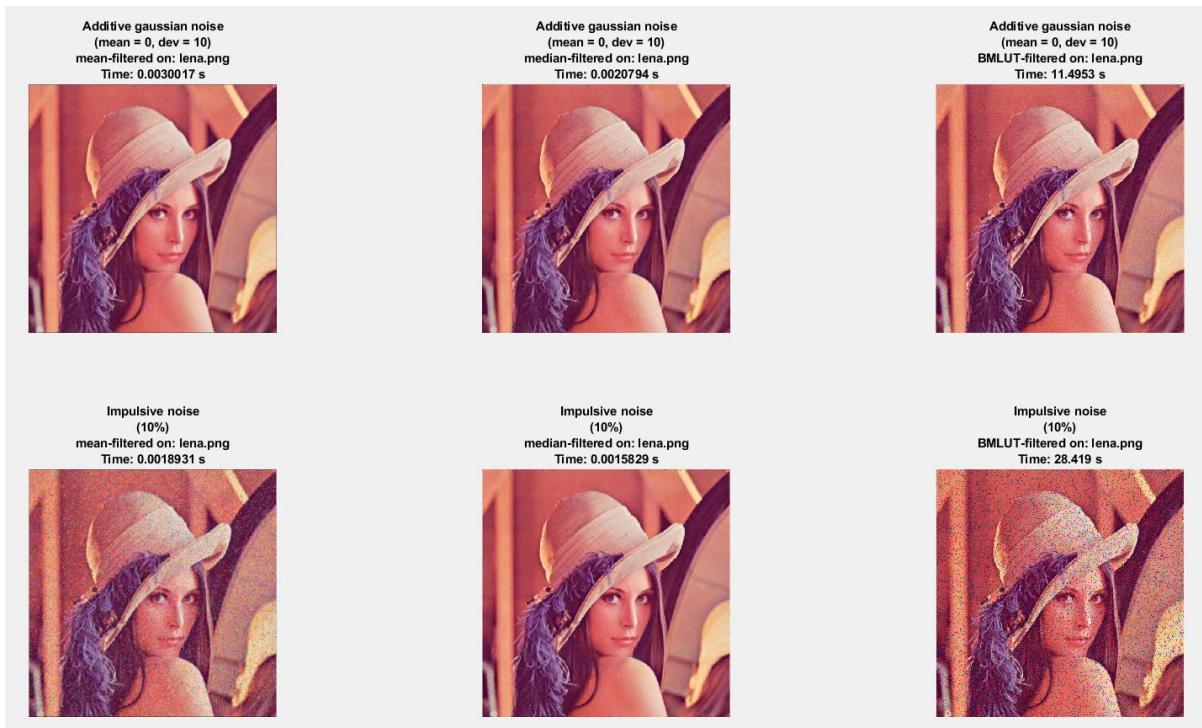
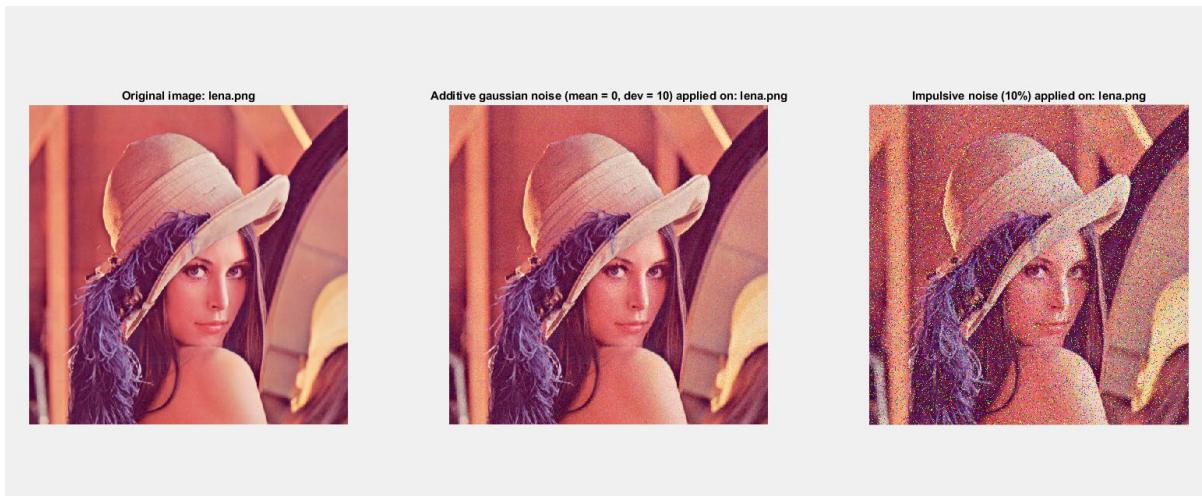
Comentariile sunt făcute pentru toate tabelele unde intensitățile variază pentru o anumită imagine, iar acele comentarii se pot găsi la ultimul tabel pentru imaginea respectivă.

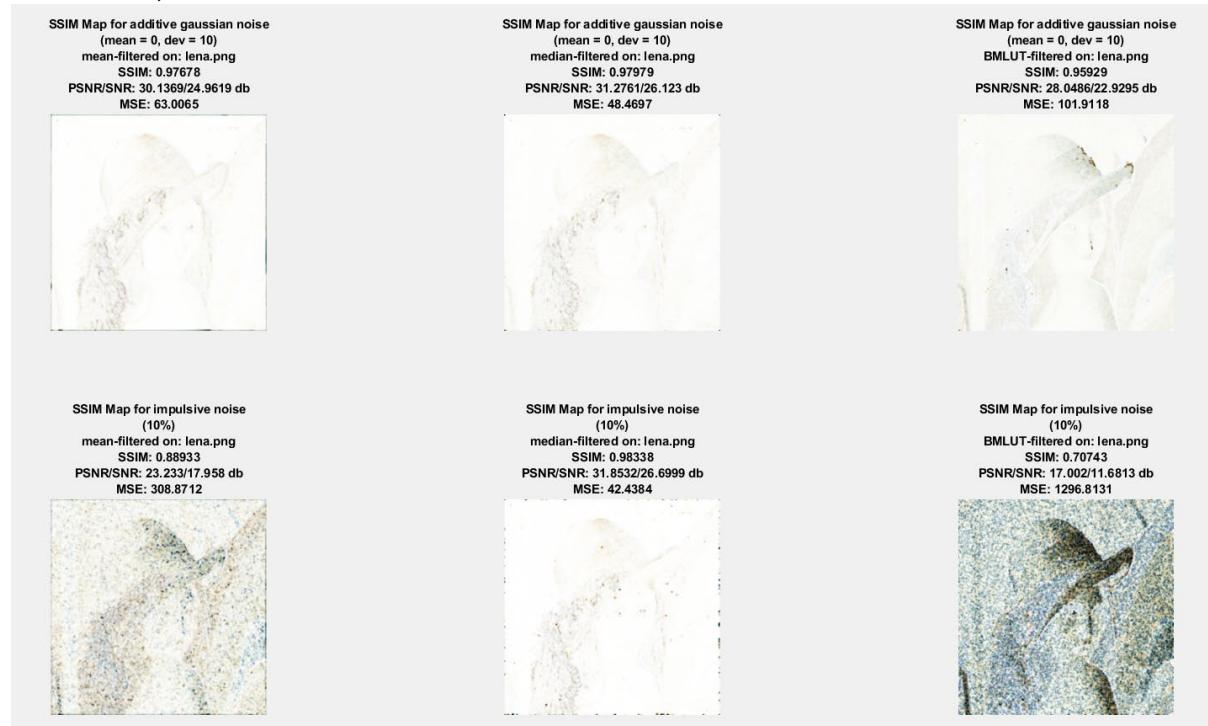
Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

gaussian cu mean = 0, sigma =10; impulsiv 10%





lena.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.003	0.976	30.13/24.96	63	
	Median	0.002	0.979	31.27/26.12	48.46	
	BMLUT	11.49	0.959	28.04/22.92	101.91	
Impulsive (10%)	Mean	0.0018	0.889	23.23/17.95	308.87	
	Median	0.0015	0.983	31.85/26.69	42.43	
	BMLUT	28.49	0.707	17/11.68	1296.81	

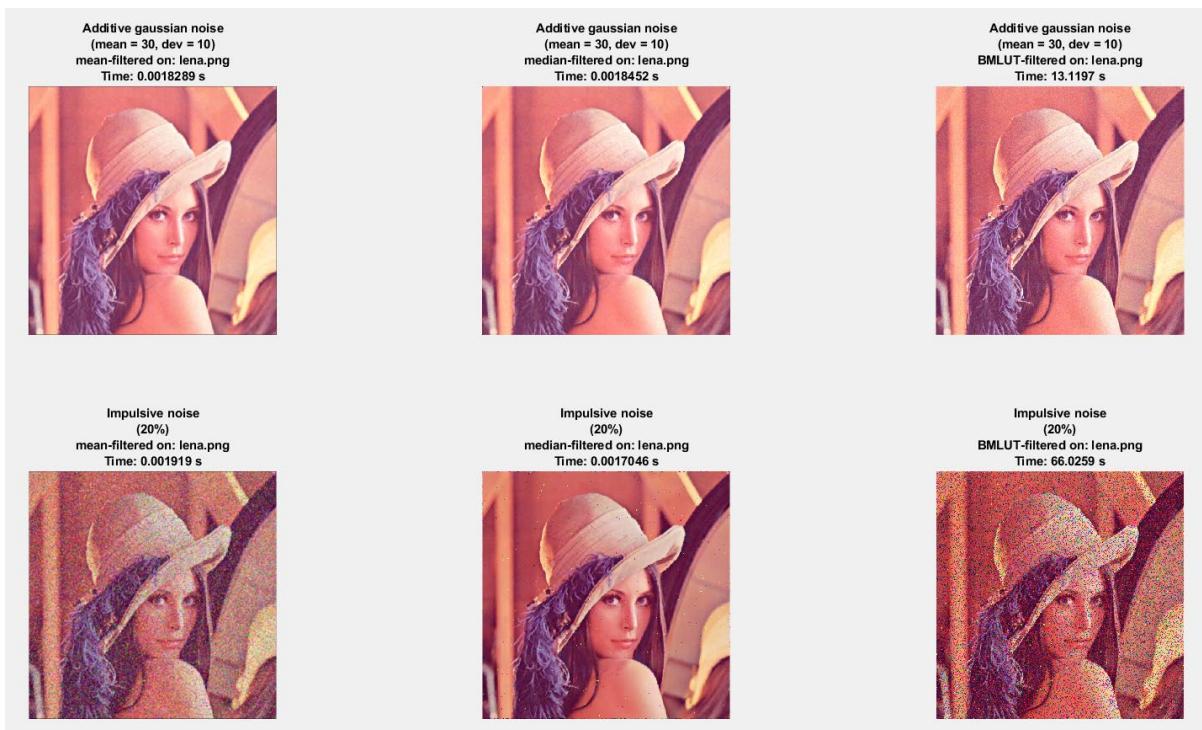
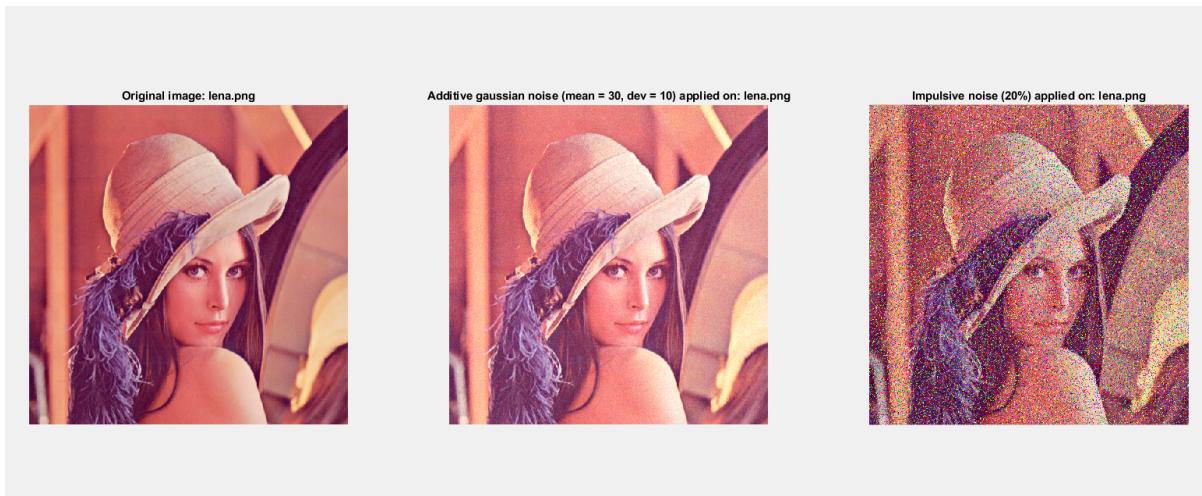
Comentariile sunt făcute pentru toate tabelele unde intensitățile variază pentru o anumită imagine, iar acele comentarii se pot găsi la ultimul tabel pentru imaginea respectivă.

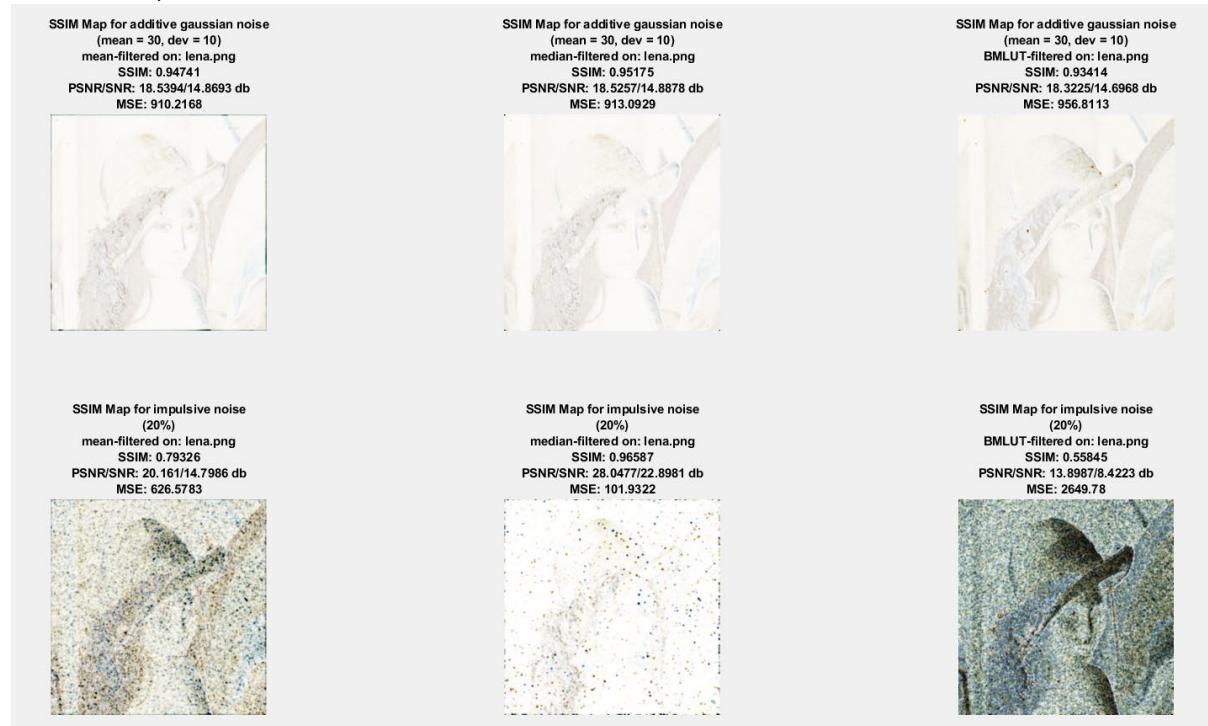
Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

gaussian cu mean = 30, sigma =10; impulsiv 20%





lena.png						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 30, sigma = 10)	Mean	0.0018	0.947	18.53/14.86	910.21	
	Median	0.0018	0.951	18.52/14.88	913.09	
	BMLUT	13.11	0.934	18.32/14.69	956.81	
Impulsive (20%)	Mean	0.0019	0.793	20.16/14.79	626.57	
	Median	0.0017	0.965	28.04/22.89	101.93	
	BMLUT	66.02	0.558	13.89/8.42	2649.78	

În cazul imaginii lena, pentru valori mici ale mediei, deviației pentru zgomotul gaussian și ale probabilității pentru zgomotul impulsiv, filtrul BMLUT se descurcă mai bine în a elimina numărul mai mic de pixeli zgomotoși deși în comparație cu celelalte filtre, rezultatele vizuale și obiective nu sunt tocmai favorabile.

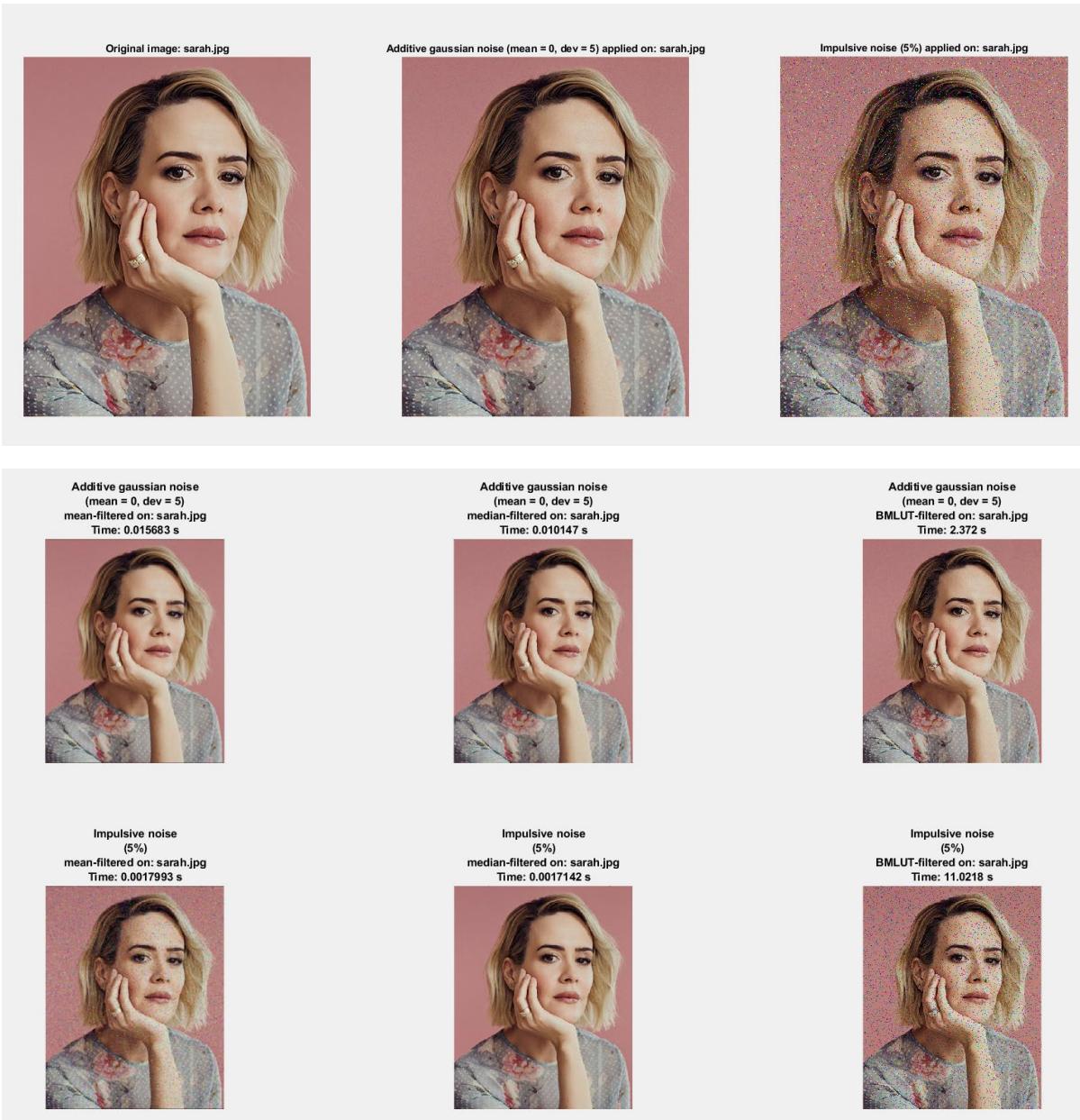
Trăistaru Vlad-Viorel

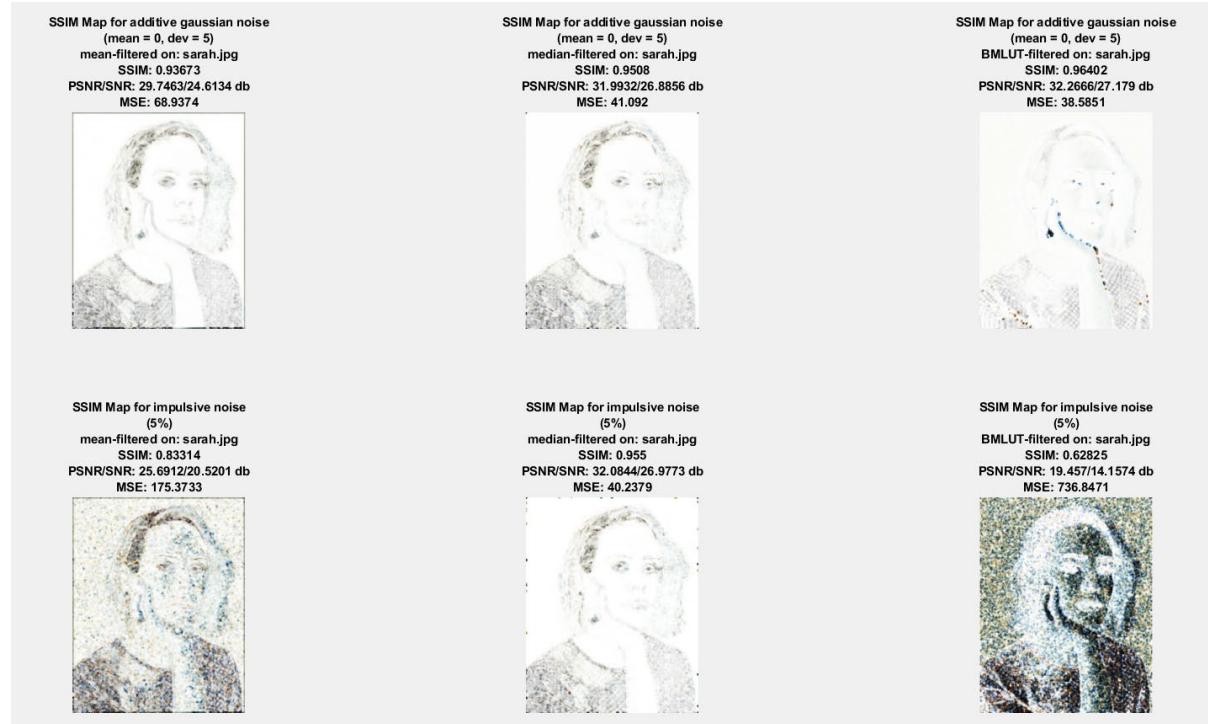
411-TAID

Proiect PAIC, numărul 15

sarah.jpg (2 tipuri de zgomot * 3 intensități diferite)

gaussian cu mean = 0, sigma =5; impulsiv 5%





sarah.jpg						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 5)	Mean	0.015	0.93	29.74/24.61	68.93	
	Median	0.010	0.95	31.99/26.88	41.09	
	BMLUT	2.37	0.96	32.26/27.17	38.58	
Impulsive (5%)	Mean	0.0017	0.83	25.69/20.52	175.37	
	Median	0.0017	0.95	32.08/26.97	40.23	
	BMLUT	11.02	0.62	19.45/14.15	736.84	

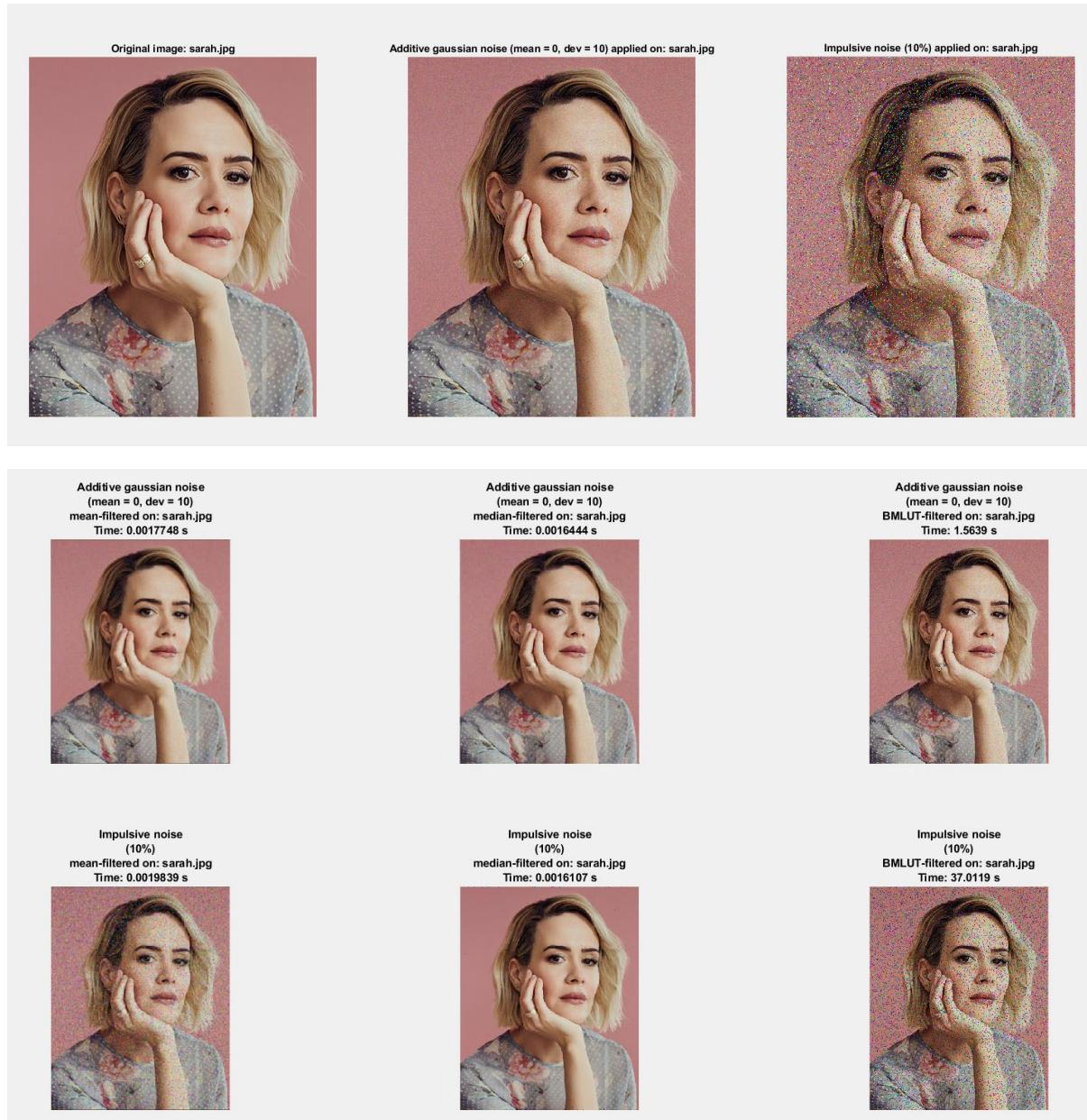
Comentariile sunt făcute pentru toate tabelele unde intensitățile variază pentru o anumită imagine, iar acele comentarii se pot găsi la ultimul tabel pentru imaginea respectivă.

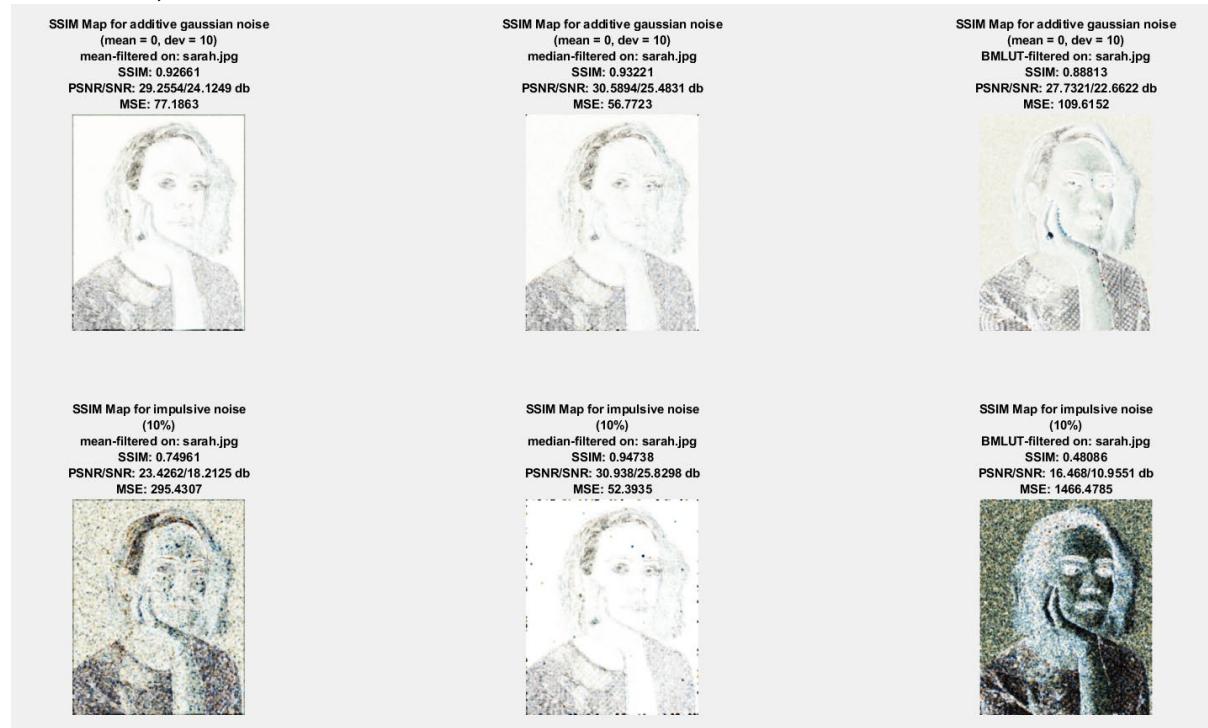
Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

gaussian cu mean = 0, sigma =10; impulsiv 10%





sarah.jpg						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 0, sigma = 10)	Mean	0.0017	0.926	29.25/24.12	77.18	
	Median	0.0016	0.932	30.58/25.48	56.77	
	BMLUT	1.56	0.888	27.73/22.66	109.61	
Impulsive (10%)	Mean	0.0019	0.749	23.42/18.21	295.43	
	Median	0.0016	0.947	30.93/25.82	52.39	
	BMLUT	37.01	0.48	16.46/10.95	1466.47	

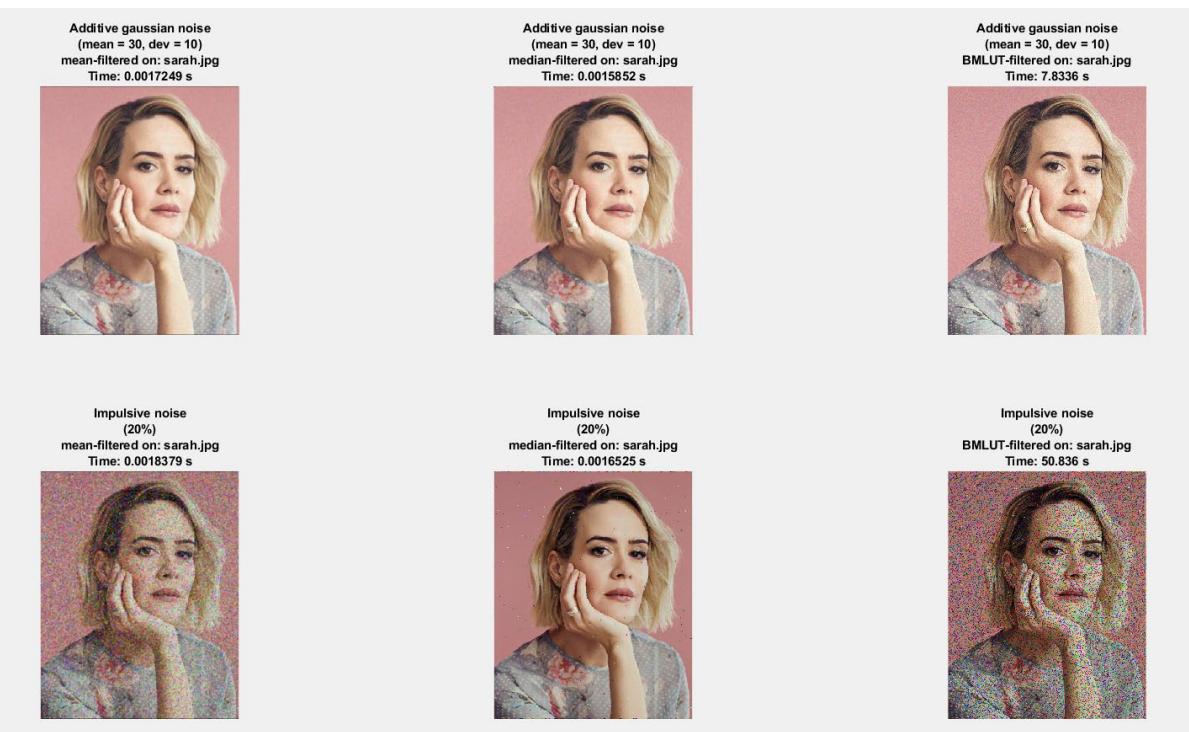
Comentariile sunt făcute pentru toate tabelele unde intensitățile variază pentru o anumită imagine, iar acele comentarii se pot găsi la ultimul tabel pentru imaginea respectivă.

Trăistaru Vlad-Viorel

411-TAID

Proiect PAIC, numărul 15

gaussian cu mean = 30, sigma =10; impulsiv 20%





sarah.jpg						
Noise Type	Filter Type	Filter completion time (s)	SSIM	PSNR/SNR (db)	MSE	
Gaussian (mean = 30, sigma = 10)	Mean	0.0017	0.89	18.33/14.80	954.92	
	Median	0.0015	0.90	18.34/14.84	951.71	
	BMLUT	7.83	0.86	18.12/14.64	1001.19	
Impulsive (20%)	Mean	0.0018	0.61	20.60/15.32	566.24	
	Median	0.0016	0.91	27.56/22.45	113.98	
	BMLUT	50.83	0.32	13.40/7.48	2967.14	

Pentru cazul imaginii sarah, se poate face același argument ca și la imaginea lena, unde rezultatele filtrului BMLUT lasă de dorit iar celealte filtre se descurcă mai bine și mai rapid în filtrarea zgomotului.

Concluzii și comentarii personale

Articolul discutat în cadrul acestui studiu descrie o metodă eficientă de a filtra zgomotul impulsiv găsit într-o imagine cu astfel de zgomot.

Detectia pixelilor zgomotoși după anumite criterii și eliminarea acestora bazată pe prezența lor într-un tabel de căutare cu blocuri de potrivire aduce un nou mod rapid de a parcurge și de a procesa o imagine la nevoie.

În cazul implementării din acest studiu, rezultatele și performanțele afirmate de articol nu se regăsesc întocmai. Această implementare nu integrează pasul de împărțire a blocurilor tintă și zgomotoase și parcurge imaginea de mai multe ori în căutarea unor informații esențiale pentru stergerea parțială sau uneori totală a zgomotului.

Chiar dacă rezultatele prezentate în acest document sunt pentru imagini color, iar rezultatele prezente în articol sunt pentru imagini gri, așteptarea ar fi ca performanțele să fie similare în timp ce timpul de rulare al filtrului să fie triplat pentru imagini RGB. Deși performanțele lasă de dorit, în cazul timpului de rulare se pare că pentru anumite imagini, acest timp este conform așteptărilor.

Deși metoda de filtrare poate fi transformată din pseudocod în cod cu ajutorul pașilor prezenți în articol, implementarea prezentată aici ar putea fi îmbunătățită prin optimizarea numărului de parcurgeri ale imaginii, prin detectarea mai bună a pixelilor zgomotoși sau prin sărirea anumitor pași de care nu mai sunt nevoie.

În general, performanța de eliminare a zgomotului pentru BMLUT scade cu cât avem mai mulți pixeli zgomotoși deoarece algoritmul, la un moment dat, nu mai poate face diferența dintre pixelii neafectați și cei afectați deoarece se bazează pe similaritatea pixelilor vecini.

Cu toate acestea, filtrul, în esență, încearcă să optimizeze căutarea pixelilor zgomotoși și a blocurilor care ajută la eliminarea acestora pentru a furniza o schemă rapidă utilizabilă în domeniul larg al prelucrării și analizei de imagini.

Bibliografie

- [1] G. Pok and K. H. Ryu, "Efficient Block Matching for Removing Impulse Noise," *IEEE Signal Processing Letters*, vol. 25, no. 8, pp. 1176-1180, 2018.
- [2] G. Pok, J.-C. Liu and A. Nair, "Selective removal of impulse noise based on homogeneity level information," *IEEE Transactions on Image Processing*, vol. 12, no. 1, pp. 85-92, 2003.
- [3] MathWorks, „Create gray-level co-occurrence matrix from image - MATLAB graycomatrix,” [Interactiv]. Available: <https://www.mathworks.com/help/images/ref/graycomatrix.html>.
- [4] P. Datta, „All about Structural Similarity Index (SSIM): Theory + Code in PyTorch | by Pranjal Datta | SRM MIC | Medium,” 3 September 2020. [Interactiv]. Available: <https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e>.
- [5] MathWorks, „Normal random numbers - MATLAB normrnd,” [Interactiv]. Available: <https://www.mathworks.com/help/stats/normrnd.html>.
- [6] MathWorks, „Add noise to image - MATLAB imnoise,” [Interactiv]. Available: <https://www.mathworks.com/help/images/ref/imnoise.html>.
- [7] MathWorks, „N-D filtering of multidimensional images - MATLAB imfilter,” [Interactiv]. Available: <https://www.mathworks.com/help/images/ref/imfilter.html>.
- [8] MathWorks, „2-D median filtering - MATLAB medfilt2,” [Interactiv]. Available: <https://www.mathworks.com/help/images/ref/medfilt2.html>.