Iterative Minkowski–Weighted Mean Filter: An Improvement of "An Iterative Mean Filter for Image Denoising [1]"

Panaj Abhavudhichai
Department of Electrical Engineering
Chulalongkorn University
Bangkok, Thailand
6471037521@stu.chula.ac.th

***Abstract*—This report is mainly studied the Iterative Mean Filter (IMF) algorithm in [1] and proposed an improved version by adopting the weighted mean idea that exploits the Minkowski distance from the center pixel (p=0.125) as the weighted window with a size of 3 × 3. The proposed filter still uses the constrained mean of the intensity of noise-free pixels in a fixed-size window to suppress the salt-and-pepper noise in grayscale images. The experiment was conducted with the same settings as in [1] except that the experiment used only the IMF method to compare with the proposed one and used the different random seed to generate the noisy images (the results can't be the same because we didn't know the initial random seed they used in [1]). In conclusion, the experimental results show that the proposed filter outperforms the IMF in almost all cases.**

Keywords*—Salt-and-pepper noise, image denoising, noise removal, image restoration, image processing, nonlinear filter.

I. Introduction

Normally, Salt-and-pepper noise (SPN) occurs during image acquisition, image transmission, or even image decoding processes. It's typically caused by random impulse interference (salt noise) and by equipment malfunctions (pepper noise) i.e., image acquisition sensor [2]. This kind of noise can interfere with the image analyses and cause a severe quality reduction because the noise pixels become maximum or minimum intensity values, in other words, they are completely lost information (like missing data pixels). Thus, SPN denoising is an important procedure to eliminate the noise in the image pre-processing task. Basically, the image denoising process aims to reconstruct the noise-corrupted image with the preservation of image structures (e.g., edges, details, and textures) which is essential for the accuracy and performance improvement in other post-processing tasks e.g., object segmentation, feature extraction, object recognition, and image classification [1].

The ordinary methods for removing SPN include but are not limited to median-based methods [3-6], total variation-based methods [7-9], or deep learning-based methods [10-12]. For the median-based methods, one of the most renowned methods is the classical Median Filter (MF) [13] which uses a window sliding technique with a fixed-size window and assigns the median of the intensity values (including SPN values which are maximum and minimum gray values) in that window to the center pixel. This MF works well with low-density SPN but failed to eliminate medium and high noise density. Subsequently, the notable Adaptive Median Filter (AMF) [3] was proposed to refine this issue by using the adaptive window size instead. The idea of AMF is basically to enlarge the window size until it becomes large enough to cover the noise particle (if the window size is smaller than the noise particle, the median will be the maximum or minimum intensity) or the maximum window size is reached. However, this procedure increases the computational complexity and reduces the accuracy of the filter. These drawbacks persist in the later adaptive window-based algorithm such as Adaptive Weighted Mean Filter (AWMF) [14] which use weighted mean instead of median.

In [1], they review some of the modern SPN denoising filters (median-based methods) and aim to bypass the adaptive window's drawbacks mentioned above but still effective on the high-density SPN removal simultaneously (many filters adopt the adaptive window's idea to suppress high-density SPN successfully, but it comes with the drawbacks cost: higher complexity and lower accuracy). Their work in [1] overcomes these drawbacks by using only 3 × 3 fixed window size and finding the mean of the intensity values of the noise-free pixels in the window to evaluate a new intensity value for the noisy center pixel. Then overcomes the fixed window's drawback (poor high-density SPN elimination) by combining the filter with an iterative process.

After we studied the IMF in [1], we found that the IMF works effectively from low to high densities of SPN, but we thought that the algorithm may be improved by changing the mean to other measures of central tendency/order statistics and the stopping criterion. As a result, we ended up using weighted mean instead. These weights come from the idea that the nearer pixels have more important and are more correlated to the center pixel than the farther pixels and we should put more weight on them. Our proposed improvement utilized the Minkowski distance with p=0.125 to generate the weights as it yields the best result in the simulation. Nevertheless, even if it can outperform the original IMF in the quality assessments sense, it still comes with a little higher computational cost.

II. Iterative Mean Filter

A. Definition and Notions

In this report, we use almost the same notions as in [1]. Let (i, j) be the coordinate of all pixel locations in the cartesian set $I = \{1, 2, \ldots, m\}\times\{1, 2, \ldots, n\}$ where m and n are the number of rows and columns in the image respectively and $[\delta_{min}, \delta_{max}]$ is the range of intensity values in the image (i.e., for an 8-bit grayscale image, $\delta_{min} = 0$ and $\delta_{max} = 255$) with the assumption that, in natural images, there are a very small number of noise-free pixels that reaching the boundary values δ_{min} and δ_{max}.

Definition 1: Let $U := [u_{ij}]_{m\times n}$ be a ground truth (noise-free) image where $\delta_{min} \le u_{ij} \le \delta_{max}$. and $B := [b_{ij}]_{m\times n}$ be a SPN corrupted (noisy) image of U with $p + q$ noise density $\in [0,1]$,

where p, $q \in [0,1]$ and $p = q$ when the value of the noise density is even. Then we have:

$$b_{ij} := \begin{cases} \delta_{min}, & with\ probability\ p \\ \delta_{max}, & with\ propability\ q \\ u_{ij}, & with\ probability\ 1-(p+q) \end{cases} \quad (1)$$

Definition 2: Let $A := [a_{ij}]_{m\times n}$ be an image, $W_{ij}(A,r)$ denotes a window of size $(2r+1) \times (2r+1)$ centered at (i,j) (r = 1, 2, …). the indices set of a $W_{ij}(A,r)$ is defined as:

$$\{(i^*,j^*) \in I : |i^*-i| \le r, |j^*-j| \le r\} \quad (2)$$

Definition 3: Let $W^*_{ij}(A,r)$ denotes a window $W_{ij}(A,r)$ that counts only noise-free pixel $a_{i^*j^*}$ (centered pixel a_{ij} can be noisy pixel). the strict indices set of a $W^*_{ij}(A,r)$ is defined as:

$$\{(i^*,j^*) \in W_{ij}(A,r) : a_{i^*j^*} \ne \delta_{min},\ a_{i^*j^*} \ne \delta_{max}\} \quad (3)$$

Definition 4: Let $\bar{W}^{mean}_{ij}(A,r)$ denotes a constrained mean (only noise-free pixels) of $W_{ij}(A,r)$ and defined as:

$$\begin{cases} a_{ij} & W^*_{ij}(A,r) = \varnothing \\ \dfrac{1}{card(W^*_{ij}(A,r))} \displaystyle\sum_{(i^*,j^*)\in W^*_{ij}(A,r)} & W^*_{ij}(A,r) \ne \varnothing \end{cases} \quad (4)$$

where $card(\cdot)$ denotes the set cardinality (number of pixels).

Definition 5: Let $A^* := [a^*_{ij}]_{m\times n}$ be another image. Define the ℓ_1-distance between 2 images A and A^* (equivalent to Sum of Absolute Differences: SAD) as follows:

$$|A-A^*|_1 := \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}-a^*_{ij}|, \quad (5)$$

where $|\cdot|$ denotes absolute value.

B. IMF Algorithm

Algorithm 1 Iterative Mean Filter (IMF) (Mode=1)

Input: A noisy image $B := [b_{ij}]_{m\times n}$
Output: A restored image $A := [a_{ij}]_{m\times n}$

Initialize $r := 1, k := 0, A^{[0]} := B, \varepsilon$.
Compute : $\delta_{max} := \max_{\substack{1\le i\le m\\ 1\le j\le n}} \{b_{ij}\}$, $\delta_{min} := \min_{\substack{1\le i\le m\\ 1\le j\le n}} \{b_{ij}\}$.
Repeat
  For each pixel (i,j) of an image $A^{[k]}$ at a step k
    If $\left(a^{[0]}_{ij} \ge \delta_{max} \,||\, a^{[0]}_{ij} \le \delta_{min}\right)$
      Define : $W^*_{ij}(A^{[k]}, r)$.
      Update : $a^{[k+1]}_{ij} := \bar{W}^{mean}_{ij}(A^{[k]}, r)$
    Else
      Assign : $a^{[k+1]}_{ij} := a^{[k]}_{ij}$
    End
  End
Until $|A^{[k+1]} - A^{[k]}|_1 \le \varepsilon$

Algorithm 2 Iterative Mean Filter (IMF) (Mode=2)

Input: A noisy image $B := [b_{ij}]_{m\times n}$
Output: A restored image $A := [a_{ij}]_{m\times n}$

Initialize $r := 1, k := 0, A^{[0]} := B$.
Compute : $\delta_{max} := \max_{\substack{1\le i\le m\\ 1\le j\le n}} \{b_{ij}\}$, $\delta_{min} := \min_{\substack{1\le i\le m\\ 1\le j\le n}} \{b_{ij}\}$.
Repeat
  For each pixel (i,j) of an image $A^{[k]}$ at a step k
    If $\left(a^{[0]}_{ij} \ge \delta_{max} \,||\, a^{[0]}_{ij} \le \delta_{min}\right)$
      Define : $W^*_{ij}(A^{[k]}, r)$.
      Update : $a^{[k+1]}_{ij} := \bar{W}^{mean}_{ij}(A^{[k]}, r)$
    Else
      Assign : $a^{[k+1]}_{ij} := a^{[k]}_{ij}$
    End
  End
Until $card(\{(i,j) \in A^{[k+1]} : a^{[k+1]}_{ij} = \delta_{min} \vee a^{[k+1]}_{ij} = \delta_{max}\}) = 0$

In [1], they point out that IMF exploits the advantage of MF (low complexity) by using a fixed-size window of 3 × 3 instead of an adaptive window. This comes with a drawback in a high-density SPN scheme (failed to restore if there are few/no noise-free pixels in that small window). So, they used the constrained mean (eq. (4)) instead of the median (inspired

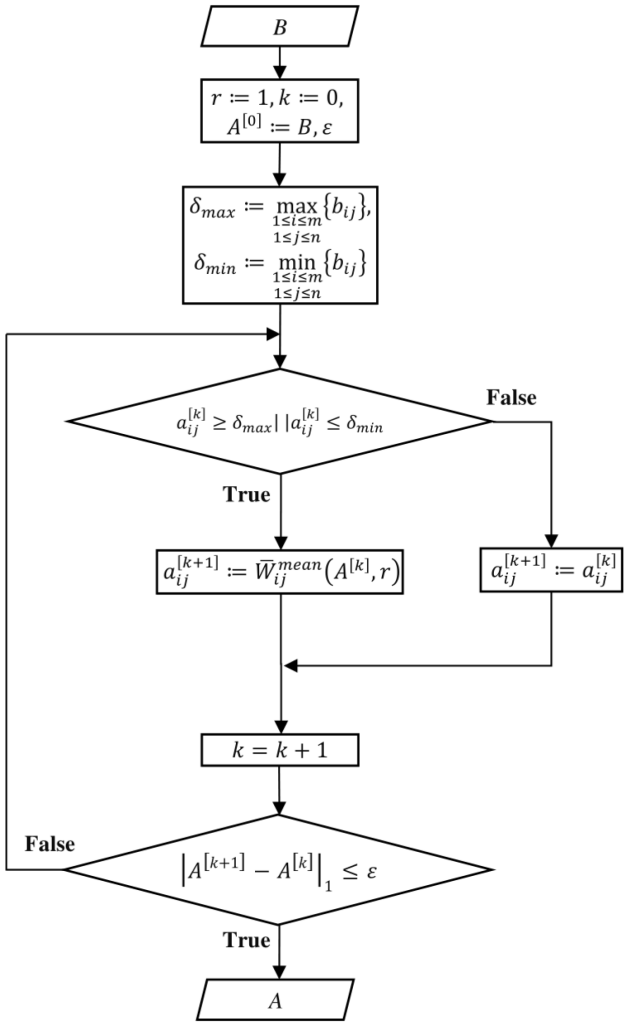

Fig.1. The flowchart of IMF (Mode=1).

by the improved accuracy of AWMF - using weighted mean that outperforms AMF - using median, especially in high-density cases, although both utilize the same noise detection technique). This constrained mean will also result in higher accuracy when evaluating the new intensity of the center pixel. But it's still not effective when dealing with high-density SPN. Therefore, they incorporate their algorithm with an iterative process to guarantee that all noisy pixels will be evaluated and replaced with the new intensity values.

The IMF algorithm from [1] is described in Algorithm 1 and its corresponding flowchart is depicted in Fig.1. Basically, for each pixel in every iteration, if the center pixel is a noisy pixel, it will be replaced with the constrained mean of the noise-free intensities of that pixel's window. On the contrary, the center pixel will remain the same intensity value as the last iteration if there are no noise-free pixels in the window or the center pixel itself is the noise-free pixel. that's mean IMF won't try to evaluate the noisy pixel immediately if it can't, unlike the AMF or AWMF that enlarge the window size until it can evaluate, IMF will try to evaluate those noisy pixels again in the next iteration instead – IMF won't have an issue of replacing with intensity value that's very far away from the center pixel (far away pixels shouldn't have much effect to the center pixel unless it'll cause artifacts). Hence, IMF can avoid the artifacts and achieve sharp edges. Lastly, the iterative process will be stopped if the SAD between restored images from 2 consecutive iterations is less than some small constant tolerance (i.e., $\varepsilon = 0$ if we consider intensity values in integer and maybe $\varepsilon = 10^{-6}$ if we consider it in real numbers).

One remaining drawback of IMF mentioned in [1] is from the same noise detection technique that's also used in MF, AMF, and AWMF which always assume δ_{min} and δ_{max} as a noisy pixel. This can cause a severe problem in the synthetic images (if they have a lot of noise-free pixels that are δ_{min} and δ_{max} – all of them will be considered noisy pixels). However, this is acceptable because it's the same drawback as many other nonlinear SPN filters and in typical applications, we emphasize only the natural images (normally, the number of noise-free pixels that are δ_{min} and δ_{max} is negligible).

However, after we followed Algorithm 1 with the same image datasets in MATLAB simulation, we found that the results are not quite close to the original paper's (particularly in the visual results). Then we found that the authors in [1] released their IMF program as the P-code file (an obfuscated, execute-only form of MATLAB – they didn't reveal their source code of IMF) here: https://github.com/uerkan80/IMF. But after some trial and error (by gradually changing the small size matrix input and then analyzing the output), we have completely reconstructed the algorithm that can reproduce the same results as their P-code file. For disambiguation, through this report, we'll call the original paper's IMF Algorithm 1 as IMF (Mode=1), IMF that we reconstructed as IMF (Mode=2), and IMF from their P-code file as IMF (Mode=3). The result comparisons between our reconstructed IMF (Mode=2) and the P-code file from the authors in [1] IMF (Mode=3) were shown in Fig.8.–9. and Fig.16.–17. in the appendix section.

There are only 2 different things between Algorithm 1: IMF (Mode=1) and our IMF (Mode=2) in Algorithm 2. The first thing is that, for every iteration, IMF (Mode=2) will compare each pixel (i, j) of image A from iteration $k = 0$ (instead of image A from iteration k in Algorithm 1) with δ_{min} and δ_{max} in the noise detection step. In other words, Algorithm 2 will repeat the constrained mean on every noisy pixel from the initial noisy image B for each iteration until the stopping criterion is reached (apply the constrained mean many times to the same position of the initial noisy pixels). Whereas Algorithm 1 will skip all noisy pixels that are already denoised from the previous iteration i.e., it'll apply the constrained mean only once on each initial noisy pixel. The second thing is that the SAD stopping criterion, which worked well with Algorithm 1, will have some convergence issues in Algorithm 2 as shown in Fig.18. – Fig.19. in the appendix section i.e., when there are no noisy pixels left in the image, the SAD from Algorithm 1 is exactly zero (because the denoised pixels from the previous iteration will remain intact, so, the difference between those 2 pixels is exactly zero). On the other hand, the difference between any 2 corresponding noisy pixels from any consecutive iterations in Algorithm 2 is harder to be exactly zero (because the denoised pixels changed every iteration and these differences may not be monotonic decreasing) as shown in Fig.19. that in some cases, even there are no noisy pixels left, the SAD value is still large (in fact, the SAD will grow as the image size grow – Mean of Absolute Differences: MAD criterion is better as it's not depended on the image size, but it still has a problem when selecting "best" tolerance value ε). We have already conducted the experiment on some datasets, and we can conclude that if we keep repeating Algorithm 2 after the number of noisy pixels in the image is already zero, it will degrade the image quality and the results will differ from IMF (Mode=3). We also found that if we stopped at the first iteration with zero number of noisy pixels, the results will be exactly the same as the results from IMF (Mode=3) in every image from all datasets in the experiment. As a result, we end up with $card(\{(i,j) \in A^{[k+1]} : a_{ij}^{[k+1]} = \delta_{min} \vee a_{ij}^{[k+1]} = \delta_{max}\}) = 0$ as the stopping criterion. This will guarantee that Algorithm 2 will be stopped when there are no noisy pixels left in the image. This stopping criterion is reasonable and practicable because if IMF assumes that there is no noise detection error in each window, it implies that all noisy pixel locations in the whole image (at every iteration) are known as well.

III. Proposed Filter

A. Our Experiments

We start an experiment based on the viewpoint that SPN is missing data and denoising is like missing data imputation in some sense. And we know that in statistics and data science fields, regular mean imputation (on 1-dimensional data), even though it's the most popular, is actually not good (it does not preserve the correlation among variables, and it'll reduce the variance of imputed variables which leads to standard errors shrinkage – most hypothesis tests and confidence interval will be unreliable). Therefore, we begin with a literature review on popular modern imputation techniques such as Multiple Imputation by Chained Equations (MICE) [15], MissForest [16], and k-Nearest-Neighbor (kNN) [17]. Unfortunately, we didn't find any suitable imputation method for our scenario, because fundamentally, an image is a 2-dimensional signal (it's only one simple variable with 2-dimensional data) and it doesn't applicable to most of those imputation techniques that are developed for 1-dimensional data or multiple variables. However, SPN denoising also resembles image interpolation in the sense that they both substituted missing intensities with information from nearby pixels and there is the well-known nearest-neighbor interpolation technique, thus, we adopt the

idea and tried the modified kNN imputation on our scenario with some distances that available in MATLAB. Unluckily, the results can't outperform the IMF (Mode=1) (but we've got some prospect ideas with distance function). Later on, we tried to adjust the IMF (Mode=1) algorithm by replacing the mean in eq. (4) to other measures of central tendency/order statistics e.g., geometric mean, harmonic mean, median, mode, and other medoids (we already did some experiments earlier and found that replacing median in MF/AMF with geometric median i.e., medoid with Euclidean distance will refine those filters significantly), but none of them can beat the arithmetic mean, especially when do it iteratively. We also tried the adaptive window's idea based on the variance of the intensities of noise-free pixels in the window integrated with an iterative procedure. And also tried to use the small, fixed window size in the early iteration but expand it in later iterations depending on the percentage of noisy pixels that are left in the image (fixed window in each iteration but the window size can be enlarged to give the pressure on the rate of convergence), we found that both trials yield better results than IMF (Mode=1) in some cases and the latter trial also converged faster / lower execution time (particularly in high-density SPN), but in overall cases, they still can't outperform the IMF (Mode=1) and far inferior to IMF (Mode=2).

Finally, we come up with an idea that the pixels that are closer distance to the center pixel have more relationships than the farther pixels and should have more importance when taken into consideration. We combine this idea with the idea of the weighted mean filter and formulate the weighted mean that puts more weight on the nearer pixels and less weight on the farther pixels, then use it as the constrained weighted mean instead of the mean in eq. (4). After we survey various distances in [18], we have selected the Minkowski distance due to its simplicity, adjustable p parameter, and can be changed into the renowned distances i.e., ℓ_∞ or Chebyshev distance (a special case of Minkowski distance when p $\to \infty$), ℓ_2 or Euclidean distance (p=2), and ℓ_1 or City block or Manhattan distance (p=1). The norm balls of various p of Minkowski distance are shown in Fig.2. and the general form of Minkowski distance is defined in Table 1 (3.) as follows:

Table 1. L_p Minkowski family	
1. Euclidean L_2	$d_{Euc} = \sqrt{\sum_{i=1}^{d} \lvert P_i - Q_i \rvert^2}$
2. City block L_1	$d_{CB} = \sum_{i=1}^{d} \lvert P_i - Q_i \rvert$
3. Minkowski L_p	$d_{Mk} = \sqrt[p]{\sum_{i=1}^{d} \lvert P_i - Q_i \rvert^p}$
4. Chebyshev L_∞	$d_{Cheb} = \max_i \lvert P_i - Q_i \rvert$

Table.1. Definition of ℓ_p distance in the Minkowski family.

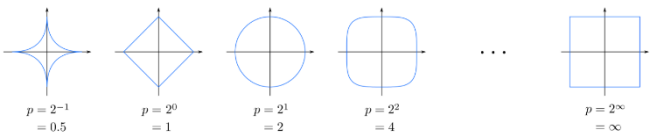

Fig.2. Unit norm balls of various p of Minkowski distance.

After varied p=0.0625, p=0.125, p=0.25, p=0.5, p=1, and p=2 at the generalized form of the Minkowski distance in our experiments, we found that p=0.125 yields the best results in all datasets/test scenarios, it makes our proposed Algorithm 3 outperforms both IMF(Mode=1) and IMF(Mode=2) in almost all cases (the lower p, the better overall results until p=0.125 that lower the p didn't give any significant improvement).

Noted that there are a lot more distances in [18] for further studies, we didn't compare all those distances with our method in our experiments yet (some of them look prospect and may yield better results than the Minkowski distance), since the Minkowski distance already achieve our goal to outperform the IMF. And we didn't improve the stopping criterion from the IMF (Mode=2) because, in our experiments, we found that it's better to stop the iterative process when there are no noisy pixels left in the image as mentioned earlier (we also develop some MAD criteria, but it turns out that it didn't work better than the current one).

As the result, we develop an improved version of IMF called Iterative Minkowski–Weighted Mean Filter (IMWMF) as described in Algorithm 3 (we use replication padding in Algorithm 1-3 as it makes IMF (Mode=2) ≡ IMF (Mode=3)).

B. IMWMF (Proposed) Algorithm

Algorithm 3 Iterative Minkowski–Weighted Mean Filter

Input: A noisy image $B := [b_{ij}]_{m \times n}$

Output: A restored image $A := [a_{ij}]_{m \times n}$

Initialize $r := 1, k := 0, A^{[0]} := B$, p:=0.125.

Compute : $\delta_{max} := \max_{\substack{1 \le i \le m \\ 1 \le j \le n}} \{b_{ij}\}$, $\delta_{min} := \min_{\substack{1 \le i \le m \\ 1 \le j \le n}} \{b_{ij}\}$.

Repeat

  For each pixel (i, j) of an image $A^{[k]}$ at a step k

    If $\left(a_{ij}^{[0]} \ge \delta_{max} \,||\, a_{ij}^{[0]} \le \delta_{min}\right)$

      Define : $W_{ij}^{*}\left(A^{[k]}, r\right)$.

      Update : $a_{ij}^{[k+1]} := \bar{W}_{ij}^{wmean}\left(A^{[k]}, r, \mathrm{p}\right)$

    Else

      Assign : $a_{ij}^{[k+1]} := a_{ij}^{[k]}$

    End

  End

Until $card(\{(i, j) \in A^{[k+1]} : a_{ij}^{[k+1]} = \delta_{min} \vee a_{ij}^{[k+1]} = \delta_{max}\}) = 0$

The IMWMF is simply followed Algorithm 2 except that it uses the constrained Minkowski–weighted mean instead of the constrained mean in eq. (4).

(1,1) (1,2) (1,3) (1,4) (1,5)
(2,1) (2,2) (2,3) (2,4) (2,5)
(3,1) (3,2) (3,3) (3,4) (3,5)
(4,1) (4,2) (4,3) (4,4) (4,5)
(5,1) (5,2) (5,3) (5,4) (5,5)

Lp distance (p=0.125) between (3,3) and (5,2)

$d_{Mk} = (|3\text{-}5|^{0.125} + |3\text{-}2|^{0.125})^{1/0.125}$

$\approx (1.0905 + 1)^8 \approx 364.7659$

Lp distance matrix $\boldsymbol{D}_{Mk}$ =

512.0000	364.7659	2.0000	364.7659	512.0000
364.7659	256.0000	1.0000	256.0000	364.7659
2.0000	1.0000	0	1.0000	2.0000
364.7659	256.0000	1.0000	256.0000	364.7659
512.0000	364.7659	2.0000	364.7659	512.0000

Minkowski Weighted Matrix $\boldsymbol{W}_{Mk} = 1./\boldsymbol{D}_{Mk}$ =

0.0020	0.0027	0.5000	0.0027	0.0020
0.0027	0.0039	1.0000	0.0039	0.0027
0.5000	1.0000	-	1.0000	0.5000
0.0027	0.0039	1.0000	0.0039	0.0027
0.0020	0.0027	0.5000	0.0027	0.0020

Fig.3. Example of 5×5 Minkowski Weighted Matrix calculation.

The Minkowski distance in our context means a distance between 2 coordinates of the center pixel (i,j) and nearby pixel (i^*,j^*) in any window as shown in Fig.3. However, we use only 3 × 3 fixed windows in Algorithm 3 and apply weights from the Minkowski Weighted Matrix only on the corresponding noise-free pixels in the window (the denominator is also calculated only from the corresponding weights that are used in the numerator). So, the constrained Minkowski–weighted mean used in Algorithm 3 can be formulated as follows:

$$\bar{W}_{ij}^{wmean}(A,r,p)=\begin{cases} a_{ij} & W_{ij}^{*}(A,r)=\varnothing \\ \dfrac{\sum_{(i^*,j^*)\in W_{ij}^{*}(A,r)} a_{i^*j^*}\Big/\sqrt[p]{|i-i^*|^p+|j-j^*|^p}}{\sum_{(i^*,j^*)\in W_{ij}^{*}(A,r)} 1\Big/\sqrt[p]{|i-i^*|^p+|j-j^*|^p}} & W_{ij}^{*}(A,r)\neq\varnothing \end{cases} \quad (6)$$

IV. Experimental Results

A. Image Quality Assessment Metrics

For image quality assessment after denoising, we also use the same error metrics that used in [1] i.e., Peak Signal-to-Noise Ratio (PSNR) [19], Structural Similarity (SSIM) [19], Visual Information Fidelity (VIF) [20], Image Enhancement Factor (IEF) [21], and Multiscale SSIM (MSSIM) [22]. The values of SSIM, VIF, and MSSIM are ranged in [0,1] and the higher PSNR, SSIM, VIF, IEF, or MSSIM indicates the better image quality.

PSNR is defined as follows:

$$PSNR(U,V) := 10\log_{10}\left(\frac{255^2}{MSE(U,V)}\right) \quad (7)$$

where Mean Square Error (MSE) is defined as follows:

$$MSE(U,V) := \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(u_{ij}-v_{ij})^2 \quad (8)$$

and U:=[u_{ij}] is a ground truth (noise-free) image, V:=[v_{ij}] is an evaluated (restored A:=[a_{ij}] or noisy B:=[b_{ij}]) image.

VIF is defined as follows:

$$VIF(U,V) := \frac{\sum_{j\in subbands} I\left(\vec{C}^{N,j};\vec{F}^{N,j}|s^{N,j}\right)}{\sum_{j\in subbands} I\left(\vec{C}^{N,j};\vec{E}^{N,j}|s^{N,j}\right)} \quad (9)$$

where

$$I\left(\vec{C}^{N,j};\vec{E}^{N,j}|s^{N,j}\right) := \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{M}\log_2\left(1+\frac{s_i^2\lambda_k}{\sigma_n^2}\right) \quad (10)$$

$$I\left(\vec{C}^{N,j};\vec{F}^{N,j}|s^{N,j}\right) := \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{M}\log_2\left(1+\frac{g_i^2 s_i^2\lambda_k}{\sigma_u^2+\sigma_n^2}\right) \quad (11)$$

eq. (10) and eq. (11) are the information that can be extracted (by the human brain) from a subband in the reference and the evaluated images (U and V) respectively. $\vec{E}^{N,j},\vec{F}^{N,j},\vec{C}^{N,j}$ are vectors of N components of the visual signal for the j-subband at the output of the Human Visual System (HVS) of U, the visual signal for the j-subband at the output of the HVS of V, and the random field from the j-subband in U respectively. σ_n^2,σ_u^2 are the variances of visual noise and Gaussian noise of the distortion model respectively. λ_k are covariance matrix's eigenvalues. g_i is deterministic scalar field. s^N is the maximum likelihood estimator of S^N, S^N is a random field vector size N of positive scalars, and $s^{N,j}$ is the j-subband of s^N.

IEF is defined as follows:

$$IEF(U,V,B) := \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(b_{ij}-u_{ij})^2}{\sum_{i=1}^{m}\sum_{j=1}^{n}(v_{ij}-u_{ij})^2} \quad (12)$$

where U, V, and B are ground truth, evaluated, and noisy images respectively.

SSIM and MSSIM are defined (respectively) as follows:

$$SSIM(U,V) := \frac{(2\mu_U\mu_V+C_1)+(2\sigma_{UV}+C_2)}{(\mu_U^2+\mu_V^2+C_1)+(\sigma_U^2+\sigma_V^2+C_2)} \quad (13)$$

$$MSSIM(U,V) := (l_M(U,V))^{\alpha M}\prod_{j=1}^{M}(c_j(U,V))^{\beta_j} \times (s_j(U,V))^{\gamma_j} \quad (14)$$

where

$$l_M(U,V) := \frac{2\mu_U\mu_V+C_1}{\mu_U^2+\mu_V^2+C_1} \text{ on scale } M,$$

$$c_j(U,V) := \frac{2\sigma_U\sigma_V+C_2}{\sigma_U^2+\sigma_V^2+C_2} \text{ on each scale } \quad j=1,..,M,$$

$$s_j(U,V) := \frac{\sigma_{UV}+C_3}{\sigma_U\sigma_V+C_3} \text{ on each scale } \quad j=1,\ldots,M,$$

μ_U, μ_V, σ_U, σ_V, and σ_{UV} are the intensity means, standard deviations, and cross-covariance of U and V respectively. $C_1 := (K_1L)^2$, $C_2 := (K_2L)^2$ are constants ($K_1 := 0.01$, $K_2 := 0.03$, $L := 255$ for 8-bit grayscale images). j is a resolution scale after low-pass filtering and downsampling, and M is the total number of scales (in the experiment we use M=5). α, β_j, γ_j are tuning parameters for adjusting the relative importance of different components.

B. Datasets and Test Cases

We conducted the experiments on MATLAB R2021a by using 18 traditional images (size 512 × 512 pixels): Baboon, Barbara, Blonde Woman, Boat, Bridge, Cameraman, Dark-Haired Woman, Einstein, Elaine, Flintstones, Hill, House, Lake, Lena, Living Room, Peppers, Pirate, and Plane (note that in [1], they used 20 images including Flower and Parrot, but we didn't sure which images are the mentioned images, thus, we only used 18 images instead), 40 images (size 600 × 600 pixels) from the TESTIMAGES dataset [23], and 200 images (size 481 × 321 or 321 × 481 pixels) from the BSDS dataset (https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/html/dataset/images.html) of the UC Berkeley. All images are in 8-bit grayscale format.

There are 3 test cases that we are taking into consideration 1) evaluate visual image quality for 3 selected images, 2) evaluate assessment metrics for 3 image datasets mention above, 3) evaluate execution time (we tried to consider the total number of iterations as well). For each case, we compare our proposed method with IMF(Mode=1) and IMF(Mode=2).

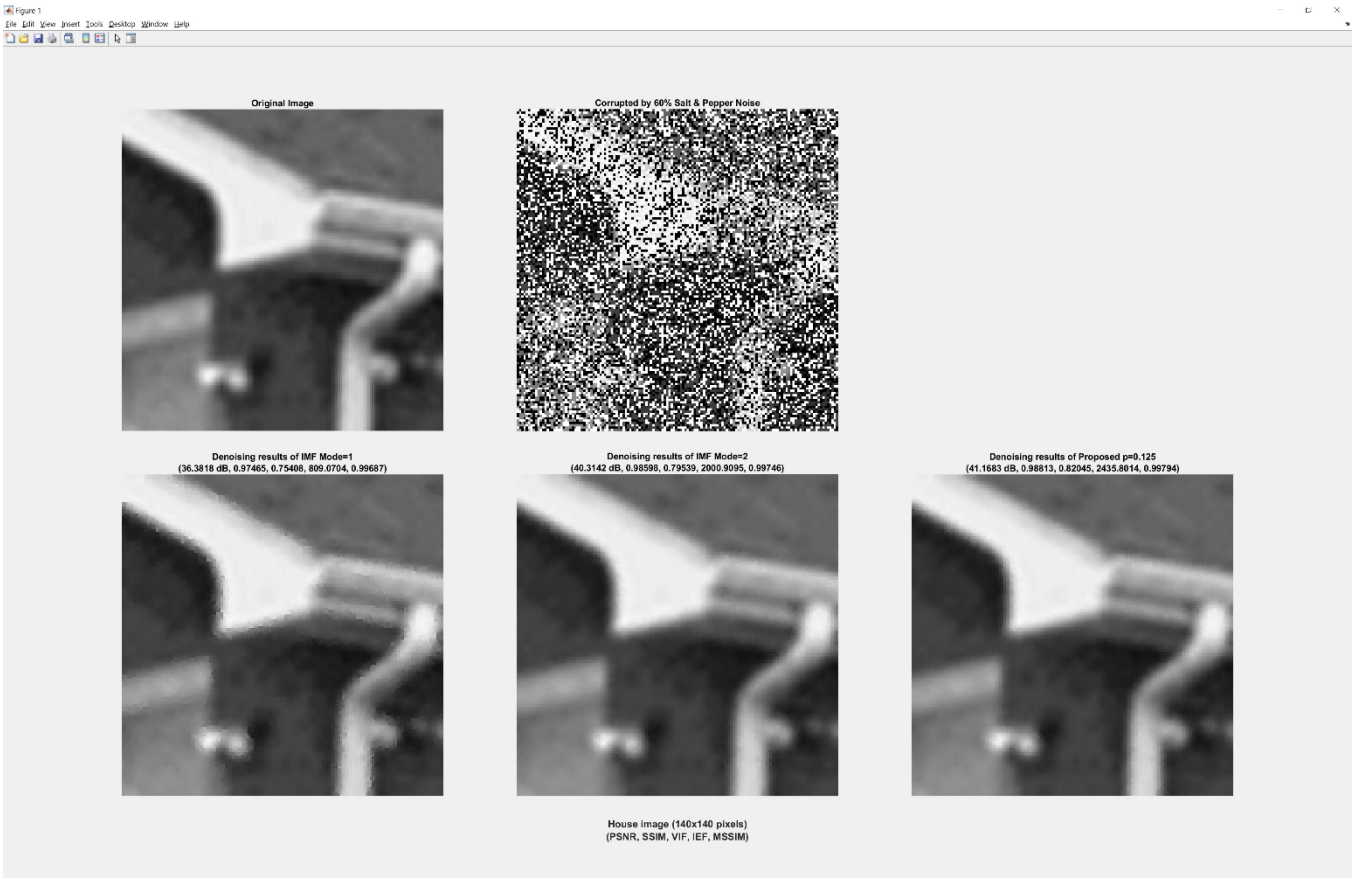

Fig.4. Denoising results for the cropped House image (60%).

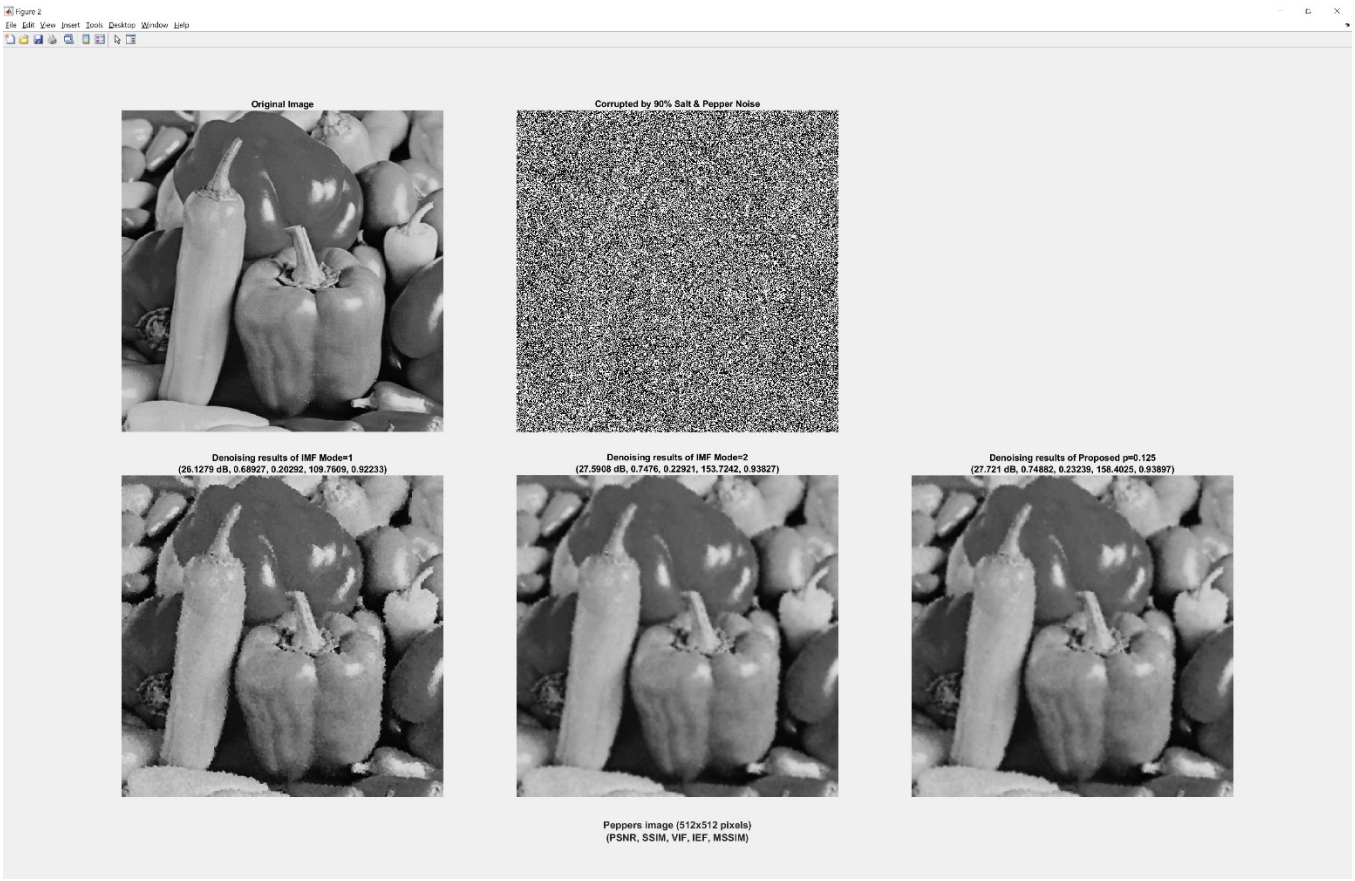

Fig.5. Denoising results for the Peppers image (90%).

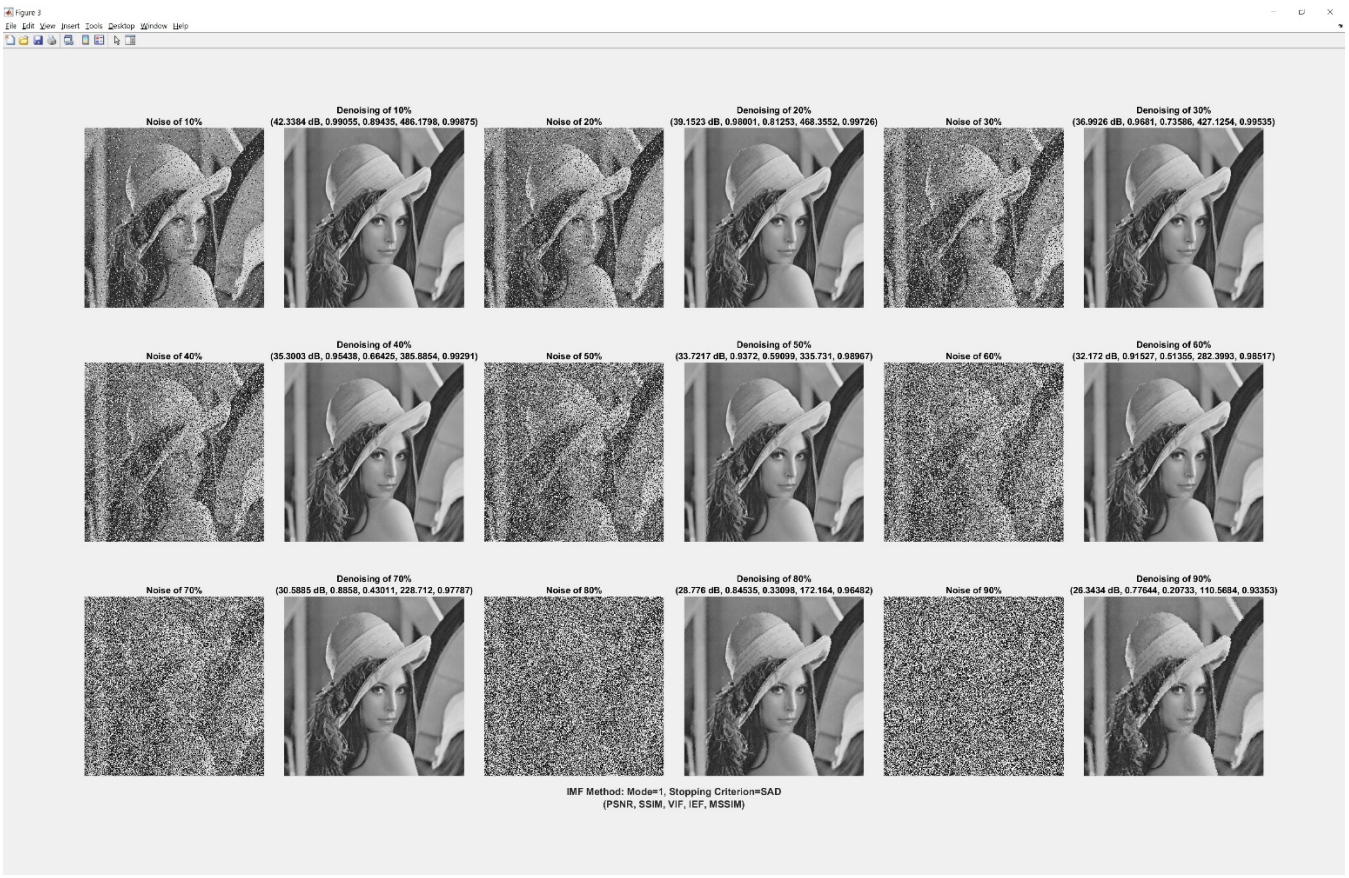

Fig.6. Denoising results of IMF (Mode=1) for the Lena image.

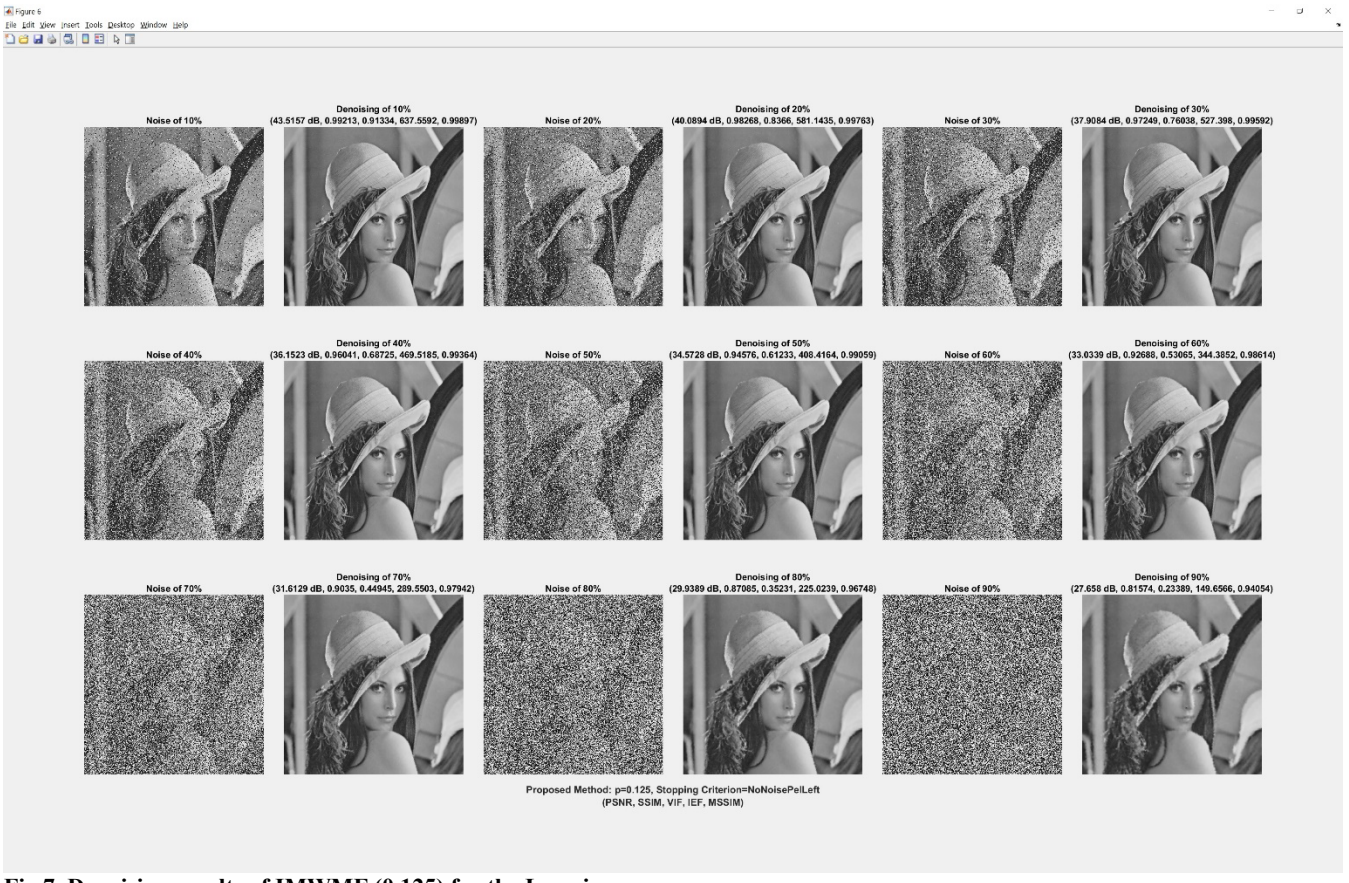

Fig.7. Denoising results of IMWMF (0.125) for the Lena image.

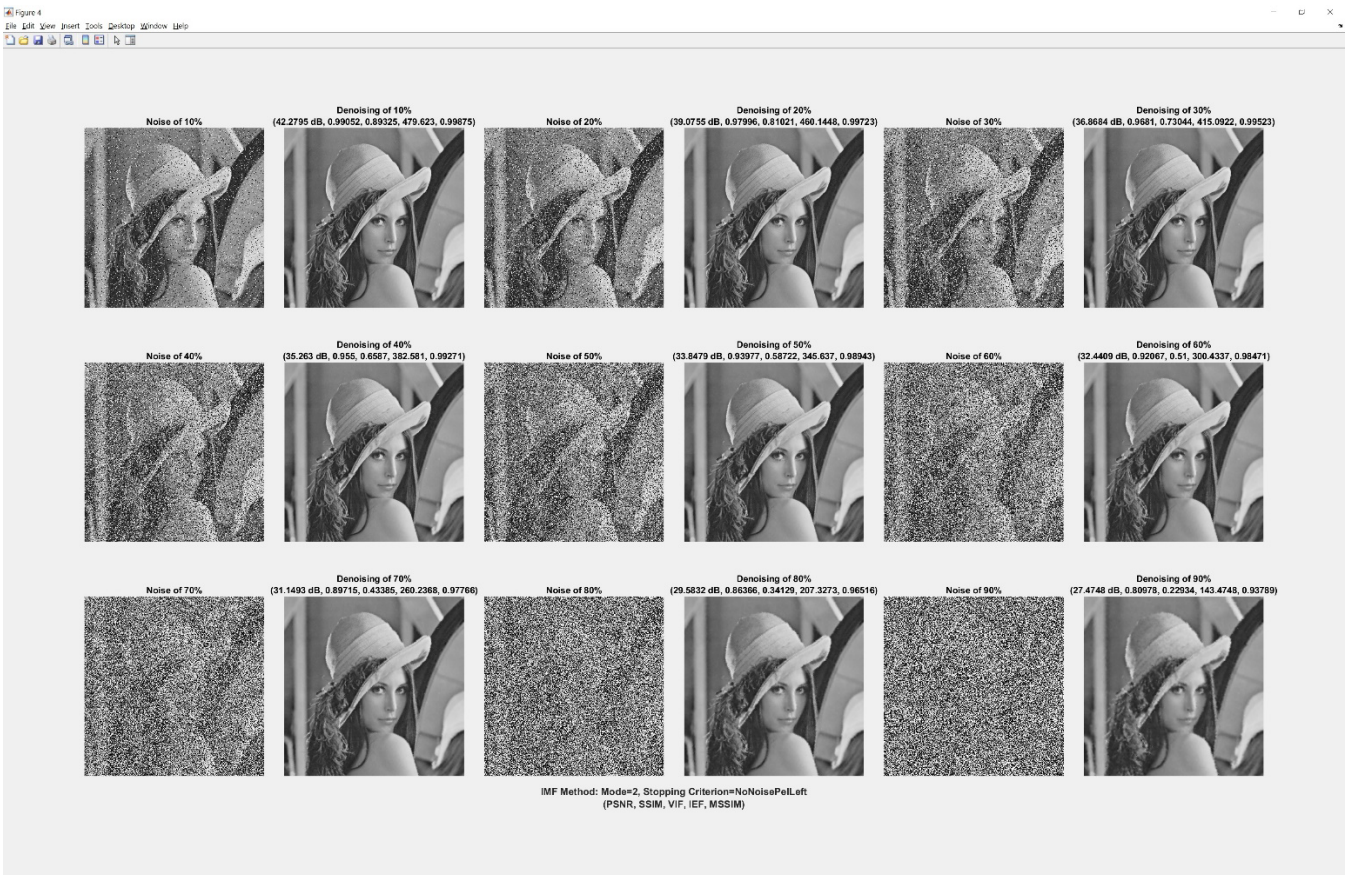

Fig.8. Denoising results of IMF (Mode=2) for the Lena image.

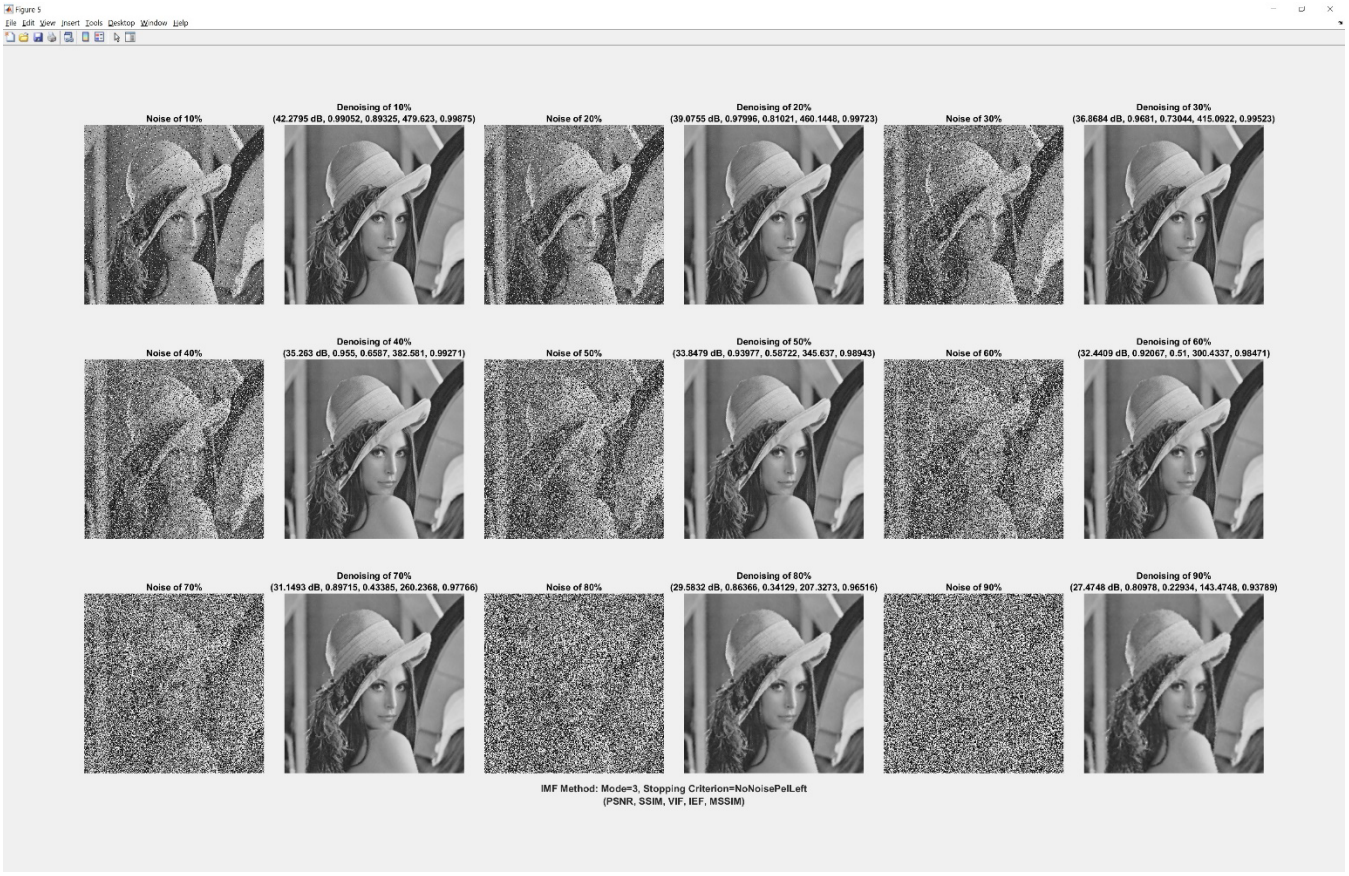

Fig.9. Denoising results of IMF (Mode=3) for the Lena image.
(Just for comparison that IMF (Mode=2) ≡ IMF (Mode=3)).

PSNR results of the methods for three traditional images with different SPN ratios.

Image	Filters	10%	20%	30%	40%	50%	60%	70%	80%	90%
Lena	IMF(Mode=1)	42.34	39.15	36.99	35.30	33.72	32.17	30.59	28.78	26.34
	IMF(Mode=2)	42.28	39.08	36.87	35.26	33.85	32.44	31.15	29.58	27.47
	Proposed	43.52	40.09	37.91	36.15	34.57	33.03	31.61	29.94	27.66
Peppers	IMF(Mode=1)	41.18	37.93	35.92	34.28	32.87	31.47	30.03	28.43	26.13
	IMF(Mode=2)	41.21	38.01	36.04	34.55	33.32	32.12	30.97	29.59	27.59
	Proposed	41.21	38.05	36.14	34.67	33.47	32.37	31.20	29.79	27.72
House	IMF(Mode=1)	50.58	45.97	43.21	40.80	38.99	36.38	34.08	31.52	27.41
	IMF(Mode=2)	51.43	47.98	45.69	43.70	41.97	40.31	37.71	34.80	30.47
	Proposed	53.49	49.52	46.48	45.17	43.18	41.17	38.51	35.42	30.83

Fig.10. PSNR results for three selected traditional images.

SSIM results of the methods for three traditional images with different SPN ratios.

Image	Filters	10%	20%	30%	40%	50%	60%	70%	80%	90%
Lena	IMF(Mode=1)	0.9905	0.9800	0.9681	0.9544	0.9372	0.9153	0.8858	0.8453	0.7764
	IMF(Mode=2)	0.9905	0.9800	0.9681	0.9550	0.9398	0.9207	0.8971	0.8637	0.8098
	Proposed	0.9921	0.9827	0.9725	0.9604	0.9458	0.9269	0.9035	0.8709	0.8157
Peppers	IMF(Mode=1)	0.9816	0.9611	0.9391	0.9140	0.8847	0.8494	0.8079	0.7582	0.6893
	IMF(Mode=2)	0.9817	0.9619	0.9411	0.9187	0.8944	0.8673	0.8371	0.8003	0.7476
	Proposed	0.9795	0.9584	0.9357	0.9124	0.8878	0.8645	0.8353	0.7989	0.7488
House	IMF(Mode=1)	0.9985	0.9964	0.9937	0.9898	0.9839	0.9747	0.9595	0.9332	0.8662
	IMF(Mode=2)	0.9986	0.9971	0.9952	0.9927	0.9895	0.9860	0.9785	0.9643	0.9283
	Proposed	0.9990	0.9977	0.9960	0.9945	0.9917	0.9881	0.9814	0.9681	0.9326

Fig.11. SSIM results for three selected traditional images.

Summary of denoising results for the traditional image dataset with different SPN ratios.

Algorithm	Criterion	10%	20%	30%	40%	50%	60%	70%	80%	90%	Mean
IMF(Mode=1)	PSNR	40.35	37.13	35.05	33.38	31.85	30.32	28.77	27.06	24.70	32.07
	SSIM	0.9860	0.9708	0.9535	0.9333	0.9087	0.8774	0.8373	0.7830	0.6939	0.8827
	VIF	0.8651	0.7739	0.6959	0.6229	0.5511	0.4764	0.3979	0.3069	0.1896	0.5422
	IEF	664.3	583.5	511.3	436.7	364.7	290.7	230.0	169.3	102.9	372.6
	MSSIM	0.9977	0.9951	0.9919	0.9878	0.9824	0.9748	0.9632	0.9429	0.8960	0.9702
	#Iteration	2.111	2.111	3.000	3.000	3.000	4.000	4.000	5.000	7.000	3.691
	ExecutionTime	0.08643	0.1632	0.2459	0.3237	0.4038	0.4904	0.5883	0.7248	0.9795	0.4451
IMF(Mode=2)	PSNR	40.33	37.12	35.03	33.45	32.10	30.72	29.45	27.96	25.90	32.45
	SSIM	0.9860	0.9707	0.9532	0.9334	0.9105	0.8815	0.8471	0.7989	0.7233	0.8894
	VIF	0.8642	0.7717	0.6909	0.6174	0.5465	0.4709	0.3964	0.3096	0.2020	0.5411
	IEF	681.9	618.9	546.7	485.1	429.7	352.7	296.2	224.4	142.5	419.8
	MSSIM	0.9977	0.9950	0.9917	0.9875	0.9821	0.9742	0.9631	0.9437	0.9019	0.9708
	#Iteration	2.111	2.111	3.000	3.000	3.000	4.000	4.000	5.000	7.000	3.691
	ExecutionTime	0.1797	0.3475	0.7282	0.9606	1.198	1.905	2.213	3.139	4.846	1.724
Proposed	PSNR	41.55	38.11	36.04	34.31	32.80	31.30	29.88	28.27	26.03	33.14
	SSIM	0.9874	0.9733	0.9575	0.9391	0.9172	0.8897	0.8556	0.8080	0.7308	0.8954
	VIF	0.8854	0.7981	0.7200	0.6452	0.5713	0.4914	0.4116	0.3201	0.2065	0.5611
	IEF	976.9	818.3	738.6	626.3	529.6	420.3	337.2	245.5	148.3	537.9
	MSSIM	0.9980	0.9956	0.9926	0.9888	0.9837	0.9763	0.9655	0.9467	0.9050	0.9725
	#Iteration	2.111	2.111	3.000	3.000	3.000	4.000	4.000	5.000	7.000	3.691
	ExecutionTime	0.2151	0.4213	0.8875	1.174	1.462	2.332	2.696	3.807	5.873	2.096

Fig.12. Avg. denoising results for the 18 traditional test images.

Summary of denoising results for the TESTIMAGES image dataset with different SPN ratios.

Algorithm	Criterion	10%	20%	30%	40%	50%	60%	70%	80%	90%	Mean
	PSNR	42.05	38.88	36.72	34.95	33.23	31.53	29.75	27.77	24.92	33.31
	SSIM	0.9916	0.9832	0.9735	0.9617	0.9463	0.9256	0.8967	0.8531	0.7698	0.9224
	VIF	0.8898	0.8113	0.7394	0.6705	0.5995	0.5261	0.4451	0.3486	0.2213	0.5835
IMF(Mode=1)	IEF	842.3	806.6	724.0	629.1	517.9	411.5	313.5	223.8	127.1	510.6
	MSSIM	0.9989	0.9975	0.9958	0.9936	0.9904	0.9858	0.9782	0.9636	0.9249	0.9810
	#Iteration	7.250	7.375	7.550	7.600	7.750	7.825	8.050	8.725	9.825	7.994
	ExecutionTime	0.1555	0.2630	0.3724	0.4790	0.5871	0.7022	0.8370	1.024	1.370	0.6433
	PSNR	42.04	38.81	36.64	34.92	33.32	31.80	30.24	28.47	26.09	33.59
	SSIM	0.9915	0.9829	0.9727	0.9604	0.9450	0.9251	0.8982	0.8585	0.7912	0.9250
	VIF	0.8886	0.8074	0.7321	0.6601	0.5868	0.5126	0.4343	0.3460	0.2367	0.5783
IMF(Mode=2)	IEF	846.6	807.3	730.0	648.5	553.0	462.3	371.8	279.1	176.1	541.6
	MSSIM	0.9988	0.9974	0.9956	0.9930	0.9893	0.9837	0.9749	0.9589	0.9224	0.9793
	#Iteration	7.250	7.375	7.550	7.600	7.750	7.825	8.050	8.725	9.825	7.994
	ExecutionTime	0.9092	1.714	2.565	3.379	4.267	5.139	6.132	7.536	9.399	4.560
	PSNR	43.92	40.46	38.09	36.17	34.39	32.69	30.95	29.00	26.36	34.67
	SSIM	0.9938	0.9871	0.9787	0.9682	0.9543	0.9358	0.9101	0.8709	0.8017	0.9334
	VIF	0.9154	0.8425	0.7693	0.6956	0.6187	0.5396	0.4553	0.3603	0.2428	0.6044
Proposed	IEF	1301.	1176.	1016.	862.7	706.8	567.9	438.7	315.5	187.6	730.3
	MSSIM	0.9992	0.9981	0.9966	0.9944	0.9912	0.9862	0.9782	0.9632	0.9273	0.9816
	#Iteration	7.250	7.375	7.550	7.600	7.750	7.825	8.050	8.725	9.825	7.994
	ExecutionTime	1.091	2.084	3.127	4.133	5.226	6.295	7.498	9.199	11.44	5.565

Fig.13. Avg. denoising results for the TESTIMAGES dataset.

Summary of denoising results for the BSDS image dataset with different SPN ratios.

Algorithm	Criterion	10%	20%	30%	40%	50%	60%	70%	80%	90%	Mean
	PSNR	36.87	34.00	32.11	30.62	29.25	27.91	26.51	24.98	22.95	29.47
	SSIM	0.9841	0.9681	0.9496	0.9281	0.9016	0.8683	0.8242	0.7641	0.6651	0.8726
	VIF	0.7942	0.6949	0.6149	0.5439	0.4760	0.4084	0.3360	0.2560	0.1542	0.4754
IMF(Mode=1)	IEF	254.6	256.7	245.0	227.8	205.8	179.0	149.3	118.1	80.82	190.8
	MSSIM	0.9970	0.9936	0.9894	0.9840	0.9765	0.9661	0.9501	0.9229	0.8622	0.9602
	#Iteration	3.280	3.345	4.115	4.390	4.395	4.440	5.040	5.955	7.825	4.754
	ExecutionTime	0.06050	0.1060	0.1549	0.2019	0.2493	0.2987	0.3582	0.4388	0.5932	0.2735
	PSNR	36.83	33.93	32.02	30.56	29.31	28.18	26.99	25.73	24.07	29.74
	SSIM	0.9840	0.9675	0.9483	0.9260	0.8996	0.8677	0.8258	0.7700	0.6853	0.8749
	VIF	0.7932	0.6924	0.6099	0.5375	0.4698	0.4036	0.3323	0.2556	0.1620	0.4729
IMF(Mode=2)	IEF	252.3	252.2	239.4	225.0	209.3	191.3	167.9	141.1	105.6	198.2
	MSSIM	0.9969	0.9935	0.9890	0.9832	0.9753	0.9646	0.9484	0.9220	0.8680	0.9601
	#Iteration	3.280	3.345	4.115	4.390	4.395	4.440	5.040	5.955	7.825	4.754
	ExecutionTime	0.1928	0.3461	0.6076	0.8445	1.045	1.256	1.649	2.204	3.203	1.261
	PSNR	38.34	35.17	33.17	31.50	30.06	28.73	27.36	25.97	24.13	30.49
	SSIM	0.9884	0.9751	0.9598	0.9398	0.9150	0.8835	0.8400	0.7838	0.6944	0.8866
	VIF	0.8289	0.7306	0.6467	0.5693	0.4960	0.4237	0.3461	0.2643	0.1652	0.4968
Proposed	IEF	361.2	338.2	316.1	282.1	250.8	219.0	184.0	150.0	107.6	245.4
	MSSIM	0.9978	0.9950	0.9912	0.9860	0.9787	0.9684	0.9522	0.9261	0.8713	0.9630
	#Iteration	3.280	3.345	4.115	4.390	4.395	4.440	5.040	5.955	7.825	4.754
	ExecutionTime	0.2314	0.4213	0.7412	1.033	1.280	1.538	2.017	2.688	3.898	1.539

Fig.14. Avg. denoising results for the BSDS dataset.

Execution time comparison of the methods(averaged from 3 image datasets).

Noise Density	IMF(Mode=1)	IMF(Mode=2)	Proposed
10%	0.1	0.43	0.51
20%	0.18	0.8	0.98
30%	0.26	1.3	1.59
40%	0.33	1.73	2.11
50%	0.41	2.17	2.66
60%	0.5	2.77	3.39
70%	0.59	3.33	4.07
80%	0.73	4.29	5.23
90%	0.98	5.82	7.07
Mean	0.45	2.51	3.07

Fig.15. Execution time comparison (in second).

C. Discussion

1) The First Test Case

The 3 selected images from traditional images are cropped House (size 140 × 140 pixels), Peppers, and Lena (both size 512 × 512 pixels). These 3 images are corrupted with SPN of 60%, 90%, and 10% to 90% of noise densities respectively.

For the cropped House image corrupted with SPN of 60% noise density, the denoising results are depicted in Fig.4. and the results show that our proposed IMWMF worked best in terms of assessment metrics and visual quality (not much noticeable visual quality difference between IMWMF and IMF (Mode=2) and the result from [1]). All 3 methods can eliminate all noisy pixels, but only IMF (Mode=1) has some visible artifacts on the edges of the objects, while the rest methods have a lot smoother on the edges.

For the Peppers image corrupted with SPN of 90% noise density, the denoising results are portrayed in Fig.5. and the results show that our proposed IMWMF still worked best in terms of assessment metrics and visual quality (but for SPN of 10%-80%, IMF (Mode=2) has better SSIM than our proposed IMWMF in only this particular Peppers image, especially in low-density SPN, but our IMWMF still outperforms in other metrics as we can see in Fig.10.–11.). But only IMF (Mode=1) can remove all noisy pixels, the other 2 methods (and also the result from [1]) can alleviate all noisy pixels, but they still have a few noticeable impulse pixels (they are not SPN with 255 or 0 intensities anymore) which is not desirable. However, these impulse pixels from our proposed IMWMF also look softer as we can see in Fig. 20.–21. in the appendix section. The object details from IMF (Mode=1) look sharper, but it has visible artifacts on both object details and edges, while the other 2 methods look more blur on the object details but have smoother edges (these observations persist in the next Lena case, especially on high-density SPN).

For the Lena image corrupted with SPN of 10%-90% noise densities, the denoising results are presented in Fig.6.–8. The results show that our proposed IMWMF still worked best in terms of overall assessment metrics (except for some metrics in a few cases) and visual quality in all noise densities. It has a few impulse pixels left in high-density SPN as in IMF (Mode=2) and in the results from [1], but it's softer (which is better) than the latter two. No significant artifacts remained even in high-density SPN cases. In summary, all 3 methods have decent preservation on edges (except IMF (Mode=1) in medium to high-density SPN), object details, and other image structures, but our proposed IMWMF performs a little better in terms of noticeable visual quality.

2) The Second Test Case

The denoising results in terms of image quality assessment metrics for 3 image datasets: traditional, TESTIMAGES, and BSDS datasets are shown in Fig.12.–14. respectively as the average values of the corresponding metrics/noise densities from all images in the particular dataset. We can clearly see that our proposed IMWMF outperforms the others in almost all cases (except only the average MSSIM value at a noise density of 80% SPN in the TESTIMAGES dataset and the execution time at all noise densities that IMF (Mode=1) can outperform our proposed IMWMF). Noted that, the number of iterations used until the algorithm converged in all methods is equal in all cases, so, we may omit it from our consideration.

3) The Third Test Case

The execution time of the denoising algorithms (reflect the computational complexity/processing performance) is shown in Fig.15. as the average values of the execution time from the aforementioned datasets. We can see that IMF (Mode=1) is the fastest algorithm and far better than the others (because it doesn't repeat the calculation on the denoised pixels, thus, it has less computational burden compared to the others), our proposed IMWMF is the slowest, but it's still comparable to IMF (Mode=2). In other words, the execution time is not much different and still acceptable.

V. Conclusion

We have studied the IMF algorithm in [1], reproduced the results with the same settings, and developed an improved version of IMF by introducing the Minkowski–weighted to the constrained mean, and with some other adjustments, we have proposed an IMWMF algorithm that can outperform the IMF in almost all cases. The experimental results show that our proposed method is better in the quality assessment metrics sense, but there is not much difference from IMF (Mode=2) in the visual quality (but still noticeable in Fig.20.–21. that our proposed method is better).

From the 3 test cases in the previous section, in the visual quality results, we can see that the results from IMF (Mode=2) look closer to the results in [1], and in the assessment metrics results, they are also closer to the results in [1]. Therefore, it's more likely that the authors in [1] implement something like Algorithm 2, not Algorithm 1 (maybe they update their code that is a different version – we can notice that IMF (Mode=1) performs better than IMF (Mode=2) in most of low-density SPN, looks sharper in object details even it has the artifacts in high-density SPN, and a lot lower execution times, so, we can't conclude that IMF (Mode=2) outperforms IMF (Mode=1)).

REFERENCES

[1] U. Erkan, D. N. H. Thanh, L. M. Hieu, and S. Engínoğlu, "An Iterative Mean Filter for Image Denoising," in *IEEE Access*, vol. 7, pp. 167847–167859, 2019.

[2] M. Yildirim, "Analog circuit implementation based on median filter for salt and pepper noise reduction in image," *Analog Integr. Circuits Signal Process*, vol. 107(1), pp. 195–202, 2021.

[3] H. Hwang and R.A. Haddad, "Adaptive median filters: New algorithms and results," *IEEE Trans. Image Process*, vol. 4(4), pp. 499–502, 1995.

[4] R.H. Chan, C.-W. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail preserving regularization," *IEEE Trans. Image Process*, vol. 14(10), pp. 1479–1485, 2005.

[5] Y. Dong and S. Xu, "A new directional weighted median filter for removal of random-valued impulse noise," *IEEE Signal Process Lett*, vol. 14(3), pp. 193–196, 2007.

[6] U. Erkan, D.N.H. Thanh, S. Engínoğlu, and S. Memis, "Improved adaptive weighted mean filter for salt-and-pepper noise removal," in *Int. Conf. on Electrical, Communication, and Computer Engineering*, Istanbul, Turkey, 2020.

[7] L.I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D*, vol. 60(1–4), pp. 259–268, 1992.

[8] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, "An iterative regularization method for total variation-based image restoration," *SIAM J. Multiscale Model. Simul*, vol. 4(2), pp. 460–489, 2005.

[9] D.N.H. Thanh, N.N. Hien, and S. Prasath, "Adaptive total variation L1 regularization for salt and pepper image denoising," *Optik*, vol. 208, 163677, 2020.

[10] B. Fu, X. Zhao, Y. Li, X. Wang, and Y. Ren, "A convolutional neural networks denoising approach for salt and pepper noise," *Multimed. Tools Appl*, vol. 78(21), pp. 30707–30721, 2018.

[11] G. Li, X. Xu, M. Zhang, and Q. Liu, "Densely connected network for impulse noise removal," *Pattern Anal. Appl*, vol. 23(5), pp. 1263–1275, 2020.

[12] C.T. Lu, H.J. Hsu, and L.L. Wang, "Image denoising using dlnn to recognize the direction of pixel variation," *Signal Image Video Process*, vol. 10, pp. 1–10, 2021.

[13] J. S. Lim, *Two-Dimensional Signal and Image Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1990.

[14] P. Zhang and F. Li, "A New Adaptive Weighted Mean Filter for Removing Salt-and-Pepper Noise," in *IEEE Signal Processing Letters*, vol. 21, no. 10, pp. 1280–1283, 2014.

[15] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: what is it and how does it work?," *International journal of methods in psychiatric research*, vol. 20(1), pp. 40–49, 2011.

[16] D. J. Stekhoven and P. Bühlmann, "MissForest–nonparametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28(1), pp. 112–118, 2012.

[17] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. Altman, "Missing value estimation methods for DNA microarrays," *Bioinformatics* vol. 17(6), pp. 520–525, 2001.

[18] S.H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1(4), pp. 300–307, 2007.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13(4), pp. 600–612, 2004.

[20] H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Trans. Image Process*., vol. 15(2), pp. 430–444, 2006.

[21] I. Djurović, "Combination of the adaptive Kuwahara and BM3D filters for filtering mixed Gaussian and impulsive noise," *Signal, Image Video Process*., vol. 11(4), pp. 753–760, 2017.

[22] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Proc. Conf. Rec. Asilomar Conf. Signals, Syst. Comput*., 2003, pp. 1398–1402.

[23] N. Asuni and A. Giachetti, "TESTIMAGES: A large-scale archive for testing visual devices and basic image processing algorithms," in *Proc. Eurograph. Italian Chapter Conf.*, 2014, pp. 1–3.

APPENDIX

1) Comparison between IMF (Mode=2) and IMF (Mode=3).

####Results of IMF Method: Mode=2, Stopping Criterion=NoNoisePelLeft (lena.jpg)####

	10%	20%	30%	40%	50%	60%	70%	80%	90%	Mean
PSNR	42.279	39.076	36.868	35.263	33.848	32.441	31.149	29.583	27.475	34.22
SSIM	0.99052	0.97996	0.9681	0.955	0.93977	0.92067	0.89715	0.86366	0.80978	0.92496
VIF	0.89325	0.81021	0.73044	0.6587	0.58722	0.51	0.43385	0.34129	0.22934	0.57715
IEF	479.62	460.14	415.09	382.58	345.64	300.43	260.24	207.33	143.47	332.73
MSSIM	0.99875	0.99723	0.99523	0.99271	0.98943	0.98471	0.97766	0.96516	0.93789	0.98208
#Iteration	2	2	3	3	3	4	4	5	7	3.6667
ExecutionTime	0.1785	0.32455	0.72254	0.95885	1.1964	1.8975	2.2136	3.1392	4.8576	1.721

Fig.16. Denoising results (in terms of assessment metrics) of IMF (Mode=2) for the Lena image.

####Results of IMF Method: Mode=3, Stopping Criterion=NoNoisePelLeft (lena.jpg)####

	10%	20%	30%	40%	50%	60%	70%	80%	90%	Mean
PSNR	42.279	39.076	36.868	35.263	33.848	32.441	31.149	29.583	27.475	34.22
SSIM	0.99052	0.97996	0.9681	0.955	0.93977	0.92067	0.89715	0.86366	0.80978	0.92496
VIF	0.89325	0.81021	0.73044	0.6587	0.58722	0.51	0.43385	0.34129	0.22934	0.57715
IEF	479.62	460.14	415.09	382.58	345.64	300.43	260.24	207.33	143.47	332.73
MSSIM	0.99875	0.99723	0.99523	0.99271	0.98943	0.98471	0.97766	0.96516	0.93789	0.98208
#Iteration	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ExecutionTime	0.16358	0.30763	0.68162	0.8986	1.1178	1.7888	2.0719	2.9374	4.4934	1.6068

Fig.17. Denoising results (in terms of assessment metrics) of IMF (Mode=3) for the Lena image.

2) SAD comparison between IMF (Mode=1) and IMF (Mode=2).

```
####Applying IMF Method: Mode=1, Stopping Criterion=SAD (lena.jpg)####
-For 10% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=26020, SAD=9834, PSNR=42.3384
Iteration#2(k=1): #Noise pixels left=0, SAD=0, PSNR=42.3384
*PSNR=42.3384,SSIM=0.99055,VIF=0.89435,IEF=486.1798,MSSIM=0.99875(TotalIteration=2,ExecutionTime=0.08751sec.)
-For 20% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=52223, SAD=16398, PSNR=39.1523
Iteration#2(k=1): #Noise pixels left=0, SAD=0, PSNR=39.1523
*PSNR=39.1523,SSIM=0.98001,VIF=0.81253,IEF=468.3552,MSSIM=0.99726(TotalIteration=2,ExecutionTime=0.1655sec.)
-For 30% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=78487, SAD=24239, PSNR=36.9051
Iteration#2(k=1): #Noise pixels left=4, SAD=141, PSNR=36.9926
Iteration#3(k=2): #Noise pixels left=0, SAD=0, PSNR=36.9926
*PSNR=36.9926,SSIM=0.9681,VIF=0.73586,IEF=427.1254,MSSIM=0.99535(TotalIteration=3,ExecutionTime=0.25031sec.)
-For 40% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=104535, SAD=30531, PSNR=34.3092
Iteration#2(k=1): #Noise pixels left=70, SAD=480, PSNR=35.3003
Iteration#3(k=2): #Noise pixels left=0, SAD=0, PSNR=35.3003
*PSNR=35.3003,SSIM=0.95438,VIF=0.66425,IEF=385.8854,MSSIM=0.99291(TotalIteration=3,ExecutionTime=0.33685sec.)
-For 50% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=130882, SAD=37369, PSNR=30.2264
Iteration#2(k=1): #Noise pixels left=514, SAD=1165, PSNR=33.7217
Iteration#3(k=2): #Noise pixels left=0, SAD=0, PSNR=33.7217
*PSNR=33.7217,SSIM=0.9372,VIF=0.59099,IEF=335.731,MSSIM=0.98967(TotalIteration=3,ExecutionTime=0.41359sec.)
-For 60% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=157294, SAD=44263, PSNR=24.461
Iteration#2(k=1): #Noise pixels left=2759, SAD=2627, PSNR=32.1598
Iteration#3(k=2): #Noise pixels left=1, SAD=171, PSNR=32.172
Iteration#4(k=3): #Noise pixels left=0, SAD=0, PSNR=32.172
*PSNR=32.172,SSIM=0.91527,VIF=0.51355,IEF=282.3993,MSSIM=0.98517(TotalIteration=4,ExecutionTime=0.50401sec.)
-For 70% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=183606, SAD=47274, PSNR=19.0926
Iteration#2(k=1): #Noise pixels left=10622, SAD=6668, PSNR=30.4528
Iteration#3(k=2): #Noise pixels left=28, SAD=452, PSNR=30.5885
Iteration#4(k=3): #Noise pixels left=0, SAD=0, PSNR=30.5885
*PSNR=30.5885,SSIM=0.8858,VIF=0.43011,IEF=228.712,MSSIM=0.97787(TotalIteration=4,ExecutionTime=0.59891sec.)
-For 80% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=209843, SAD=47513, PSNR=14.0444
Iteration#2(k=1): #Noise pixels left=35448, SAD=14863, PSNR=25.9713
Iteration#3(k=2): #Noise pixels left=1092, SAD=2135, PSNR=28.7293
Iteration#4(k=3): #Noise pixels left=11, SAD=411, PSNR=28.776
Iteration#5(k=4): #Noise pixels left=0, SAD=0, PSNR=28.776
*PSNR=28.776,SSIM=0.84535,VIF=0.33098,IEF=172.164,MSSIM=0.96482(TotalIteration=5,ExecutionTime=0.73902sec.)
-For 90% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=235997, SAD=40594, PSNR=9.5297
Iteration#2(k=1): #Noise pixels left=101625, SAD=27395, PSNR=16.4777
Iteration#3(k=2): #Noise pixels left=18878, SAD=9554, PSNR=24.1556
Iteration#4(k=3): #Noise pixels left=1463, SAD=3644, PSNR=26.2767
Iteration#5(k=4): #Noise pixels left=38, SAD=715, PSNR=26.3416
Iteration#6(k=5): #Noise pixels left=1, SAD=130, PSNR=26.3434
Iteration#7(k=6): #Noise pixels left=0, SAD=0, PSNR=26.3434
*PSNR=26.3434,SSIM=0.77644,VIF=0.20733,IEF=110.5684,MSSIM=0.93353(TotalIteration=7,ExecutionTime=0.98274sec.)
```

Fig.18. Execution log of IMF (Mode=1) on Lena image with SPN, noise density from 10%-90%.

```
####Applying IMF Method: Mode=2, Stopping Criterion=NoNoisePelLeft (lena.jpg)####
-For 10% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=26020, SAD=9834, PSNR=42.3384
Iteration#2(k=1): #Noise pixels left=0, SAD=68, PSNR=42.2795
*PSNR=42.2795,SSIM=0.99052,VIF=0.89325,IEF=479.623,MSSIM=0.99875(TotalIteration=2,ExecutionTime=0.1785sec.)
-For 20% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=52223, SAD=16398, PSNR=39.1523
Iteration#2(k=1): #Noise pixels left=0, SAD=222, PSNR=39.0755
*PSNR=39.0755,SSIM=0.97996,VIF=0.81021,IEF=460.1448,MSSIM=0.99723(TotalIteration=2,ExecutionTime=0.32455sec.)
-For 30% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=78487, SAD=24239, PSNR=36.9051
Iteration#2(k=1): #Noise pixels left=4, SAD=405, PSNR=36.953
Iteration#3(k=2): #Noise pixels left=0, SAD=55, PSNR=36.8684
*PSNR=36.8684,SSIM=0.9681,VIF=0.73044,IEF=415.0922,MSSIM=0.99523(TotalIteration=3,ExecutionTime=0.72254sec.)
-For 40% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=104535, SAD=30531, PSNR=34.3092
Iteration#2(k=1): #Noise pixels left=70, SAD=784, PSNR=35.403
Iteration#3(k=2): #Noise pixels left=0, SAD=110, PSNR=35.263
*PSNR=35.263,SSIM=0.955,VIF=0.6587,IEF=382.581,MSSIM=0.99271(TotalIteration=3,ExecutionTime=0.95885sec.)
-For 50% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=130882, SAD=37369, PSNR=30.2264
Iteration#2(k=1): #Noise pixels left=514, SAD=1434, PSNR=34.0432
Iteration#3(k=2): #Noise pixels left=0, SAD=211, PSNR=33.8479
*PSNR=33.8479,SSIM=0.93977,VIF=0.58722,IEF=345.637,MSSIM=0.98943(TotalIteration=3,ExecutionTime=1.1964sec.)
-For 60% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=157294, SAD=44263, PSNR=24.461
Iteration#2(k=1): #Noise pixels left=2759, SAD=3191, PSNR=32.77
Iteration#3(k=2): #Noise pixels left=1, SAD=387, PSNR=32.5862
Iteration#4(k=3): #Noise pixels left=0, SAD=174, PSNR=32.4409
*PSNR=32.4409,SSIM=0.92067,VIF=0.51,IEF=300.4337,MSSIM=0.98471(TotalIteration=4,ExecutionTime=1.8975sec.)
-For 70% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=183606, SAD=47274, PSNR=19.0926
Iteration#2(k=1): #Noise pixels left=10622, SAD=6983, PSNR=31.2783
Iteration#3(k=2): #Noise pixels left=28, SAD=630, PSNR=31.3246
Iteration#4(k=3): #Noise pixels left=0, SAD=287, PSNR=31.1493
*PSNR=31.1493,SSIM=0.89715,VIF=0.43385,IEF=260.2368,MSSIM=0.97766(TotalIteration=4,ExecutionTime=2.2136sec.)
-For 80% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=209843, SAD=47513, PSNR=14.0444
Iteration#2(k=1): #Noise pixels left=35448, SAD=15152, PSNR=26.356
Iteration#3(k=2): #Noise pixels left=1092, SAD=2325, PSNR=29.7455
Iteration#4(k=3): #Noise pixels left=11, SAD=606, PSNR=29.704
Iteration#5(k=4): #Noise pixels left=0, SAD=335, PSNR=29.5832
*PSNR=29.5832,SSIM=0.86366,VIF=0.34129,IEF=207.3273,MSSIM=0.96516(TotalIteration=5,ExecutionTime=3.1392sec.)
-For 90% salt & pepper noisy image of lena.jpg
Iteration#1(k=0): #Noise pixels left=235997, SAD=40594, PSNR=9.5297
Iteration#2(k=1): #Noise pixels left=101625, SAD=27882, PSNR=16.5087
Iteration#3(k=2): #Noise pixels left=18878, SAD=9809, PSNR=24.6913
Iteration#4(k=3): #Noise pixels left=1463, SAD=3826, PSNR=27.4549
Iteration#5(k=4): #Noise pixels left=38, SAD=939, PSNR=27.5713
Iteration#6(k=5): #Noise pixels left=1, SAD=499, PSNR=27.5338
Iteration#7(k=6): #Noise pixels left=0, SAD=373, PSNR=27.4748
*PSNR=27.4748,SSIM=0.80978,VIF=0.22934,IEF=143.4748,MSSIM=0.93789(TotalIteration=7,ExecutionTime=4.8576sec.)
```

Fig.19. Execution log of IMF (Mode=2) on Lena image with SPN, noise density from 10%-90%

3) Comparison between IMF (Mode=2) and Proposed IWMMF (p=0.125) at 90% density of SPN.

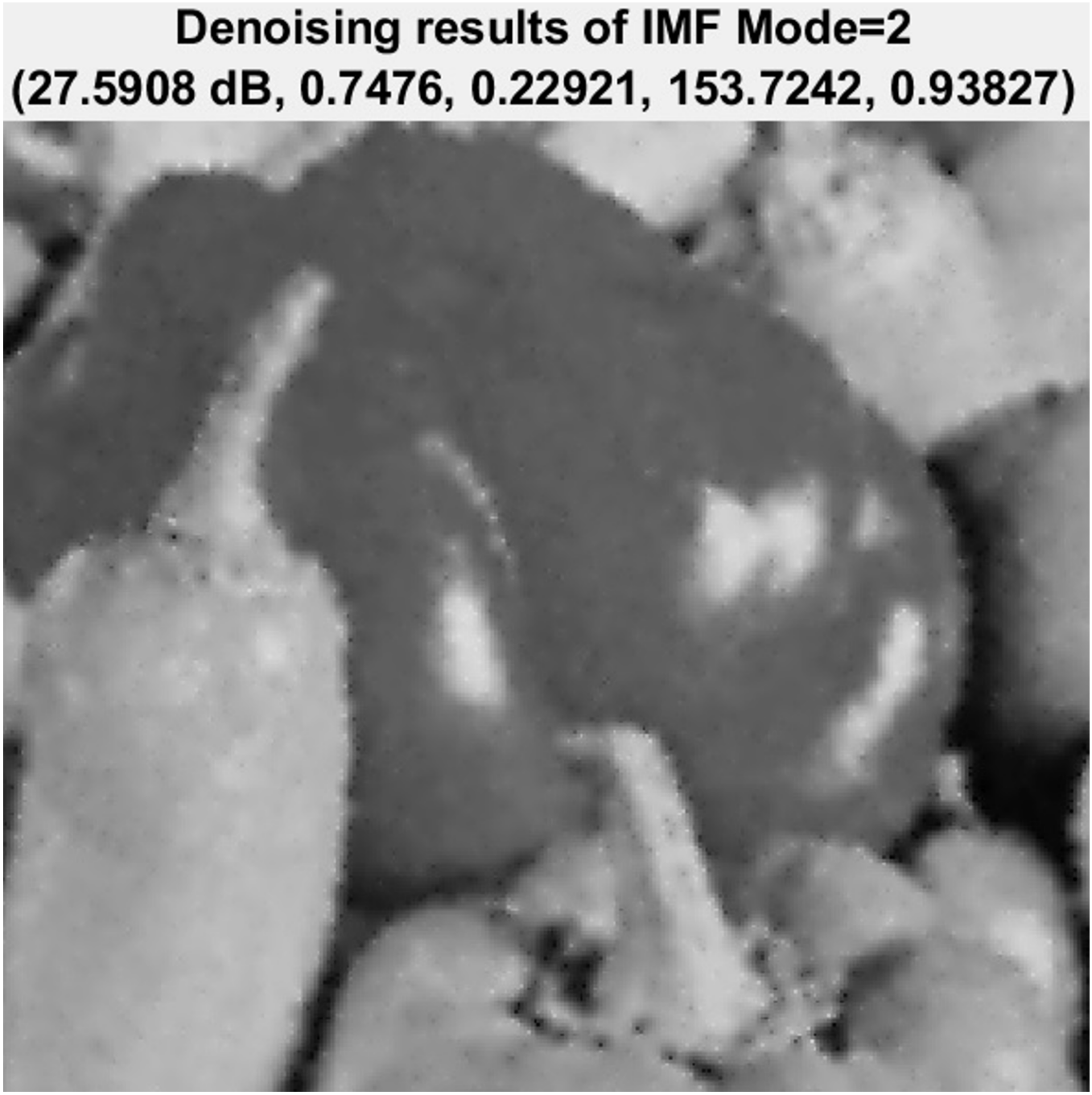

Fig.20. Denoising results (zoom-in version of Fig.5.) of IMF (Mode=2) for the Peppers image.

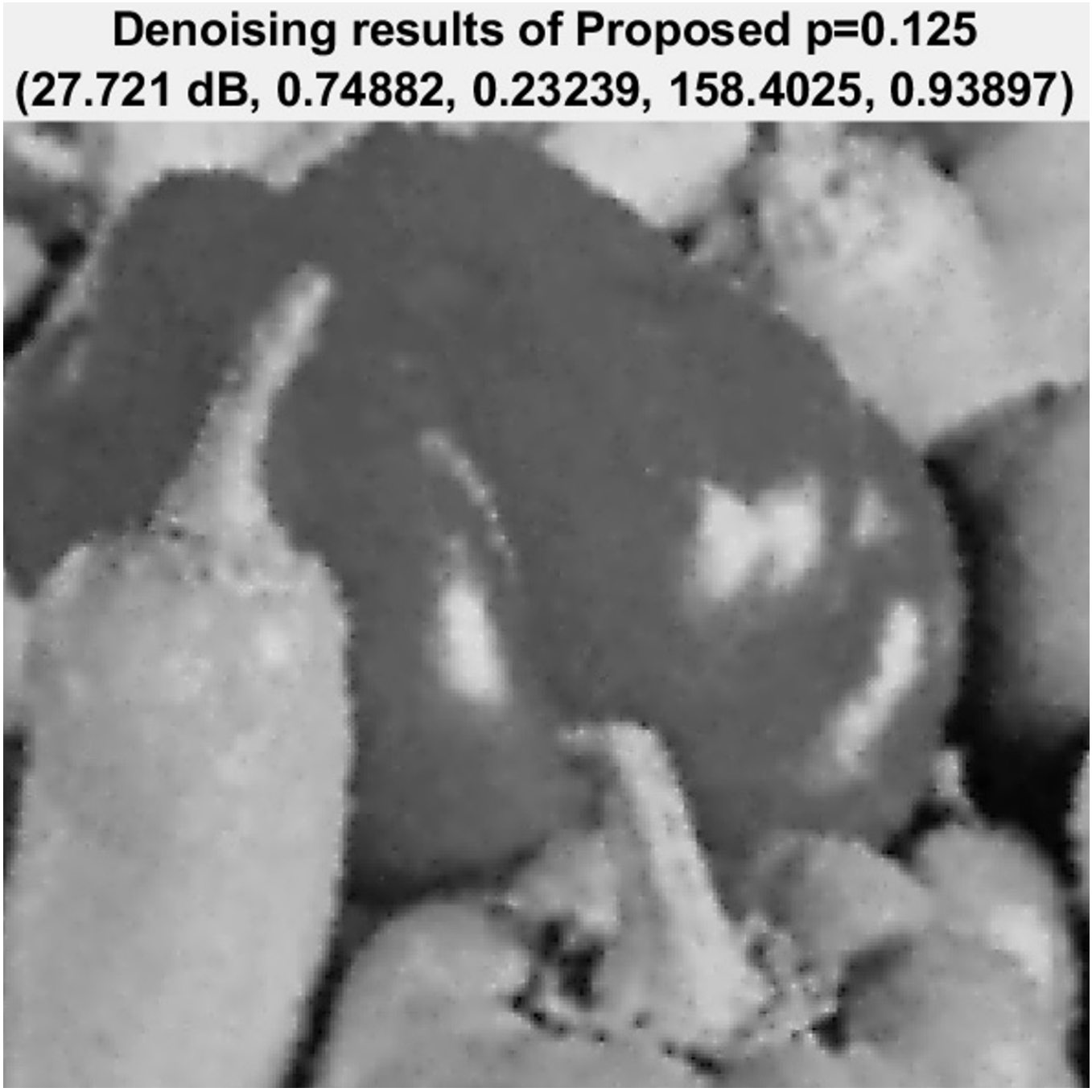

Fig.21. Denoising results (zoom-in version of Fig.5.) of our Proposed IMWMF for the Peppers image.

4) MATLAB Code.

```
clc;close all;clear;
if isfile("log.txt"),delete("log.txt");end
diary("log.txt");%Record to Log file
addpath(pwd+"\matlabPyrTools");%PyrTools required for VIF
global W1 W2 W3 W4 OriImg
p=0.125;%Minkowski distance of order p=0.125(best results among p=0.0625,0.125,0.25,0.5,1,2 in many datasets)
%Precompute weight matrix. If use adaptive size in loop(obsolete), precompute will reduce execution time
W1=ComputeMinkowskiWeightMatrix(p,1);W2=ComputeMinkowskiWeightMatrix(p,2);
W3=ComputeMinkowskiWeightMatrix(p,3);W4=ComputeMinkowskiWeightMatrix(p,4);OriImg=[];
AllowedImgExtension={'jpg','jpeg','png'};%Process only these image extensions(can add more)

SkipThisDataset={'Test_dataset'};%Use this one for faster testing
%SkipThisDataset={'traditional_dataset','BSDS_dataset','TESTIMAGES_dataset'};%Skip for faster testing

%Get all folders with _dataset(case insensitive use Regular Expression to match folder name)
files=dir;AllUsedFolders={files([files.isdir]).name};
AllUsedFolders=AllUsedFolders(~cellfun(@isempty,regexp(AllUsedFolders,"_(?i)dataset")));
AllUsedFolders=AllUsedFolders(~ismember(AllUsedFolders,SkipThisDataset));%Remove skip list
DatasetCount=0;DatasetList={};TotalUsedFolders=size(AllUsedFolders,2);
SummaryTablesArray=cell(1,TotalUsedFolders); SummaryTablesExcTime=cell(1,TotalUsedFolders);
%Loop all folders and process all image files
for fol=AllUsedFolders
    files=dir(pwd+"\"+fol);AllUsedFiles={files(~[files.isdir]).name};%Get all files in folder
    AllUsedFiles=AllUsedFiles(~cellfun(@isempty,regexp(AllUsedFiles,...
        "\.(?i)("+strjoin(AllowedImgExtension,"|")+")$")));%Match all file with allowed extensions
    if size(AllUsedFiles,2)>0
        DatasetCount=DatasetCount+1;
        DatasetName=regexp(fol,".*(?=_(?i)dataset)",...
            "match");%Get datasets name(use Positive Lookahead RegEx to match every thing before _dataset)
        disp("////Processing "+DatasetName{1}+" image dataset////"); DatasetList(end+1)={char(DatasetName{1})};
        ImgDataset=string(fol+"\"+AllUsedFiles);
        MethodName={'IMF(Mode=1)','IMF(Mode=2)','Proposed'};%Name of each method in ResultsArray
        ResultsArray=cell(1,3);%3=#Total methods used
        ResultsArray{1}=ProcessDataset(DatasetName,ImgDataset,"IMF",1);
        ResultsArray{2}=ProcessDataset(DatasetName,ImgDataset,"IMF",2);
        ResultsArray{3}=ProcessDataset(DatasetName,ImgDataset,"Proposed",2,p);
        if size(AllUsedFiles,2)>5 %Normal case use 4 significant digits
            SummaryTables=GetSummaryTable(ResultsArray,MethodName,"Algorithm","Criterion","Significant");
        else %For small image dataset use fixed 4 digits(for Test dataset)
            SummaryTables=GetSummaryTable(ResultsArray,MethodName,"Algorithm","Criterion","FixedDigits");
        end
        SummaryTablesArray{DatasetCount}=SummaryTables;
        SummaryTablesExcTime{DatasetCount}=GetOneMetricTable(ResultsArray,MethodName,"ExecutionTime",0);
    end
end
if DatasetCount>0
    ResultsExcTime=0;
    for i=1:DatasetCount
        disp("    Summary of denoising results for the "+DatasetList{i}+" image dataset with different SPN ratios.");
        disp(SummaryTablesArray{i});
        Tmp=SummaryTablesExcTime{i};%ExecutionTime
        if istable(ResultsExcTime), ResultsExcTime{:,:}=ResultsExcTime{:,:}+Tmp{:,:}; else, ResultsExcTime=Tmp; end
    end
    ResultsExcTime{:,:}=round(ResultsExcTime{:,:}/DatasetCount,2);%Overall Mean
    ResultsExcTime=rows2vars(ResultsExcTime,"VariableNamingRule","preserve");
    ResultsExcTime=renamevars(ResultsExcTime,1,"Noise Density");
    ResultsExcTime.("Noise Density")=categorical(ResultsExcTime{:,1});
    disp("    Execution time comparison of the methods(averaged from "+DatasetCount+" image datasets).");
    disp(ResultsExcTime);
end

disp("////Processing House, Peppers, and Lena images////");
MiniComparison(p,"House",im2gray(imread("house.jpg")),0.6)
MiniComparison(p,"Peppers",im2gray(imread("peppers.jpg")),0.9)

ImgName={'Lena','Peppers','House'};%Name of each image in Results
ResultsPSNR=cell(1,3);%3=#Total image used
ResultsSSIM=cell(1,3);%3=#Total image used
MethodName={'IMF(Mode=1)','IMF(Mode=2)','Proposed'};%Name of each method in Results
ResultsHouse=cell(1,3);%3=#Total methods to show
ResultsHouse{1}=ProcessAllNoiseDensity("house.jpg","IMF",1,0,0);%disp(ResultsHouse{1});
ResultsHouse{2}=ProcessAllNoiseDensity("house.jpg","IMF",2,0,0);%disp(ResultsHouse{2});
ResultsHouse{3}=ProcessAllNoiseDensity("house.jpg","Proposed",2,0,0,p);%disp(ResultsHouse{3});
ResultsPSNR{3}=GetOneMetricTable(ResultsHouse,MethodName,"PSNR",1);%Get only PSNR data
ResultsSSIM{3}=GetOneMetricTable(ResultsHouse,MethodName,"SSIM",1);%Get only SSIM data
ResultsPeppers=cell(1,3);%3=#Total methods to show
ResultsPeppers{1}=ProcessAllNoiseDensity("peppers.jpg","IMF",1,0,0);%disp(ResultsPeppers{1});
ResultsPeppers{2}=ProcessAllNoiseDensity("peppers.jpg","IMF",2,0,0);%disp(ResultsPeppers{2});
ResultsPeppers{3}=ProcessAllNoiseDensity("peppers.jpg","Proposed",2,0,0,p);%disp(ResultsPeppers{3});
ResultsPSNR{2}=GetOneMetricTable(ResultsPeppers,MethodName,"PSNR",1);%Get only PSNR data
ResultsSSIM{2}=GetOneMetricTable(ResultsPeppers,MethodName,"SSIM",1);%Get only SSIM data
ResultsLena=cell(1,3);%3=#Total methods to show
ResultsLena{1}=ProcessAllNoiseDensity("lena.jpg","IMF",1,1,1);
ResultsLena{2}=ProcessAllNoiseDensity("lena.jpg","IMF",2,1,1);
ResultsLenaIMFpcode=ProcessAllNoiseDensity("lena.jpg","IMF",3,1,1);
ResultsLena{3}=ProcessAllNoiseDensity("lena.jpg","Proposed",2,1,1,p);
ResultsPSNR{1}=GetOneMetricTable(ResultsLena,MethodName,"PSNR",1);%Get only PSNR data
```

```
ResultsSSIM{1}=GetOneMetricTable(ResultsLena,MethodName,"SSIM",1);%Get only SSIM data
SummaryTablesPSNR=GetSummaryTable(ResultsPSNR,ImgName,"Image","Filters","Significant");
disp("    PSNR results of the methods for three traditional images with different SPN ratios.")
disp(SummaryTablesPSNR);
SummaryTablesSSIM=GetSummaryTable(ResultsSSIM,ImgName,"Image","Filters","Significant");
disp("    SSIM results of the methods for three traditional images with different SPN ratios.")
disp(SummaryTablesSSIM);

diary off;
function MiniComparison(p,ImgName,U,nd)
    rng(13,"twister")%RNG using Mersenne Twister, fix seed#13 for repeatability
    B=imnoise(U,"salt & pepper",nd);%U=Original Image,nd=Noise Density
    [IMFMode1,~,~]=ApplyIMF(B,1,0); [IMFMode2,~,~]=ApplyIMF(B,2,0); [Proposed,~,~]=ApplyProposed(B,p,2,0);
    figure; t=tiledlayout(2,3,"TileSpacing","tight"); nexttile, imshow(U); title("Original Image");
    nexttile, imshow(B); title("Corrupted by "+num2str(nd*100)+"% Salt & Pepper Noise"); nexttile,imshow([]);
    nexttile, imshow(IMFMode1);title({"Denoising results of IMF Mode=1",GetMetricsStr(IMFMode1,B,U)});
    nexttile, imshow(IMFMode2);title({"Denoising results of IMF Mode=2",GetMetricsStr(IMFMode2,B,U)});
    nexttile, imshow(Proposed);title({"Denoising results of Proposed p="+p,GetMetricsStr(Proposed,B,U)});
    xlabel(t,[ImgName+" image ("+size(U,1)+"x"+size(U,2)+" pixels)",...
        "(PSNR, SSIM, VIF, IEF, MSSIM)"],"fontweight","bold");
end
function MetricsStr=GetMetricsStr(A,B,U)%Called by MiniComparison
    PSNR=ComputePSNR(A,U); SSIM=ssim(A,U);
    [~,VIF]=evalc('vifvec(A,U)');%Not show log from vifvec
    IEF=ComputeIEF(A,B,U); MSSIM=multissim(A,U);
    MetricsStr="("+PSNR+" dB, "+SSIM+", "+VIF+", "+IEF+", "+MSSIM+")";
end
function OneMetricTable=GetOneMetricTable(InputArray,RowNameLists,Criterion,DelLastCol)
    OneMetricTable=0;
    for i=1:size(InputArray,2)
        TmpTable=InputArray{i};
        TmpTable.Row(Criterion)={char(string(RowNameLists{i}))};%Filter only one metric
        TmpTable=TmpTable(RowNameLists{i},:);
        if istable(OneMetricTable)
            OneMetricTable=[OneMetricTable;TmpTable];%Concat table
        else
            OneMetricTable=TmpTable;
        end
    end
    if DelLastCol, OneMetricTable(:,end)=[]; end
end
function SummaryTable=GetSummaryTable(InputArray,FirstColLists,ColName1,ColName2,RoundType)
    SummaryTable=0;
    sr=size(InputArray{1}.Row,1); sc=size(InputArray{1}.Variables,2);
    for i=1:size(InputArray,2)
        TmpTable=addvars(InputArray{i},categorical(InputArray{i}.Properties.RowNames),...
            'Before',1,'NewVariableName',ColName2);TmpTable.Row=[];%Transform RowNames to Cols
        tmp=cell(sr,1);tmp(:)={char(0)};tmp(round(sr/2))={char(string(FirstColLists{i}))};%FirstCol
        TmpTable=addvars(TmpTable,categorical(tmp),'Before',1,'NewVariableName',ColName1);
        tmp=cell(1,sc+2);tmp(:)={char(0)};NullLine=array2table(categorical(tmp));%Separator NullLine
        NullLine.Properties.VariableNames=TmpTable.Properties.VariableNames;
        for j=TmpTable.Properties.VariableNames%Convert numeric to categorical for concat with NullLine
            if isnumeric(TmpTable.(string(j)))
                if RoundType=="Significant"
                    TmpTable.(string(j))=categorical(cellstr(num2str(TmpTable.(string(j)),"%#.4g")));
                else
                    TmpTable.(string(j))=categorical(cellstr(num2str(TmpTable.(string(j)),"%.4f")));
                end
            else
                TmpTable.(string(j))=categorical(TmpTable.(string(j)));
            end
        end
        if istable(SummaryTable)
            SummaryTable=[SummaryTable;NullLine;TmpTable];%Concat table
        else
            SummaryTable=TmpTable;
        end
    end
end
function arg=getArg(Method,Mode,p)%Called by ProcessDataset,ProcessAllNoiseDensity
    switch Method
        case "Proposed"
            if Mode==1, tmp=", Stopping Criterion=|MAD Diff.|";...
            else, tmp=", Stopping Criterion=NoNoisePelLeft"; end
            arg=": p="+num2str(p)+tmp;
        case "IMF"
            if Mode==1, tmp=", Stopping Criterion=SAD";...
            else, tmp=", Stopping Criterion=NoNoisePelLeft"; end
            arg=": Mode="+Mode+tmp;
        otherwise
            error("Stop! Invalid method");
    end
end
function OverallAvgResults=ProcessDataset(DatasetName,ImgDataset,Method,Mode,p)
    switch nargin
        case 5, arg=getArg(Method,Mode,p);
        case 4, arg=getArg(Method,Mode);
        otherwise, error("Stop! Invalid number of input argument");
    end
    TotalImg=size(ImgDataset,2);
```

```
    disp("@@@@Processing "+Method+" Method"+arg+" on "+DatasetName+" image dataset@@@@");
    SumResultsTmp=0;
    for ImgName=ImgDataset
        if nargin==5,Tmp=ProcessAllNoiseDensity(string(ImgName),Method,Mode,0,0,p); else,...
            Tmp=ProcessAllNoiseDensity(string(ImgName),Method,Mode,0,0); end
        disp("-Avg.Results=("+strjoin(string(Tmp.Mean),", ")+")")
        if istable(SumResultsTmp)
            SumResultsTmp{:,:}=SumResultsTmp{:,:}+Tmp{:,:};
        else
            SumResultsTmp=Tmp;
        end
    end
    SumResultsTmp{:,:}=SumResultsTmp{:,:}/TotalImg;%Overall Mean
    OverallAvgResults=SumResultsTmp;
    disp("@@@@Avg.Results of "+Method+" Method"+arg+" on "+TotalImg+" images("+DatasetName+" dataset)@@@@");
    disp(OverallAvgResults);
end
function MetricsResults=ProcessAllNoiseDensity(ImgName,Method,Mode,Log,Plot,p)
    global OriImg
    MetricsList={'PSNR','SSIM','VIF','IEF','MSSIM','#Iteration','ExecutionTime'};
    NoiseDensityList=0.1:0.1:0.9; NoisePercentage=[string(100*(NoiseDensityList))+"%" "Mean"];
    switch nargin
        case 6, arg=getArg(Method,Mode,p);
        case 5, arg=getArg(Method,Mode);
        otherwise, error("Stop! Invalid number of input argument");
    end
    if Plot
        figure; t=tiledlayout(3,6,"TileSpacing","tight");
        xlabel(t,[Method+" Method"+arg,"(PSNR, SSIM, VIF, IEF, MSSIM)"],"fontweight","bold");
    end
    MetricsTmp=[];
    disp("####Applying "+Method+" Method"+arg+" ("+ImgName+")####");
    for nd=NoiseDensityList
        rng(13,"twister")%RNG using Mersenne Twister, fix seed#13 for repeatability
        OriImg=im2gray(imread(ImgName));%Original image
        B=imnoise(OriImg,'salt & pepper',nd);%Corrupted by Salt&Pepper noise
        if Log, disp("-For "+num2str(nd*100)+"% salt & pepper noisy image of "+ImgName); end
        switch Method
            case "Proposed"
                [OutImp,ExecutionTime,iteration]=ApplyProposed(B,p,Mode,Log);
            case "IMF"
                [OutImp,ExecutionTime,iteration]=ApplyIMF(B,Mode,Log);
        end
        PSNR=ComputePSNR(OutImp,OriImg); SSIM=ssim(OutImp,OriImg);
        [~,VIF]=evalc('vifvec(OutImp,OriImg)');%Not show log from vifvec
        IEF=ComputeIEF(OutImp,B,OriImg); MSSIM=multissim(OutImp,OriImg);
        if Log, disp("*PSNR="+PSNR+",SSIM="+SSIM+",VIF="+VIF+",IEF="+IEF+",MSSIM="+MSSIM+...
                "(TotalIteration="+iteration+",ExecutionTime="+ExecutionTime+"sec.)"); end
        tmp=[PSNR;SSIM;VIF;IEF;MSSIM;iteration;ExecutionTime];
        MetricsTmp=[MetricsTmp tmp];
        if Plot
            nexttile, imshow(B);title("Noise of "+num2str(nd*100)+"%");
            nexttile, imshow(OutImp);title({"Denoising of "+num2str(nd*100)+"%",...
                "("+PSNR+" dB, "+SSIM+", "+VIF+", "+IEF+", "+MSSIM+")"});
        end
    end
    MetricsTmp=[MetricsTmp mean(MetricsTmp,2)];
    MetricsResults=array2table(MetricsTmp,"VariableNames",NoisePercentage,"RowNames",MetricsList);
    if Log,disp("####Results of "+Method+" Method"+arg+" ("+ImgName+")####");disp(MetricsResults);end
end
function [OutImg,ExecutionTime,iteration]=ApplyIMF(B,Mode,Log)%Main IMF algorithm
%Mode=1: Literally follow the Flowchart&Algorithm1 in the paper(Stopping Criterion:SAD)
%Mode=2: Our Modified IMF (eqivalent to Mode=3)
%Mode=3: Use obfuscate P-Code file from the authors (They didn't reveal source code)
global OriImg %for PSNR calculation
    tic
    %if Log, disp("****Applying IMF (Mode="+Mode+")****");end
    if Mode==3 %Mode=3
        OutImg=IMF(B); iteration=NaN; ExecutionTime=toc;
        return
    end
    A=double(B);%Initialize A[0]=B
    r=1;%WS=2*r+1;%Window size (2*r+1)x(2*r+1)
    iteration=0;%k=0
    epsilon=0; stop=0; [m,n]=size(B); A_NextIter=zeros(m,n); %maxiteration=100;
    dmax=max(B(:)); dmin=min(B(:));
    A0_Padded=padarray(A,[r r],"replicate");%Replicate padding
    %Begin iterative loop
    while stop==0
        A_Padded=padarray(A,[r r],"replicate");%Replicate padding
        NoisePelLeft=size(A((A>=dmax)|(A<=dmin)),1);
        for i=1+r:m+r%Sliding window for all pixels
            for j=1+r:n+r
                aij=A_Padded(i,j);
                if Mode==2 %Mode=2
                    ak_ij=A0_Padded(i,j);%Aij[0]
                else %Mode=1
                    ak_ij=aij;%Aij[k]
                end
                if ak_ij>=dmax||ak_ij<=dmin
```

```
                    Wij=A_Padded(i-r:i+r,j-r:j+r);%Select mask from image
                    Wij_star=Wij((Wij<dmax)&(Wij>dmin));
                    if size(Wij_star,1)==0
                        A_NextIter(i-r,j-r)=aij;
                    else
                        A_NextIter(i-r,j-r)=round(mean(Wij_star));
                    end
                else
                    A_NextIter(i-r,j-r)=aij;
                end
            end
        end
        iteration=iteration+1;
        SAD=norm(A_NextIter-A,1);
        A=A_NextIter;
        %Stopping Criterion
        if Mode==2 %Mode=2
            %if (SAD<=epsilon)||(iteration>=maxiteration), stop=1;end
            if (NoisePelLeft<=0), stop=1;end
        else %Mode=1
            if SAD<=epsilon, stop=1;end
        end
        if Log
            disp("Iteration#"+iteration+"(k="+num2str(iteration-1)+"): "+...
                "#Noise pixels left="+NoisePelLeft+", SAD="+SAD+...
                ", PSNR="+ComputePSNR(uint8(A),OriImg))
        end
    end
    OutImg=uint8(A);
    ExecutionTime=toc;
end
function [OutImg,ExecutionTime,iteration]=ApplyProposed(B,p,Criterion,Log)%Main Proposed algorithm
global OriImg %for PSNR calculation
    tic
    %if Log, disp("****Applying Proposed (p="+p+")****");end
    A=double(B);%Initialize A[0]=B
    r=1;%Window size (2*r+1)x(2*r+1)
    iteration=0;%k=0
    stop=0; [m,n]=size(B); A_NextIter=zeros(m,n); maxiteration=100;
    dmax=max(B(:)); dmin=min(B(:));
    A0_Padded=padarray(A,[r r],"replicate");%Replicate padding
    epsilon=0.0015; ToTalPel=m*n;
    W=GetMinkowskiWeightMatrix(p,r); MADlast=Inf;
    %Begin iterative loop
    while stop==0
        A_Padded=padarray(A,[r r],"replicate");%Replicate padding
        NoisePelLeft=size(A((A>=dmax)|(A<=dmin)),1);
        for i=1+r:m+r%Sliding window for all pixels
            for j=1+r:n+r
                aij=A_Padded(i,j);
                ak_ij=A0_Padded(i,j);%Repeat on denoised pixels
                if ak_ij>=dmax||ak_ij<=dmin
                    Wij=A_Padded(i-r:i+r,j-r:j+r);%Select mask from image
                    Wij_star=Wij((Wij<dmax)&(Wij>dmin));
                    if size(Wij_star,1)==0
                        A_NextIter(i-r,j-r)=aij;
                    else
                        W_star=W((Wij<dmax)&(Wij>dmin));
                        A_NextIter(i-r,j-r)=round(sum(Wij_star.*W_star)/sum(W_star));
                    end
                else
                    A_NextIter(i-r,j-r)=aij;
                end
            end
        end
        iteration=iteration+1;
        MAD=norm(A_NextIter-A,1)/ToTalPel;
        absMADdiff=abs(MAD-MADlast);
        MADlast=MAD;
        A=A_NextIter;
        %Stopping Criterion
        if Criterion==1 %Mode=1
            if (absMADdiff<=epsilon&&NoisePelLeft<=0)||(iteration>=maxiteration),stop=1;end
            %if (MAD<=epsilon&&TotalNoisePel<=0)||(iteration>=maxiteration),stop=1;end
        else %Mode=2
            if (NoisePelLeft<=0),stop=1;end
        end
        if Log
            disp("Iteration#"+iteration+"(k="+num2str(iteration-1)+"): "+...
                "#Noise pixels left="+NoisePelLeft+", MAD="+MAD+...
                ", |MAD_diff|="+absMADdiff+", PSNR="+ComputePSNR(uint8(A),OriImg))
        end
    end
    OutImg=uint8(A);
    ExecutionTime=toc;
end
%Minkowski distance weighted matrix
function W=GetMinkowskiWeightMatrix(p,r)
    global W1 W2 W3 W4
    switch r
```

```
        case 1,W=W1;case 2,W=W2;case 3,W=W3;case 4,W=W4;
        otherwise,W=ComputeMinkowskiWeightMatrix(p,r);
    end
end
function W=ComputeMinkowskiWeightMatrix(p,r)%Main Minkowski distance weighted matrix calculation
    [i,j]=ndgrid(1:(2*r+1),1:(2*r+1));
    c=cat(3,i,j);%coordinate
    W=zeros(2*r+1,2*r+1);
    for i=1:(2*r+1)
        for j=1:(2*r+1)
            W(i,j)=1/pdist([c(i,j,1) c(i,j,2);(r+1) (r+1)],"minkowski",p);
        end
    end
    W(r+1,r+1)=0;
end
%Criteria Functions
function PSNR=ComputePSNR(img,Original)
    MSE=mean(mean((double(img)-double(Original)).^2));PSNR=10*log10((2^8-1)^2/MSE);
end
function IEF=ComputeIEF(img,noisy,Original)
    IEF=sum(sum((double(noisy)-double(Original)).^2))...
        /sum(sum((double(img)-double(Original)).^2));
end
function vif=vifvec(imdist,imorg)%Visual Information Fidelity - Pixel (VIF-P)
%H. R. Sheikh and A. C. Bovik, "Image information and visual quality,"
%IEEE Trans. Image Process., vol. 15, no. 2, pp. 430-444, Jan. 2006. Code downloaded from
%https://github.com/sattarab/image-quality-tools/tree/master/metrix_mux/metrix/vif/vifvec_release
%Prerequisites: matlabPyrTools, available at https://github.com/LabForComputationalVision/matlabPyrTools
    M=3;subbands=[4 7 10 13 16 19 22 25];sigma_nsq=0.4;
    %Do wavelet decomposition.
    [pyr,pind]=buildSpyr(imorg, 4, 'sp5Filters', 'reflect1');%Compute transform
    org=ind2wtree(pyr,pind);%Convert to cell array
    [pyr,pind]=buildSpyr(imdist, 4, 'sp5Filters', 'reflect1');dist=ind2wtree(pyr,pind);
    %Calculate the parameters of the distortion channel
    [g_all,vv_all]=vifsub_est_m(org,dist,subbands,M);
    %Calculate the parameters of the reference image
    [ssarr, larr, cuarr]=refparams_vecgsm(org,subbands,M);
    vvtemp=cell(1,max(subbands));ggtemp=vvtemp;%Reorder subbands
    for kk=1:length(subbands)
        vvtemp{subbands(kk)}=vv_all{kk};ggtemp{subbands(kk)}=g_all{kk};
    end
    %Compute reference and distorted image information from each subband
    for i=1:length(subbands)
        sub=subbands(i);g=ggtemp{sub};vv=vvtemp{sub};ss=ssarr{sub};lambda=larr(sub,:);cu=cuarr{sub};
        neigvals=length(lambda);%How many eigenvalues to sum over. default is all
        %Compute the size of the window used in the distortion channel estimation,
        %and use it to calculate the offset from subband borders
        lev=ceil((sub-1)/6);winsize=2^lev+1;offset=(winsize-1)/2;offset=ceil(offset/M);
        %Select only valid portion of the output.
        g=g(offset+1:end-offset,offset+1:end-offset);vv=vv(offset+1:end-offset,offset+1:end-offset);
        ss=ss(offset+1:end-offset,offset+1:end-offset);
        temp1=0;temp2=0;%VIF
        for j=1:length(lambda)
            %Distorted image information for the i'th subband
            temp1=temp1+sum(sum((log2(1+g.*g.*ss.*lambda(j)./(vv+sigma_nsq)))));
            temp2=temp2+sum(sum((log2(1+ss.*lambda(j)./(sigma_nsq)))));%Reference image information
        end
        num(i)=temp1;den(i)=temp2;
    end
    vif=sum(num)./sum(den);%Compuate VIF
end
function wtree=ind2wtree(pyr,ind)%Called by vifvec
    C=pyr;S=ind;offset=0;numsubs=size(ind,1);
    for i=1:numsubs%Converts the output of Eero Simoncelli's pyramid routines into subbands in a cell array
        wtree{numsubs-i+1}=reshape(C(offset+1:offset+prod(S(i,:))), S(i,1),S(i,2));offset=offset+prod(S(i,:));
    end
end
function [g_all,vv_all]=vifsub_est_m(org,dist,subbands,M)%Called by vifvec
    %Uses convolution for determining the parameters of the distortion channel
    tol=1e-15;%Tolernace for zero variance
    for i=1:length(subbands)
        sub=subbands(i);y=org{sub};yn=dist{sub};
        %Compute the size of the window used in the distortion channel estimation
        lev=ceil((sub-1)/6);winsize=2^lev+1;offset=(winsize-1)/2;win=ones(winsize);
        %Force subband size to be multiple of M
        newsize=floor(size(y)./M)*M;y=y(1:newsize(1),1:newsize(2));yn=yn(1:newsize(1),1:newsize(2));
        %Correlation with downsampling
        winstep=[M M];winstart=[1 1].*floor(M/2)+1;winstop=size(y)-ceil(M/2)+1;
        mean_x=corrDn(y,win/sum(win(:)),'reflect1',winstep,winstart,winstop);
        mean_y=corrDn(yn,win/sum(win(:)),'reflect1',winstep,winstart,winstop);
        cov_xy=corrDn(y.*yn,win,'reflect1',winstep,winstart,winstop)-sum(win(:)).*mean_x.*mean_y;
        ss_x=corrDn(y.^2,win,'reflect1',winstep,winstart,winstop)-sum(win(:)).*mean_x.^2;
        ss_y=corrDn(yn.^2,win,'reflect1',winstep,winstart,winstop)-sum(win(:)).*mean_y.^2;
        ss_x(ss_x<0)=0;ss_y(ss_y<0)=0;%Get rid of numerical problems, very small numbers
        g=cov_xy./(ss_x+tol);%Regression
        vv=(ss_y-g.*cov_xy)/(sum(win(:)));%Variance of error in regression
        %Get rid of numerical problems, very small numbers
        g(ss_x<tol)=0;vv(ss_x<tol)=ss_y(ss_x<tol);ss_x(ss_x<tol)=0;g(ss_y<tol)=0;vv(ss_y<tol)=0;
        vv(g<0)=ss_y(g<0);g(g<0)=0;%Constrain g to be non-negative
        vv(vv<=tol)=tol;%Take care of numerical errors, vv could be very small negative
```

```
        g_all{i}=g;vv_all{i}=vv;
    end
end
function [ssarr,l_arr,cu_arr]=refparams_vecgsm(org,subands,M)%Called by vifvec
    %Computes the parameters of the reference image
    for i=1:length(subands)
        sub=subands(i);y=org{sub};
        sizey=floor(size(y)./M)*M;%Crop to exact multiple size
        y=y(1:sizey(1),1:sizey(2));temp=[];
        %Collect MxM blocks, Rearrange each block into an M^2 dimensional vector and collect all
        %Collece ALL possible MXM blocks (even those overlapping) from the subband
        for j=1:M
            for k=1:M
                temp=cat(1,temp,reshape(y(k:end-(M-k), j:end-(M-j)),1,[]));
            end
        end
        %Estimate mean and covariance
        mcu=mean(temp')';cu=((temp-repmat(mcu,1,size(temp,2)))*(temp-repmat(mcu,1,size(temp,2)))')./size(temp,2);
        %Collect MxM blocks as above. Use ONLY non-overlapping blocks to calculate the S field
        temp=[];
        for j=1:M
            for k=1:M
                temp=cat(1,temp,reshape(y(k:M:end, j:M:end),1,[]));
            end
        end
        ss=(inv(cu)*temp);ss=sum(ss.*temp)./(M*M);ss=reshape(ss,sizey/M);%Calculate the S field
        [~,d]=eig(cu);l_arr(sub,:)=diag(d)';%Eigen-decomposition
        ssarr{sub}=ss;temp=0;d=diag(d);cu_arr{sub}=cu;%Rearrange for output
    end
end
```