



저작자표시 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.
- 이 저작물을 영리 목적으로 이용할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#) 

Master's Thesis

<Reservoir computing for salt-and-pepper noise
removal with a small dataset>

<In Mo Lee>

<Department of Mathematical Science>

Ulsan National Institute of Science and Technology

<2023>

<Reservoir computing for salt-and-pepper noise
removal with a small dataset>

<In Mo Lee>

<Department of Mathematical Science>

Ulsan National Institute of Science and Technology

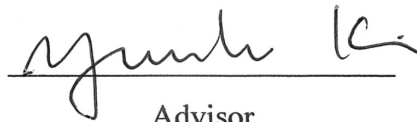
Reservoir computing for salt-and-pepper noise removal with a small dataset

A thesis/dissertation submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Master of Science

Inmo Lee

06.08.2023 of submission

Approved by



Advisor

Yunho Kim

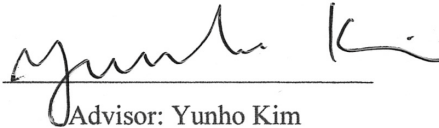
Reservoir computing for salt-and-pepper noise removal with a small dataset

Inmo Lee

This certifies that the thesis/dissertation of Inmo Lee is approved.

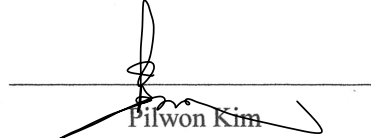
06.08.2023 of submission

Signature



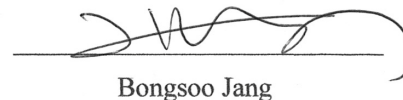
Advisor: Yunho Kim

Signature



Pilwon Kim

Signature



Bongsoo Jang

Signature

*three signatures total in case of masters

Abstract

We propose a reservoir computing inspiring neural network approach for salt-and-pepper noise removal. Despite the fact that reservoir computing was derived from RNNs dealing with sequential data and that images are not sequential data, our clever reservoir design has made it possible for the reservoir to process image data through a nonlinear image-specific forward operator replacing the linear operator of the multiplication by a randomly chosen input weight matrix. There are two essential advantages of the proposed method. One is that it takes only a small amount of training data as many as a few hundreds of 24×24 images and outperforms most analytic or machine-learning based denoising models for salt-and-pepper noise unlike most neural networks requiring a large amount of training data for good results to avoid overfitting, if not the best. Another is that the reconstruction is completely parallel, in that noisy pixels do not communicate with each other, and hence, noise in different pixels can be removed in parallel. Recursive reservoir architecture is also discussed to further improve the reconstruction quality, confirmed by various numerical simulations.

Contents

I	Introduction	1
II	Theory	2
2.1	Recurrent Neural Network(RNN)	2
2.2	Reservoir Computing(RC)	3
2.3	Echo State Network(ESN)	3
2.4	Filters to restore noisy images	6
III	Image Restoration	7
3.1	Impulse noise(also called salt-and pepper noise)	7
3.2	Finding W_{in} and a reservoir for a patch	7
3.3	Finding a reservoir state matrix for n-th reservoir	9
3.4	Finding W_{out} for n-th reservoir	9
3.5	Image restoration	10
IV	Experiment	11
4.1	The result depending on patch sizes	12
4.2	Comparing the existed methods	14
4.3	The result with the reservoir state matrix sequential concatenation and no leaky integrator neurons	19
4.4	The result depending on the number of training datasets	21

V	Conclusion	23
	References	36
	Acknowledgements	42

List of Figures

1	The structure of RNN.	2
2	The structure of RC. [1]	3
3	The structure of ESN. [2]	4
4	Salt and Pepper noise	8
5	The structure of the reservoirs to restore noisy image.	10
6	The result images depending on the pixel sizes.(Pyramid)	12
7	The result images depending on the pixel sizes.(Castle)	13
8	The parts of the result when taking 5×5 pixel	14
9	CNN with median filter [3]	14
10	The result images with 80% noise.(Pyramid)	15
11	The result images with 80% noise.(Pyramid)	16
12	The result images with 80% noise.(Castle)	16
13	The result images with 80% noise.(Castle)	17
14	The training error with 60% noise.	19
15	The training error with 80% noise.	20
16	The result comparison depending on the number of training datasets. (Castle, 60% noise)	21
17	The result comparison depending on the number of training datasets. (Castle, 80% noise)	22

18	The result comparison depending on the number of training datasets. (Pyramid, 60% noise)	22
19	The result comparison depending on the number of training datasets. (Pyramid, 80% noise)	23
20	The result images depending on the pixel sizes.(Airplane)	24
21	The result images depending on the pixel sizes.(Cheetah)	25
22	The result images depending on the pixel sizes.(Status)	25
23	The result images depending on the pixel sizes.(Butterfly)	26
24	The result images depending on the pixel sizes.(Camel)	26
25	The result images with 80% noise.(Airplane)	27
26	The result images with 80% noise.(Airplane)	27
27	The result images with 80% noise.(Cheetah)	28
28	The result images with 80% noise.(Cheetah)	28
29	The result images with 80% noise.(Butterfly)	29
30	The result images with 80% noise.(Butterfly)	29
31	The result images with 80% noise.(Status)	30
32	The result images with 80% noise.(Camel)	31
33	The result comparison depending on the number of training datasets. (Airplane)	35
34	The result comparison depending on the number of training datasets. (Cheetah)	36
35	The result comparison depending on the number of training datasets. (Butterfly)	37
36	The result comparison depending on the number of training datasets. (Statue)	38
37	The result comparison depending on the number of training datasets. (Camel)	39

I Introduction

In order to research about image restoration, image degradation is used. By giving some noises on the images, we research how to restore these noisy images. Image degradation is the behavior that gives images some noise or blur. The examples are salt-and-pepper noise, Gaussian noise using Gaussian distribution, Periodic noise, Exponential noise and so on.

Image restoration is the behavior that removes from blurry or noisy images. Also, image restoration has developed for decades such as Hunt's paper [4], using Bayesian-based method [5] and so on. Many people developed filters to restore noisy images. Median filter is effective in removing salt-and-pepper noise, Midpoint filter is effective in removing Gaussian noise and Uniform noise, and Mean filter is effective in removing Gaussian noise and uniform noise. But the limit appears when the noise is high, especially median filter is ineffective in removing salt-and-pepper noise when 40 % or more noise is given.

Echo state network([2]) is a type of Reservoir Computing([1]) that is derived from Recurrent Neural Network(RNN, [6]). Unlike other RNNs as well as other ANN(Artificial Neural Network), Reservoir Computing including ESN only trained output weights. And other weights including input weights are randomly selected. RNN is used to train weights through Back-propagation([6]). But Reservoir Computing uses no back-propagation, but least square method. Because of these differences, the computational cost of RC including ESN is cheaper than that of RNNs.

Nowadays, Image-related works such as restoration, enhancement, segmentation and so on mainly use convolutional neural network(CNN). Convolutional Neural Network mainly uses many convolutional filters such as ResNet([7]), AlexNet [8], VGGNet([9]) and so on. However, CNN has some problems about computational cost as well as other Artificial Neural Networks. And because CNN uses too many parameters and back-propagations([6]), the cost of CNN is so expensive.

Because Echo State Network has not been used in image-related work, we try to apply ESN to restore noisy images to solve this problem such as expensive cost problem. So we expect ESN can also apply image-related works such as image restoration and solve the computational cost problem.

Chapter 2 introduces theories that are used in paper. we explain about Recurrent Neural Network(RNN), Reservoir Computing(RC), Echo State Network(ESN) and some filters that is used in paper. Chapter 3 explains how to restore the noisy image. Before training, input images are given salt-and-pepper noise and we apply Echo State Network to restore them. Chapter 4 compares previous methods to show that ESN can be also used in image-related work. And Chapter 5 refer some limitations and future works.

II Theory

We introduce the theories that uses to restore noisy images. First, we explain RNN, RC and ESN. Second, we explain filters that is used.

2.1 Recurrent Neural Network(RNN)

RNN is the one of the artificial neural network (ANN) and mainly used for text generation and speech recognition. The architecture have cycles where the model feeds the network outputs from a previous step as an input to the network in order to take into account historical information. And there are many type of RNNs structures such as one-to-many(image captioning), many-to-one(sentimental analysis, spam-mail classification), and many-to-many(chatbot, translator). The main expression is like this.

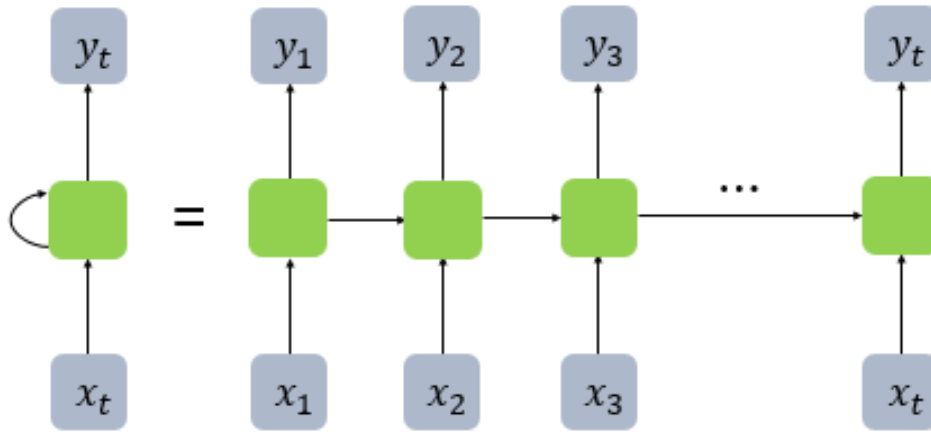


Figure 1: The structure of RNN.

$$\begin{aligned}
 h_t &= f_h(W_h x_t + U_h h_{t-1}) \\
 y_t &= f_y(W_y h_t)
 \end{aligned}
 \tag{1}$$

Where x_t is the input, h_t is the hidden layer, y_t is the output, W_h , U_h and W_y are the weights, f_h and f_y are the activation functions (for example, hyperbolic tangent or sigmoid), and b_h and b_y are the biases. Like the other ANN(for example, CNN(Convolutional Neural Network)), RNN also uses back-propagation [6] to update weights(W_h , U_h and W_y) for every epoch. Back-propagation is the method using gradient of the loss function with respect to weights. But the limit is the expensive computational cost, slow convergence and the gradient decending. Because Back-Propagation uses differentiation and many chain rules. Because of those, the application of RNN is not feasible. To overcome these shortage, improved RNNs are developed: LSTM (Long Short-Term Memory) [10], GRU (Gated Recurrent Unit) [11]. Also, RC (Reservoir Computing) is derived from RNN, but there are differences from other RNNs.

2.2 Reservoir Computing(RC)

Reservoir Computing(RC) consists of a reservoir and a readout mapping. 'Reservoir' means 'a place where a liquid is stored'. And the word 'liquid' means the hidden layer vector at RNN. So we call this liquid 'reservoir state vector'. The expression is like this.

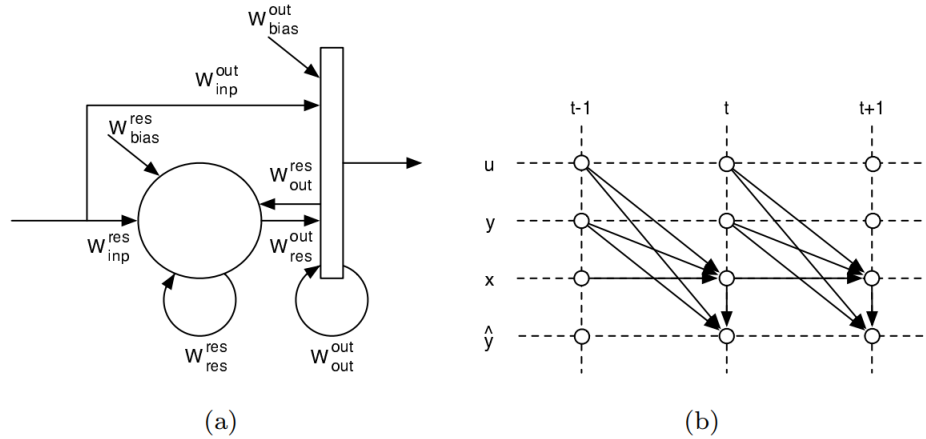


Figure 2: The structure of RC. [1]

$$\begin{aligned} h_{t+1} &= f_h(W_h x_t + U_h h_t + V_h y_t + b_h) \\ y_{t+1} &= W_y x_{t+1} + U_y h_t + V_y y_t + b_y \end{aligned} \quad (2)$$

where W_h , U_h and V_h are weights to the reservoir(input weights) and W_y , U_y and V_y are weights to the output(output weights). Also, h_t is called a 'Reservoir State Vector'.

As we can see, the structure is similar to RNN. But there are differences between RC and other RNNs. First, RNNs train all the weight including input weights, but RC trains only output weights. Instead, the input weights on RC are initialized randomly. Second, when we train weights, RNN uses back-propagation that is used in other neural networks. But RC uses a simple method such as least square method. Because the weights on the recurrent connections in the reservoir are not necessarily trained. These differences cause the difference of computational costs. Because RNNs use all the weights in training and back-propagation that uses many chain rules, the cost is expensive. On the other hand, RC uses only the output weights in training and least square method, so the cost is cheap. The examples are ESN (Echo State Network) and LSM (Liquid State Machine).

2.3 Echo State Network(ESN)

'Echo' means 'something that repeats or resembles something else' or 'something that is similar to something that happened or existed before'. So, 'Echo State' means 'if the network has been run for a very long time, the current network state is uniquely determined by the history of the input and the (teacher-forced) output'. ESN is a type of RC, and also belongs to RNN. So the expression is similar to RNN and RC.

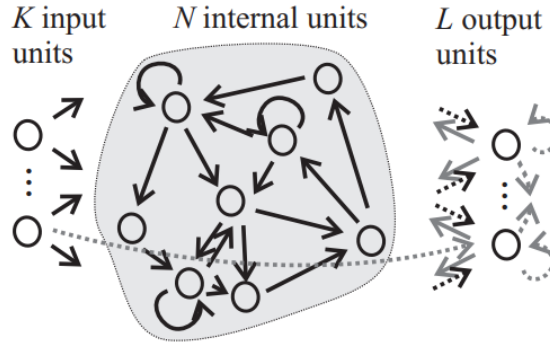


Figure 3: The structure of ESN. [2]

$$\begin{aligned}
 h_{t+1} &= f_h(W_{in}x_t + Wh_t + W_{back}y_t) \\
 y_{t+1} &= W_{out} \begin{bmatrix} x_t \\ h_{t+1} \\ y_t \end{bmatrix}
 \end{aligned} \tag{3}$$

where W_{out} is the weight to the output that is trained. And $\begin{bmatrix} x_t \\ h_{t+1} \\ y_t \end{bmatrix}$ is the concatenation of column vectors. And the row vector version is like this.

$$\begin{aligned}
 h_{t+1}^T &= f_h(x_t^T W_{in} + h_t^T W + y_t^T W_{back}) \\
 y_{t+1}^T &= \begin{bmatrix} x_t^T & h_{t+1}^T & y_t^T \end{bmatrix} W_{out}
 \end{aligned} \tag{4}$$

Echo state network finds W_{out} using the least square error.

$$\begin{aligned}
 \hat{W}_{out} &= \arg \min_{W_{out}} \|Y - XW_{out}\|_2^2 \\
 \hat{W}_{out} &= (X^T X)^\dagger X^T Y \\
 X &= \begin{bmatrix} x_1^T & h_2^T & y_1^T \\ x_2^T & h_3 & y_2^T \\ \dots & \dots & \dots \\ x_{T-1}^T & h_T^T & y_{T-1}^T \end{bmatrix} \\
 Y &= \begin{bmatrix} y_2^T \\ y_3^T \\ \dots \\ y_T^T \end{bmatrix}
 \end{aligned} \tag{5}$$

where \hat{y}_t is the ground truth output vector and the matrix X^\dagger means the pseudoinverse of the matrix X . In order to determine echo state, these propositions are used. Especially, these propositions are easy to check the echo state.

Proposition 1 Assume a sigmoid network with unit output functions $f_i = \tanh$.

(a) Let the weight matrix W satisfy $\sigma_{\max} = \Lambda < 1$, where σ_{\max} is its largest singular value. Then $d(T(x, h), T(x', h')) < \Lambda d(h, h')$ for all inputs x , for all states $h, h' \in [-1, 1]^N$. There, T means the network state update operator: $T(x_t, h_t) = f(W_{in}x_t + Wh_t)$

(b) Let the weight matrix have a spectral radius $|\lambda_{\max}| > 1$, where λ_{\max} is an eigenvalue of W with the largest absolute value. Then the network has an asymptotically unstable null state. This implies that it has no echo states for any input set U containing 0 and admissible state set $A = [-1, 1]^N$.

So according to (b) in the proposition, the spectral radius of W must be smaller than 1 to maintain the echo state. So, we should determine the weight matrix W like this.

Algorithm 1 The weight Algorithm.

1: Randomly generate W_0 by sampling the weights from a uniform distribution over $[0, 1]$

2: Normalize W_0 to W_1 with unit spectral radius λ_{\max} of W_0

$$W_1 = \frac{1}{\lambda} W_0$$

3: Scale W_1 to W , whereby W has a spectral radius of $\rho < 1$

$$W = \rho W_1$$

ESNs with leaky integrator neurons

Echo State Network with leaky integrator neurons is to learn slowly and continuous changing systems. So we introduce networks with a continuous dynamics. The evolution of a continuous-time leaky integrator network is like this,

$$\dot{h} = C(-ah + f(W^{in}x + Wh + W^{back}y)) \quad (6)$$

where C is the time constant and a is the leaking decay rate. So, this solution is like this.

$$h_{n+1} = (1 - \delta Ca)h_n + \delta C f(W^{in}x_{n+1} + Wh_n + W^{back}y_n) \quad (7)$$

where δ is the stepsize. By choosing C small, one can employ the Echo State Network of type (8) to model slow systems. Or we can write if we take the leaking decay rate $a = 1$.

$$h_{n+1} = \alpha h_n + (1 - \alpha) f(W^{in}x_{n+1} + Wh_n + W^{back}y_n) \quad (8)$$

2.4 Filters to restore noisy images

Before we introduce filters that we restore noisy images, we explain the meaning of 'Adaptive' filter. [12] Unlike other filters, 'Adaptive' filter only uses not-noisy pixels since it is appropriate in removing salt-and-pepper noise. So we define the pixel set M , which means the set of not-noisy pixels when giving salt-and-pepper noises,

$$M_{(i,j)} = \{(s,t) | 0 < [I_1]_{(i+s,j+t)} < 1\} \quad (9)$$

where we use the pixel values the interval $[0, 1]$.

Adaptive mean filter

Mean filter is the filter that used mean value around a pixel. But adaptive mean filter is excluded from the noisy pixels. The expression is like this.

$$I'_{(i,j)} = \frac{1}{m} \sum_{(s,t) \in M_{(i,j)}} I_{(i+s,j+t)} \quad (10)$$

$$m = |M_{(i,j)}|$$

where $|A|$ means the number of elements of the set A .

Adaptive gaussian filter

Gaussian filter uses the Gaussian distribution to make the filter. Adaptive gaussian filter is also excluded from the noisy pixels.

$$I'_{(i,j)} = \frac{1}{g} \sum_{(s,t) \in M_{(i,j)}} G(s,t) I_{(i+s,j+t)}$$

$$g = \sum_{(s,t) \in M_{(i,j)}} G(s,t) \quad (11)$$

$$G(s,t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{s^2+t^2}{2}}$$

Adaptive unsharp masking filter

Unsharp masking filter is the filter that enhances the edges. And Unsharp masking filter is made by subtracting blurred image from the original image. Adaptive unsharp masking filter is also excluded from the noisy pixels. There, We use Adaptive Unsharp masking filter with Gaussian distribution($G(s,t)$) for a blurred image. The expression is like this.

$$\begin{aligned}
I'_{(i,j)} &= \frac{1}{u} \sum_{(s,t) \in M(i,j)} U(s,t) I_{(i+s,j+t)} \\
u &= \sum_{(s,t) \in M(i,j)} G(s,t) \\
U(s,t) &= \begin{cases} -G(s,t) & \text{if } (s,t) \neq 0 \\ \sum_{s=-l}^l \sum_{t=-l}^l G(s,t) & \text{if } (s,t) = 0 \end{cases}
\end{aligned} \tag{12}$$

These filters will be used to find W_{in} for a patch.

III Image Restoration

We explain how to restore the noisy images. We use the salt-and-pepper noise to give noisy images. And we use ESN to restore noisy images.

3.1 Impulse noise(also called salt-and pepper noise)

We used Noise matrix N whose element is 1 where the corresponding pixel is noisy and 0 otherwise.

$$N_{(i,j,c)} = \begin{cases} 1 & \text{if } (I_1)_{(i,j,c)} = 1 \text{ or } 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

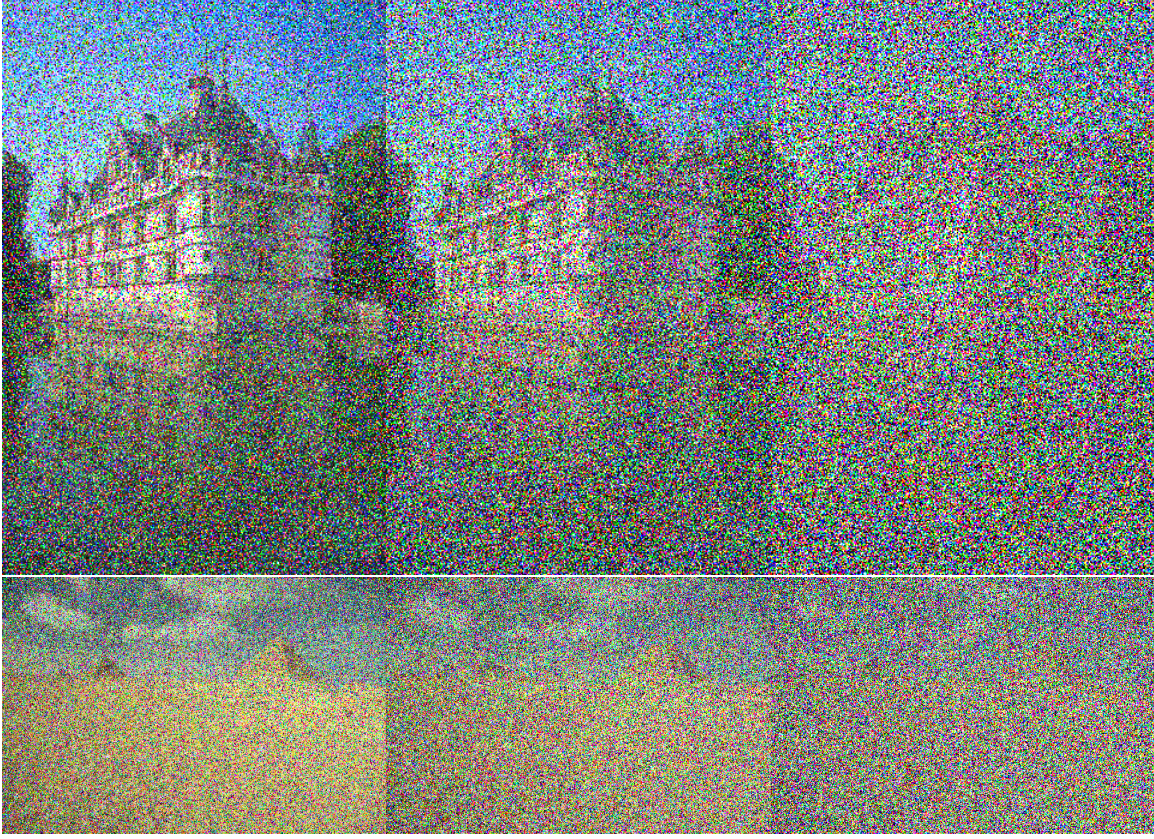
where c means the colors, red($c = 1$), green($c = 2$) and blue($c = 3$).

Next, we apply the noise partition. This means we divide the noise pixels evenly and randomly, say N_1 and N_2 . There, N_1 and N_2 are used to apply the adaptive filters to make W_{in} . And N_1 is used for the 1st reservoir, N_2 is used for the 2nd reservoir, and N is used for the other reservoirs.

And we use the noisy input image I_1 for the 1st reservoir, and the output image I_n that is generated by the previous reservoir is used to the input.

3.2 Finding W_{in} and a reservoir for a patch

We make ' W_{in} ' for a $(2p+1) \times (2p+1)$ patch using adaptive mean, adaptive gaussian and adaptive unsharp masking filter.(Section 2.4) And the size of ' W_{in} ' for a patch is $((2p+1) * (2p+1) * 3) \times (3 * 3 + 1) = ((2p+1) * (2p+1) * 3) \times 10$. So we make the element of $(W_{in})_{(i,j)}$ the coefficient of filters. [12]



(d) 40% noise

(e) 60% noise

(f) 80% noise

Figure 4: Salt and Pepper noise

$$\begin{aligned}
 [(W_{in})_{(i,j)}]_{((2p+1)(2p+1)(c-1)+(2p+1)(s+2)+t+3,c)} &= \begin{cases} \frac{1}{m} & \text{if } N_{(i,j,c)} = 0 \\ 0 & \text{otherwise} \end{cases} \\
 [(W_{in})_{(i,j)}]_{((2p+1)(2p+1)(c-1)+(2p+1)(s+2)+t+3,3+c)} &= \begin{cases} \frac{1}{g}G(s,t) & \text{if } N_{(i,j,c)} = 0 \\ 0 & \text{otherwise} \end{cases} \\
 [(W_{in})_{(i,j)}]_{((2p+1)(2p+1)(c-1)+(2p+1)(s+2)+t+3,6+c)} &= \begin{cases} \frac{1}{u}U(s,t) & \text{if } N_{(i,j,c)} = 0 \\ 0 & \text{otherwise} \end{cases} \\
 [(W_{in})_{(i,j)}]_{((2p+1)(2p+1)(c-1)+(2p+1)(s+2)+t+3,10)} &= 0
 \end{aligned} \tag{14}$$

In 2nd reservoir, we use N_2 instead of N . And from 3rd reservoir, we don't use the adaptive filter. Because in I_3 that is used for the 3rd reservoir, there are few noises. So we find W_{in} like there is no (i, j, c) such that $N_{(i,j,c)} = 0$. Then, we can make a reservoir state vector for a patch and we use an activation function to hyperbolic tangent function(\tanh). And we refer to the equation (4). Since there are no input and output vectors, we eliminate input and output terms, but remain the reservoir state term. So using (8), we can make the next state vector for the patch to make many vectors. And we concatenate the vectors, we make a reservoir state vector for the patch.

$$\begin{aligned}
[r_{(i,j)}]_1 &= \tanh(b + [(W_{in})_{(i,j)}][L_n]_{(i,j)}) \\
[r_{(i,j)}]_{k+1} &= \alpha[r_{(i,j)}]_k + (1 - \alpha) \tanh\left(\frac{0.5}{\tanh 0.5} [r_{(i,j)}]_k A\right) \\
r_{(i,j)} &= [[r_{(i,j)}]_1, [r_{(i,j)}]_2, \dots, [r_{(i,j)}]_K]_{1 \times (10K)}
\end{aligned} \tag{15}$$

where $[L_n]_{(i,j)}$ is a $(2p+1) \times (2p+1)$ input patch whose center is (i, j) , $\alpha \in [0.1]$ is the coefficient and A is the weight matrix. This matrix is made from Algorithm 1

3.3 Finding a reservoir state matrix for n-th reservoir

We made many reservoir state vectors for patches respectively. Next We concatenate the vectors to make a reservoir state matrix for an input image. So we call this matrix 'genR_n'. Then, We make a reservoir state matrix (R_n) for an image using the reservoir state matrix for an input image (genR_n) and the previous reservoir state matrix for an image. We use this equation (8). Since the output image (restored image) is used as the next reservoir input image, we don't need to use 'W_{back}'. And since the input image is used to find W_{in} s and $r_{(i,j)}$ s for patches, the term $W_{in}x_t$ is eliminated.

$$genR_n = \begin{bmatrix} R_{(p+1,p+1)} \\ R_{(p+1,p+2)} \\ \dots \\ R_{(X-p,Y-p)} \end{bmatrix}_{((X-2p)(Y-2p)) \times (10K)} \tag{16}$$

$$R_1 = genR_1$$

$$R_n = \beta genR_n + (1 - \beta) \tanh\left\{\frac{0.5}{\tanh 0.5} genR_n B_n\right\}$$

where $\beta \in [0, 1]$ is the coefficient, and B_n is the weight matrices that are made from Algorithm 1. And X and Y are the size of input image.

3.4 Finding W_{out} for n-th reservoir

Next, we find 'W_{out}' to use test images. And we use the least square method to find 'W_{out}'. The error function is like this.

$$E((W_{out})_n) = \sum_{l=1}^L \|Z^l - R_n^l (W_{out})_n\|^2 \tag{17}$$

To find 'W_{out}', we just differentiate this and find 'W_{out}' that equals to 0. So we can find 'W_{out}' like this.

$$\begin{aligned}
\frac{\partial E((W_{out})_n)}{\partial (W_{out})_n} &= -2 \sum_{l=1}^L (R_n^l)^T Z^l + 2 \left\{ \sum_{l=1}^L (R_n^l)^T R_n^l \right\} (W_{out})_n \\
\Rightarrow (W_{out})_n &= \left\{ \sum_{l=1}^L (R_n^l)^T R_n^l \right\}^\dagger \left\{ \sum_{l=1}^L (R_n^l)^T Z^l \right\}
\end{aligned} \tag{18}$$

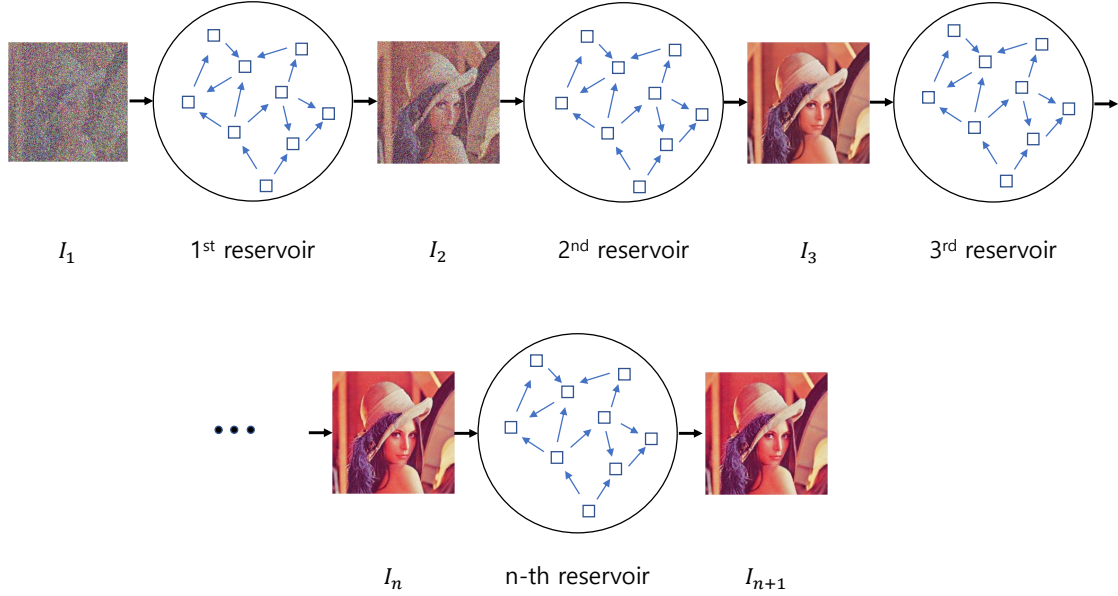


Figure 5: The structure of the reservoirs to restore noisy image.

where A^\dagger is the pseudoinverse of a matrix A and Z^l is the ground truth image.

3.5 Image restoration

Finally, we can restore the noisy image. Since we know the noisy pixels and the non-noisy pixels in the first input, we only restore the noisy parts. But we don't need to restore the non-noisy parts. And the restored image is used into the input for the next reservoir.

$$\begin{aligned}
 I_2 &= (1 - N_1) \odot I_1 + N_1 \odot R_1^l(W_{out})_1 \\
 I_3 &= (1 - N_2) \odot I_2 + N_2 \odot R_2^l(W_{out})_2 \\
 I_{n+1} &= (1 - N) \odot I_n + N \odot R_n^l(W_{out})_n
 \end{aligned} \tag{19}$$

where ' \odot ' means the element-wise multiplication between matrices. The term ' $(1 - N) \odot I_n$ ' means that the pixel values of non-noisy parts remain, but the term ' $N \odot R_n^l(W_{out})_n$ ' means that the noisy pixels are modified to the restored pixels. And the output image I_{n+1} is used to the next reservoir input.

Reservoir State Matrices Sequential Concatenation

We also consider the reservoir state matrices sequential concatenation. This method is just to concatenate all the previous state matrices. The expression is like this

$$\begin{aligned}
 RC_1 &= R_1 \\
 RC_n &= \begin{bmatrix} R_1 & R_2 & \dots & R_n \end{bmatrix}
 \end{aligned} \tag{20}$$

So, the restored image is like this.

$$\begin{aligned}
(W_{out})_n &= \left\{ \sum_{l=1}^L (RC_n^l)^T RC_n^l \right\}^\dagger \left\{ \sum_{l=1}^L (RC_n^l)^T Z^l \right\} \\
I_2 &= (1 - N_1) \odot I_1 + N \odot RC_1^l(W_{out})_1 \\
I_3 &= (1 - N_2) \odot I_2 + N \odot RC_2^l(W_{out})_2 \\
I_{n+1} &= (1 - N) \odot I_n + N \odot RC_n^l(W_{out})_n
\end{aligned} \tag{21}$$

No leaky integrator neurons

We also consider without leaky integrator neurons. That is, we also use the original expression of ESN4. There, we also eliminate the weight matrices.

$$R_n = \tanh \left\{ \frac{0.5}{\tanh 0.5} \text{gen} R_n \right\} \tag{22}$$

IV Experiment

We use the error analysis, PSNR, RMSE and SSIM to compare existed methods. PSNR is Peak Signal-Noise Ratio, RMSE is Root Mean Square Error, and SSIM is Structural Similarity Index Measure. These are mainly used to estimate the error of the images.

$$\begin{aligned}
MSE(Z, I_n) &= \frac{1}{3XY} \sum_{i=1}^X \sum_{j=1}^Y \sum_{k=1}^3 (Z_{(i,j,k)} - (I_n)_{(i,j,k)})^2 \\
PSNR(Z, I_n) &= -10 \log_{10} MSE(Z, I_n) \\
RMSE(Z, I_n) &= \sqrt{MSE(Z, I_n)} \\
SSIM(Z, I_n) &= \frac{(2\mu_Z \mu_{I_n} + c_1)(2\sigma_{(Z, I_n)} + c_2)}{(\mu_Z^2 + \mu_{I_n}^2 + c_1)(\sigma_Z^2 + \sigma_{I_n}^2 + c_2)} \\
\mu_I &= \frac{1}{3XY} \sum_{i=1}^X \sum_{j=1}^Y \sum_{c=1}^3 I_{(i,j,c)} \\
\sigma_I &= \frac{1}{3XY - 1} \sum_{i=1}^X \sum_{j=1}^Y \sum_{c=1}^3 (I_{(i,j,c)} - \mu_I)^2 \\
\sigma_{(I, J)} &= \frac{1}{3XY - 1} \sum_{i=1}^X \sum_{j=1}^Y \sum_{c=1}^3 (I_{(i,j,c)} - \mu_I)(J_{(i,j,c)} - \mu_J) \\
c_1 &= 0.01^2 \\
c_2 &= 0.03^2
\end{aligned} \tag{23}$$

We set the number of reservoirs $10(n = 1, 2, \dots, 10)$, $\alpha = 0.5$, $\beta = 0.9$, $\rho = 0.9$ (from Algorithm 1) and $K = 8$ (from the equation (15)). And we use 200 training images ($L = 200$) whose sizes are all 24×24 ($X = Y = 24$ from the equation (16)) to find $(W_{out})_n$.

4.1 The result depending on patch sizes

First, we implement depending on the patch sizes : 5×5 , 7×7 and 9×9 . And we select the 6th reservoir because test images' error results are almost no better from 6th reservoir. (Figure 6-7, Appendix Figure 20-24)

		60% noise			80% noise		
		5×5	7×7	9×9	5×5	7×7	9×9
castle	PSNR	31.4036	30.9735	30.7163	26.7905	26.6868	26.1899
	RMSE	0.0269	0.0283	0.0291	0.0458	0.0463	0.0490
	SSIM	0.9761	0.9734	0.9715	0.9253	0.9295	0.9205
pyramid	PSNR	34.3632	34.8627	34.6315	31.2212	32.7827	30.9204
	RMSE	0.0191	0.0181	0.0186	0.0275	0.0230	0.0284
	SSIM	0.9897	0.9908	0.9896	0.9694	0.9843	0.9763

Table 1: The result depending on patch sizes

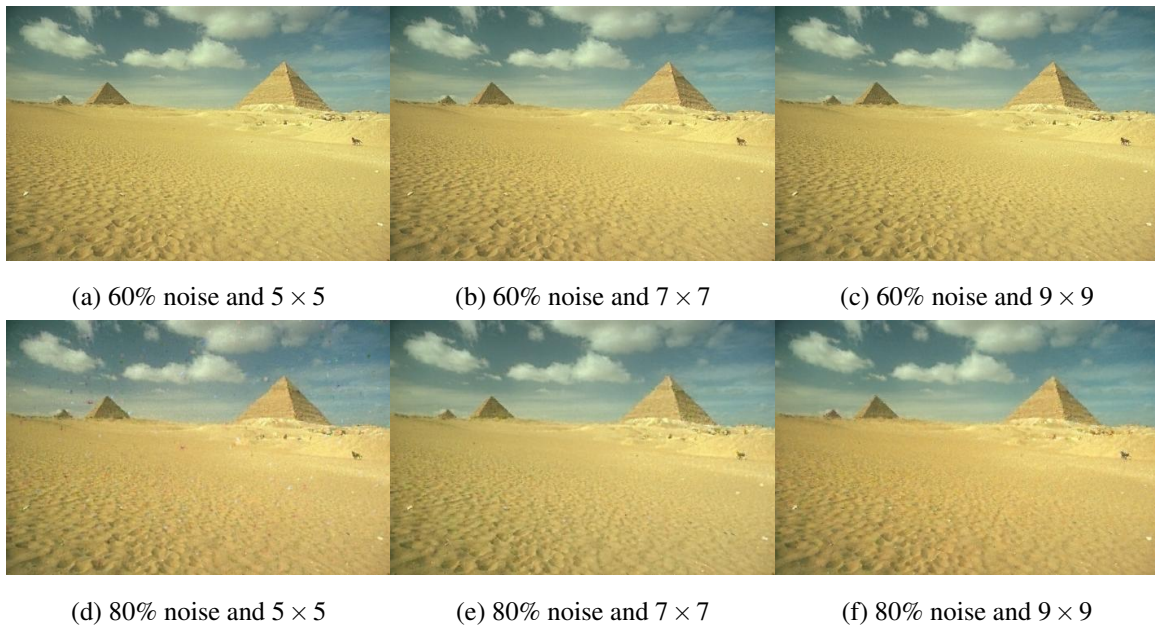


Figure 6: The result images depending on the pixel sizes.(Pyramid)

(a) 60% noise and 5×5 (b) 60% noise and 7×7 (c) 60% noise and 9×9 (d) 80% noise and 5×5 (e) 80% noise and 7×7 (f) 80% noise and 9×9

Figure 7: The result images depending on the pixel sizes.(Castle)

We can find when the noise is 60%, 5×5 patch looks best on Table 1 and Appendix Table 5. In fact, many papers use the smaller filter. That's because the smaller the filter size, the less size of input weights for patches respectively. So, the computational cost is cheaper and the performance is better. However, Although the Table 1 shows that there are some cases that the best errors(PSNR, RMSE, SSIM) are 5×5 patch, the figure (d) in Figure 6 - 7 and Appendix Figure 20-24 is found noisy by the naked eye, whereas the cases of 7×7 and 9×9 looks clean.

That's because if we take 5×5 patch, then the number of noise pixels is 20 of 25 on average. So, we can find that there are some patches that have all the noise pixels. Because we use adaptive filters to find W_{in} , the patches cannot apply the adaptive filters. So, these patches cause the remaining noise after restoration. On the other hand, if we expand the patch such as 7×7 and 9×9 , then there are few patches that have all the noise pixels. So we can apply the adaptive filters better than the case about 5×5 patch, and there are few noises on the result image((e) and (f) in Figure 6 - 7 and Appendix Figure 20-24).

Therefore, we adopt the case about 5×5 patch when the noise intensity is 60% and the case about



Figure 8: The parts of the result when taking 5×5 pixel

7×7 patch when the noise intensity is 80% on the next experiment.

4.2 Comparing the existed methods

Second, we compare other restoration methods like median filter, mean filter, gaussian filter, adaptive mean filter, adaptive gaussian filter, CNN with median filter [3], image restoration PDE and Kim's [12]. And we adopt the RSMSC, reservoir state matrices sequential concatenation. (Table 2 - ??) (Figure 13 - 30)

In the past, median filter, mean filter and gaussian filter were used to solve the noisy images as well as edge detection. The advantage of median filter is to be able to preserve the edges while removing noises. Especially, for salt and pepper noise, median filter is effective. The advantage of mean filter is to be useful in removing Gaussian noise and Uniform noise. But because mean filter uses the pixel far away from the center, the disadvantage is that the boundaries can be blurred.

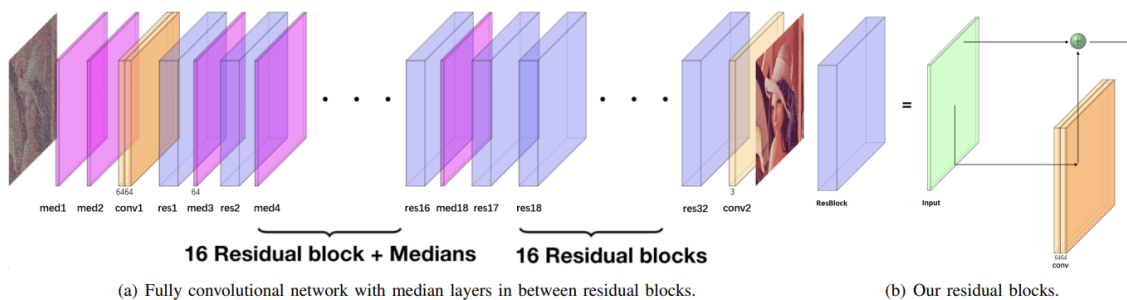


Figure 9: CNN with median filter [3]

Recently, many papers about image restoration are published, CNN with median filter [3] and image restoration PDE and so on. First, CNN with median filter uses CNN, which is mainly used in image-related work such as classification, image detection and so on. By using many residual blocks, the performance is much better than the previous filters.

Second, image restoration PDE is derived from [13] and [14]. The expression of the image restoration PDE is like this.

$$u_t = \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) \quad (24)$$

The advantage of PDE is to make the noisy image smoothly. But the disadvantage of PDE is not to be able to preserve the boundaries. Because the equation (24) is the 2nd order PDE, it does not perform well on the edge.

The comparison results are like these.(Figure 10-13, Appendix Figure 25-32)

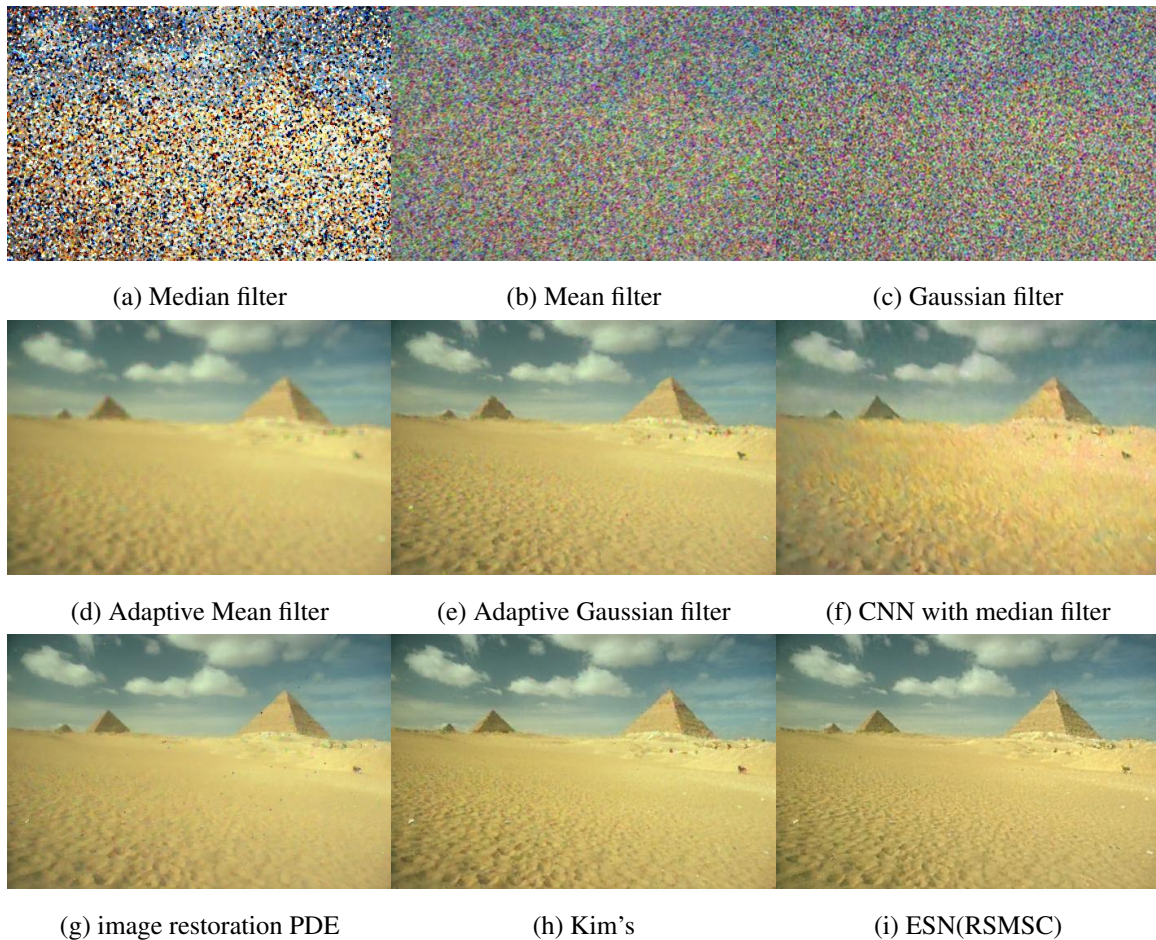


Figure 10: The result images with 80% noise.(Pyramid)

The pyramid image results tell that kim's and ESN(RSMSC) are the best among them. We can find that the (g) of Figure 10 is a little blurry. By looking at Figure 11, the sand part of the image using PDE is blurry, whereas those using kim's and ESN(RSMSC) looks clear.

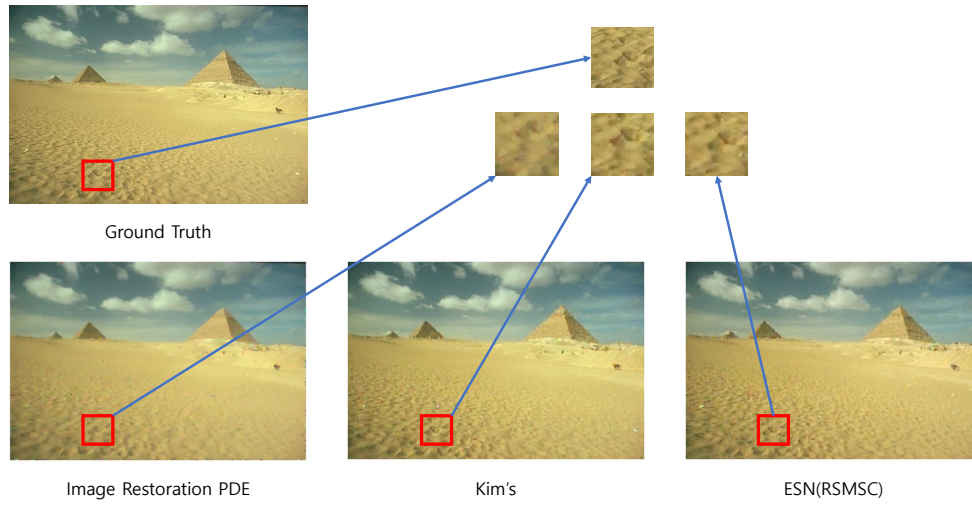


Figure 11: The result images with 80% noise.(Pyramid)

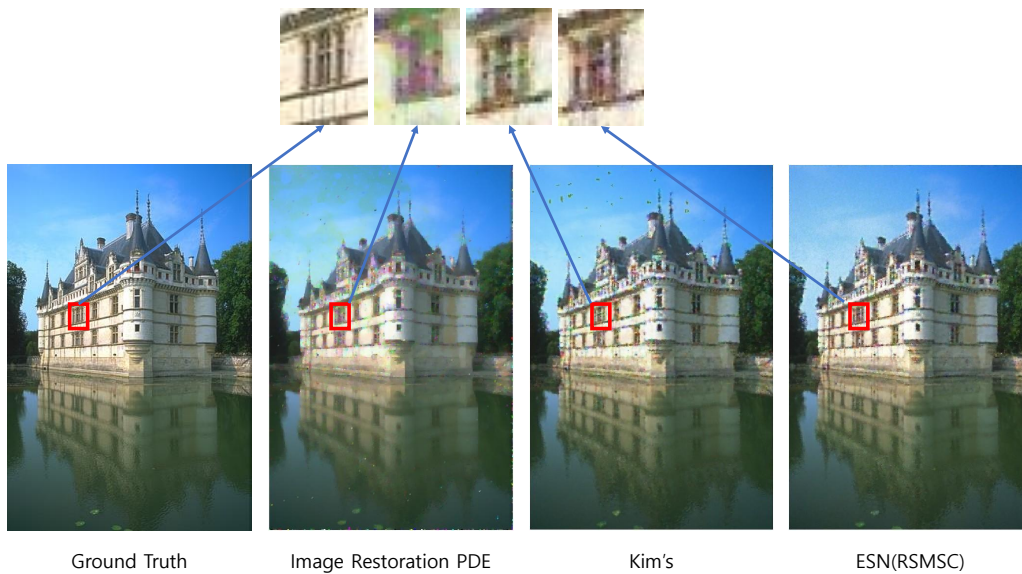
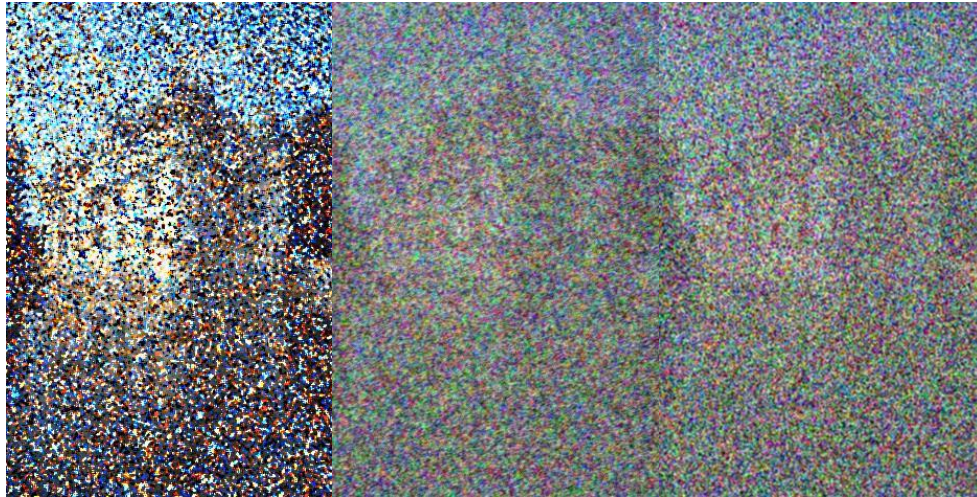


Figure 12: The result images with 80% noise.(Castle)

The castle image results tell that (i) is the best among them. the sky of (f) is blurry, those of (g) and (h) respectively still have noises. However, the sky of (i) looks clean, so does the lake. And we compare the window that the castle has. The image of PDE looks blurry and so does the bottom line. And that of ESN(RSMSC) is better than that of kim's because the bottom line looks clear and the window shape looks similar to the ground truth image.



(a) Median filter

(b) Mean filter

(c) Gaussian filter



(d) Adaptive Mean filter

(e) Adaptive Gaussian filter

(f) CNN with median filter



(g) image restoration PDE

(h) Kim's

(i) ESN(RSMSC)

Figure 13: The result images with 80% noise.(Castle)

The next tables show the error analysis. And the 'ESN(RSMSC)' adopted the reservoir whose result is the best. In addition, 'AM' means 'adaptive mean filter', 'AG' means 'adaptive gaussian filter, and 'CNN' means 'CNN with median filter [3]'.(Table 2, Appendix Table 6 - 8)

Castle						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	14.6816	13.6384	13.2709	22.3886	20.7015
	RMSE	0.1845	0.2080	0.2170	0.0760	0.0922
	SSIM	0.3168	0.2369	0.2259	0.8150	0.7918
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	27.0793	27.5422	28.6168	31.4036	
	RMSE	0.0767	0.0420	0.0371	0.0269	
	SSIM	0.9218	0.9385	0.9541	0.9761	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	8.9198	11.7790	11.4299	19.5233	17.9506
	RMSE	0.3581	0.2577	0.2682	0.1056	0.1266
	SSIM	0.0993	0.1252	0.1150	0.7317	0.7163
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	25.0120	23.9934	25.3382	26.6868	
	RMSE	0.0973	0.0631	0.0541	0.0463	
	SSIM	0.8775	0.8723	0.9066	0.9295	
Pyramid						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	15.3693	15.3897	14.7767	30.3653	23.9161
	RMSE	0.1704	0.1700	0.1825	0.0303	0.0637
	SSIM	0.3959	0.3217	0.2920	0.9715	0.8081
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	31.6906	33.9072	35.0199	34.3632	
	RMSE	0.0451	0.0202	0.0177	0.0191	
	SSIM	0.9204	0.9888	0.9904	0.9897	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	9.3725	13.6455	13.6105	26.7536	21.8543
	RMSE	0.3399	0.2078	0.2212	0.0460	0.0808
	SSIM	0.1095	0.1641	0.1491	0.9292	0.7489
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	29.9931	31.2058	32.0238	32.7827	
	RMSE	0.0548	0.0275	0.0251	0.0230	
	SSIM	0.8799	0.9781	0.9814	0.9843	

Table 2: The error results comparing other methods(Castle and Pyramid)

Although there are some cases that the image restoration PDE is the best (Table 6), the boundaries in image of PDE are mostly blurred. But, those of ESN with RSMSC are clear. That's because when we find input weights, unsharp masking filter is used. And unsharp masking filter makes boundaries clear. Therefore, the problem of image restoration PDE is that there are some noises, especially on boundaries, and the boundaries looks blurred. Also, there are some cases that kim's is the best (Table 2). But there are some noises (Figure 13,27) on the background. That's because kim's does not use many reservoirs, whereas ESN with RSMSC uses many reservoirs and can pick a reservoir whose result is the best. As we can see, the result of ESN with the reservoir state matrices sequential concatenation is the best among them. Of course, the methods of only using one filter such as Median filter, Mean filter, Gaussian filter, Adaptive Mean filter and Adaptive Gaussian filter are much worse than the others. Although CNN with median filter is not worse than PDE and ESN, the computational cost in training is much expensive than those, taking several hours. Because CNN with median filter has lots of layers and parameters, and back-propagation [6] is used many times. A few parameters and the way to find the parameters using least square method lead ESN to take lower cost.

4.3 The result with the reservoir state matrix sequential concatenation and no leaky integrator neurons

Third, we use the reservoir state matrix sequential concatenation (20), say RSMSC. We compare with 4.1, say Vanilla. Also, we compare No leaky integrator neurons (3.5), say NoLIN. Then the training result about the MSE (Mean Square Error) is like this. (Figure 14, 15)

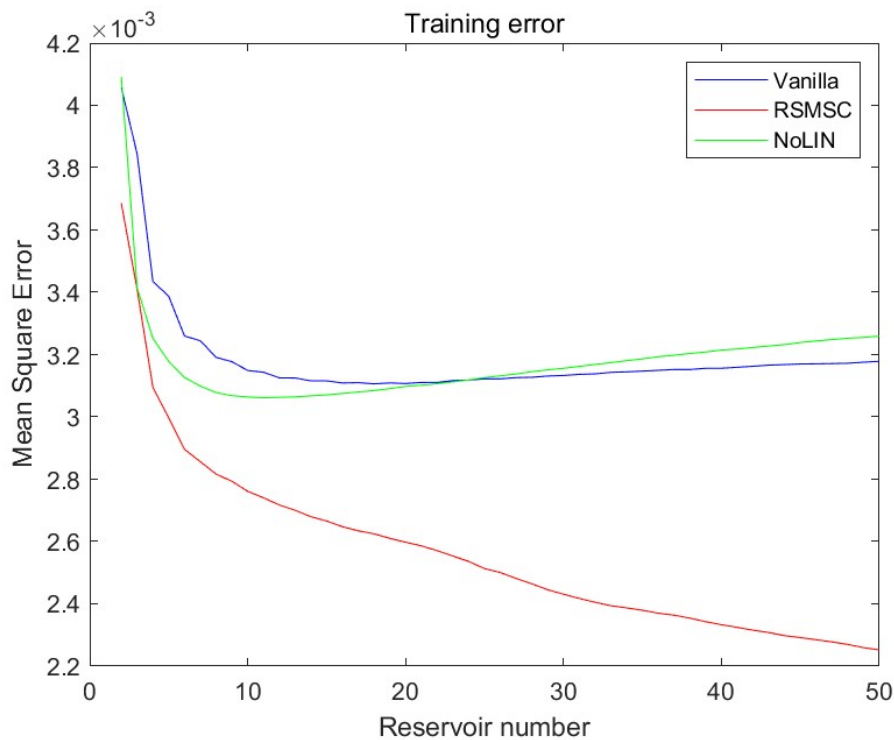


Figure 14: The training error with 60% noise.

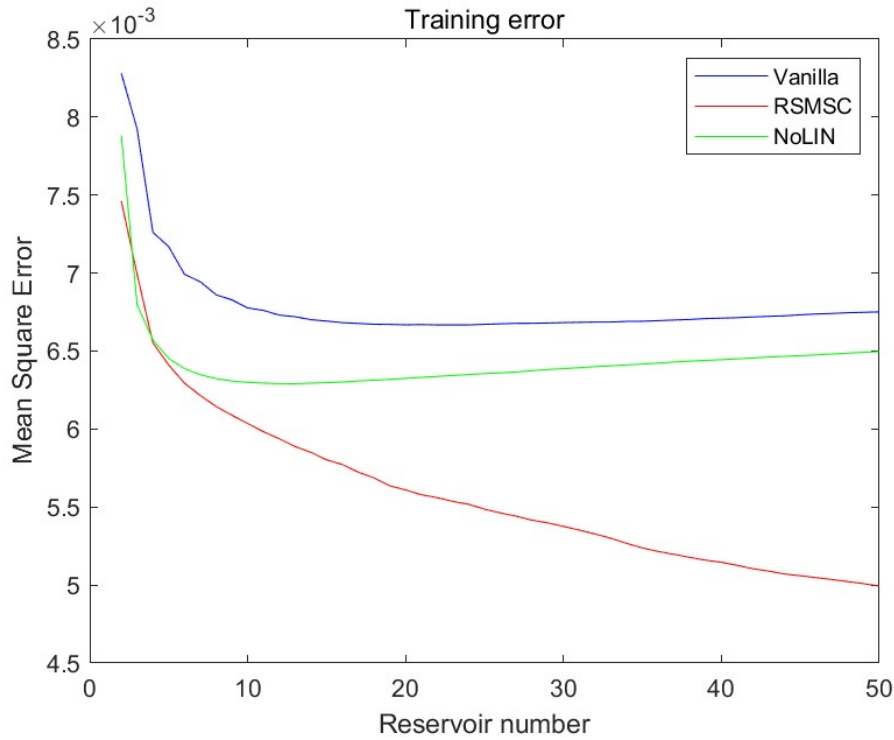


Figure 15: The training error with 80% noise.

We can find that the MSE(Mean Square Error) of RSMSC(Reservoir State Matrices Sequential Concatenation) is strictly decreasing as the number of reservoir is larger, whereas the others(Vanilla and NoLIN) are increasing slightly again. So we can expect that the test result of RSMSC will be also the best among them. The next test results show the case of 6th reservoir. (Table 3, Appendix Table 9)

		60% noise, 5×5 patch			80% noise, 7×7 patch		
		Vanilla	RSMSC	NoLIN	Vanilla	RSMSC	NoLIN
castle	PSNR	29.0934	31.4036	29.4728	23.5922	26.6868	24.5124
	RMSE	0.0351	0.0269	0.0336	0.0661	0.0463	0.0595
	SSIM	0.9380	0.9761	0.9525	0.8395	0.9295	0.8651
pyramid	PSNR	31.2479	34.3632	30.7714	26.1793	32.7827	26.5237
	RMSE	0.0274	0.0191	0.0289	0.0491	0.0230	0.0472
	SSIM	0.9743	0.9897	0.9756	0.9338	0.9843	0.9460

Table 3: The result comparing RSMSC and NoLIN

As we can see, the result of RSMSC is a little better than the others. That is, the result of the 6th reservoir of RSMSC is the best among theirs.

4.4 The result depending on the number of training datasets

We consider depending on the number of training datasets.

	The number of training datasets
Blue line	50
Red line	100
Green line	200
Sky-blue line	500
Yellow line	1000

Table 4: The meaning of the line colors

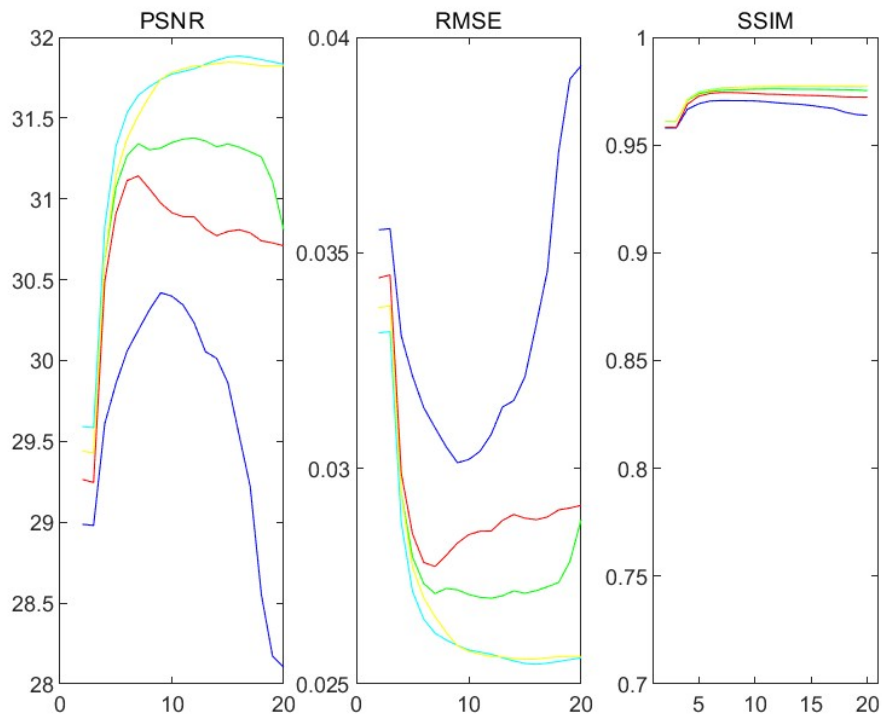


Figure 16: The result comparison depending on the number of training datasets. (Castle, 60% noise)

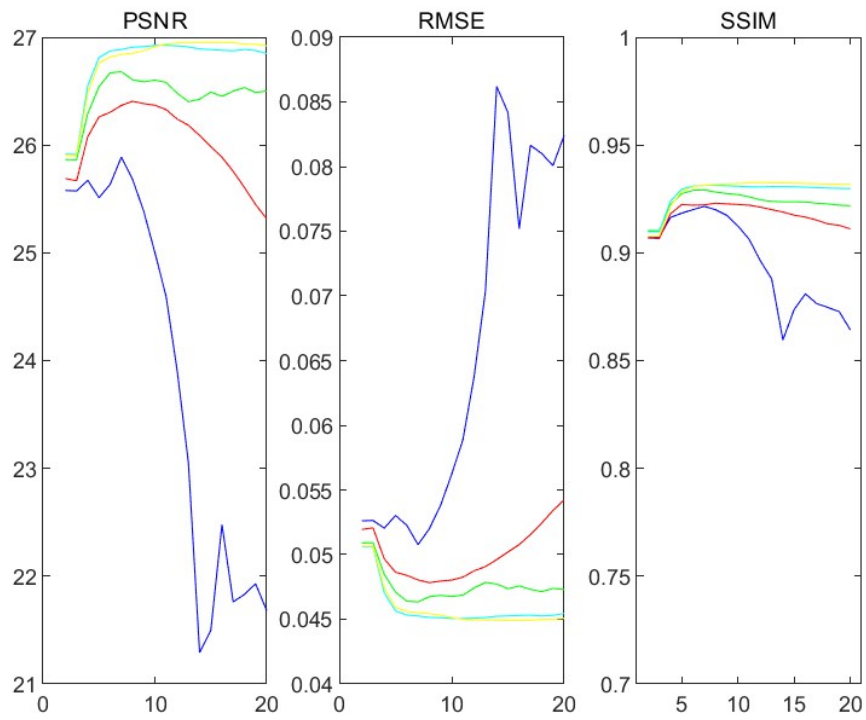


Figure 17: The result comparison depending on the number of training datasets. (Castle, 80% noise)

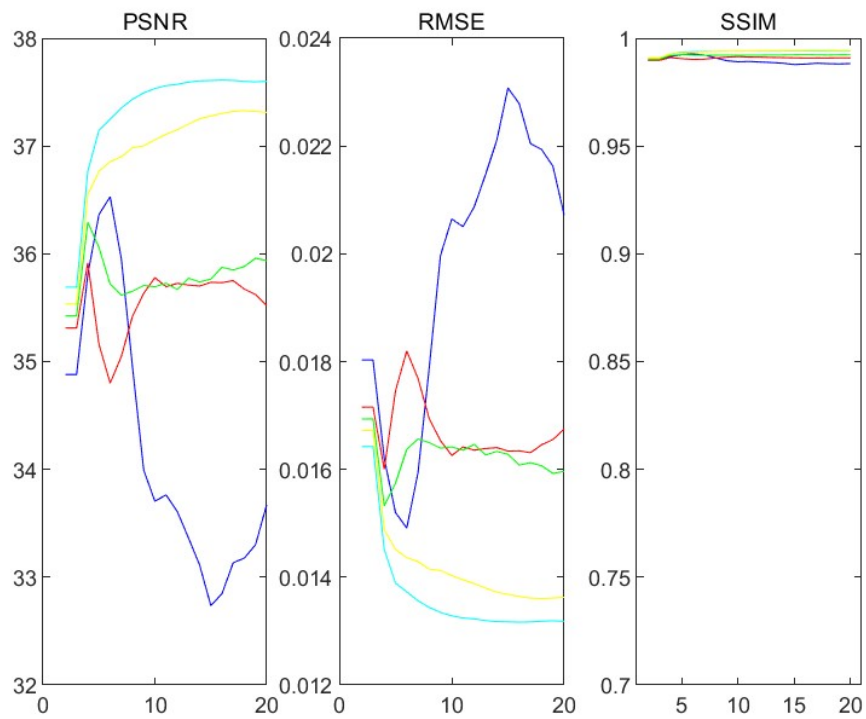


Figure 18: The result comparison depending on the number of training datasets. (Pyramid, 60% noise)

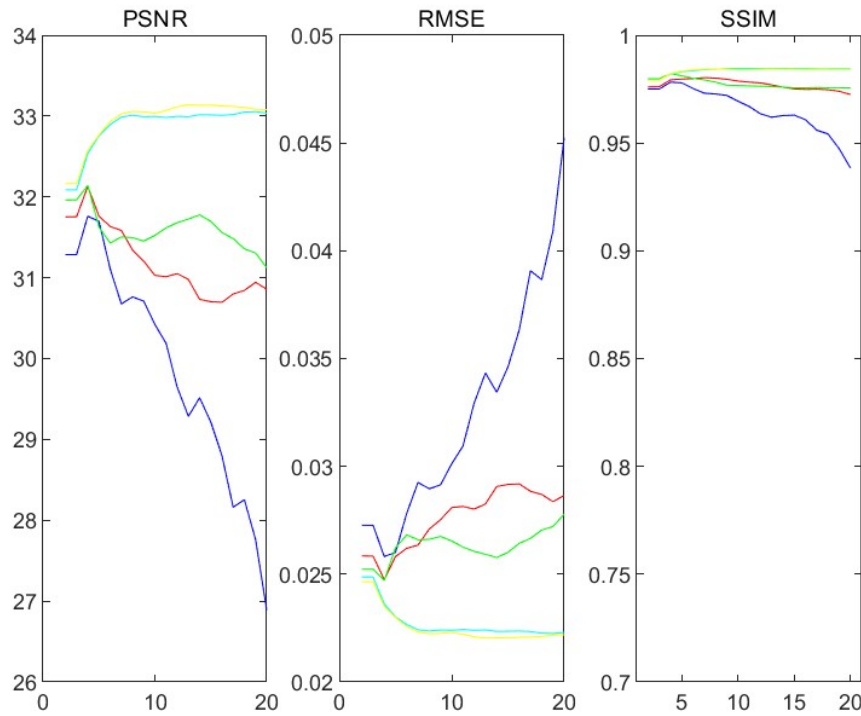


Figure 19: The result comparison depending on the number of training datasets. (Pyramid, 80% noise)

We can find that the more the number of training datasets, the better results. And the yellow lines(1000 training datasets) are the best results of most cases. (Figure 16-19, Appendix Figure 33-37)

V Conclusion

RNN is mainly used in text-related work. And CNN is used in image-related work. But RNN and ESN are also used in image-related work as well as CNN. This paper dealt with image restoration given salt-and-pepper noise. So I think ESN can also restore images given other noise modelings such as gaussian noise, uniform noise and so on. Also, I think ESN can be applied by other image-related work such as image enhancement, segmentation, line detection and so on. And ESN can solve the problem about computational cost, which is happened by other Artificial Neural Network(ANN) such as CNNs and RNNs. But the limitation is that if the number of the reservoir is larger, then the errors such as PSNR, RMSE and SSIM are getting worse than the previous reservoir. And if we use the reservoir state matrices sequential concatenation, then the errors are getting worse faster, causing the Overfitting. I think the theories about ESN are deficient, so we have to find some properties about ESN to solve the error problem in the future.

Appendix

The result depending on patch sizes(4.1)

		60% noise			80% noise		
		5×5	7×7	9×9	5×5	7×7	9×9
airplane	PSNR	41.6450	41.1023	40.5721	34.9082	36.7414	35.8637
	RMSE	0.0083	0.0088	0.0094	0.0180	0.0146	0.0161
	SSIM	0.9909	0.9905	0.9882	0.9464	0.9779	0.9609
cheetah	PSNR	32.8087	32.6997	32.2161	28.4740	28.1163	27.6197
	RMSE	0.0229	0.0232	0.0245	0.0377	0.0393	0.0416
	SSIM	0.9897	0.9894	0.9884	0.9707	0.9704	0.9602
butterfly	PSNR	34.0846	33.3768	33.0972	29.2545	29.8326	28.5153
	RMSE	0.0198	0.0214	0.0221	0.0345	0.0322	0.0375
	SSIM	0.9920	0.9903	0.9901	0.9765	0.9807	0.9734
statue	PSNR	29.1049	28.7655	28.6648	24.6654	24.5742	24.4834
	RMSE	0.0351	0.0365	0.0369	0.0584	0.0591	0.0597
	SSIM	0.9370	0.9322	0.9317	0.8184	0.8212	0.8149
camel	PSNR	33.0169	32.6349	32.3923	28.3608	28.3022	28.0573
	RMSE	0.0223	0.0233	0.0240	0.0382	0.0384	0.0395
	SSIM	0.9742	0.9704	0.9694	0.9182	0.9212	0.9124

Table 5: The result depending on patch sizes

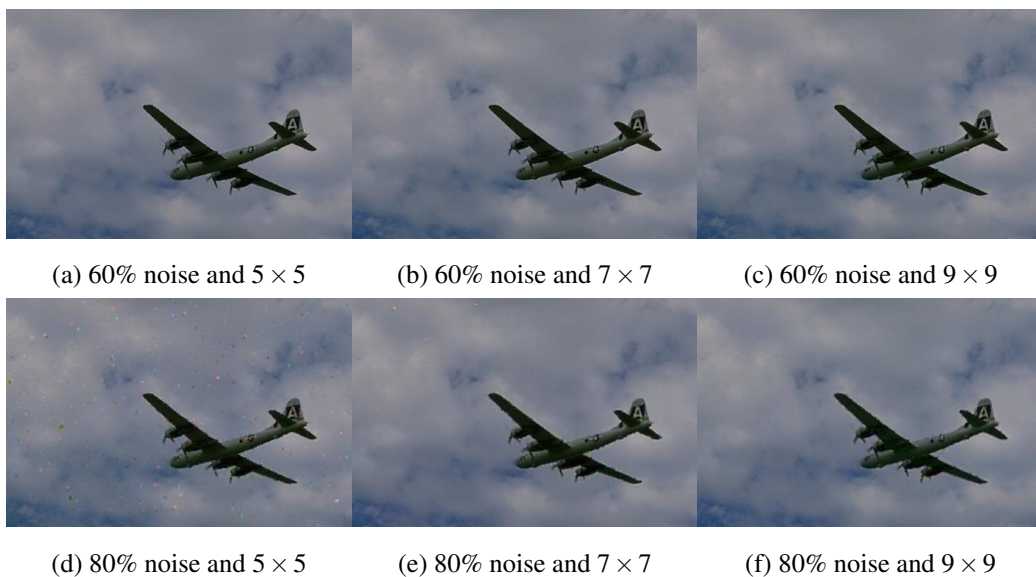


Figure 20: The result images depending on the pixel sizes.(Airplane)

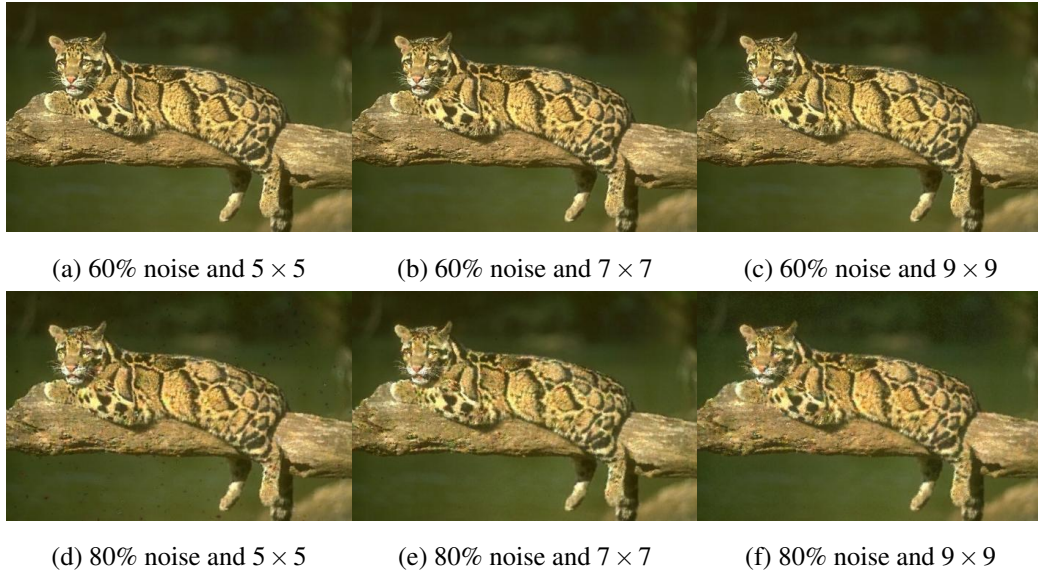


Figure 21: The result images depending on the pixel sizes.(Cheetah)

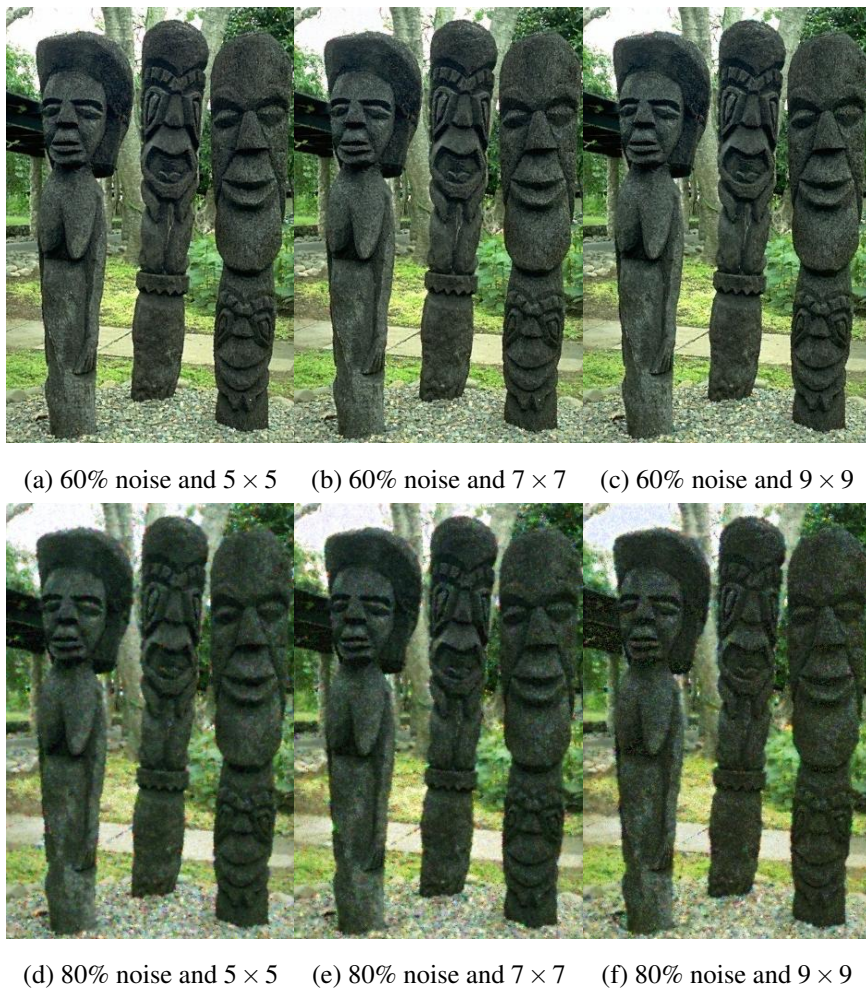


Figure 22: The result images depending on the pixel sizes.(Status)

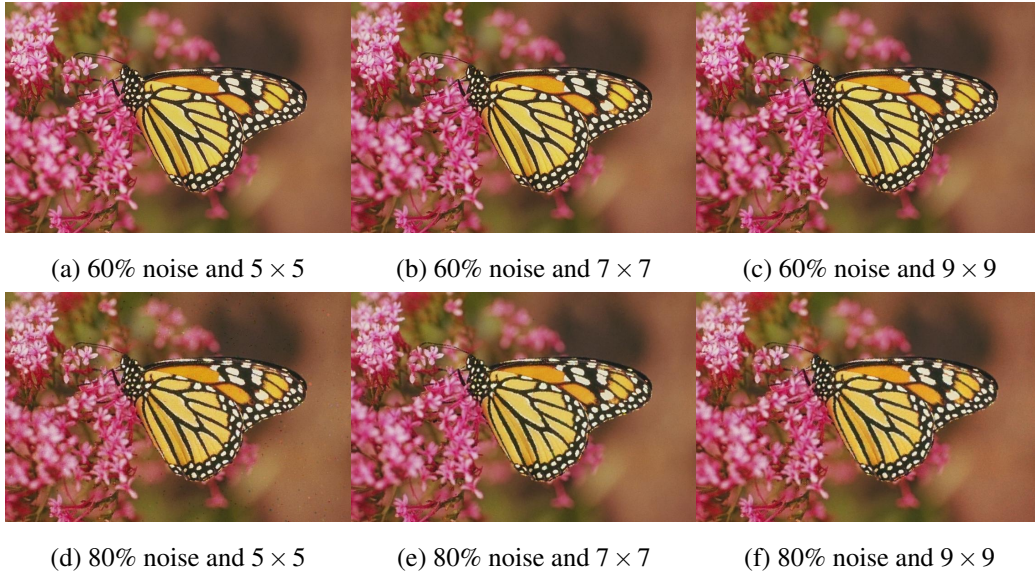


Figure 23: The result images depending on the pixel sizes.(Butterfly)

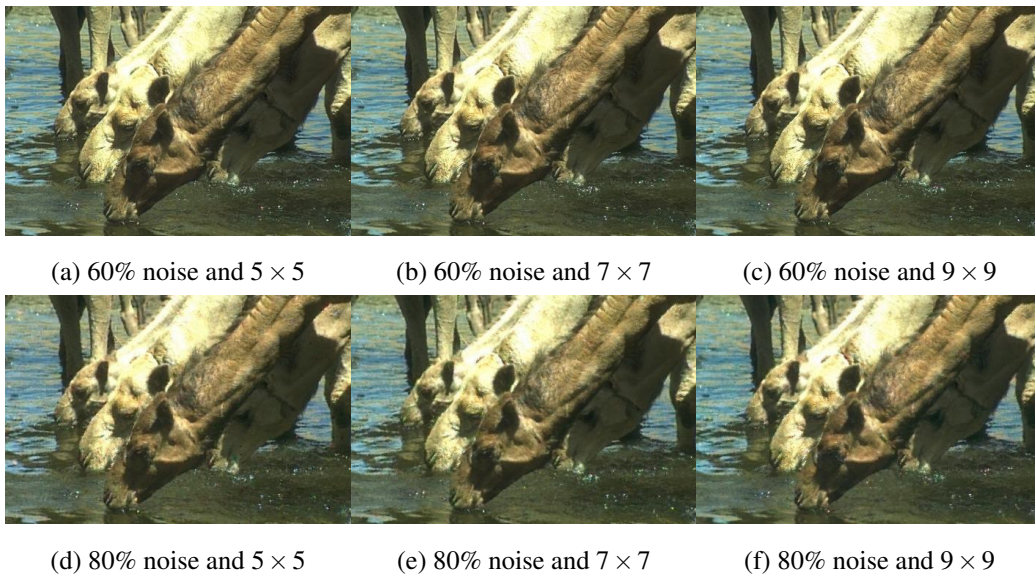


Figure 24: The result images depending on the pixel sizes.(Camel)

Comparing the existed methods(4.2)

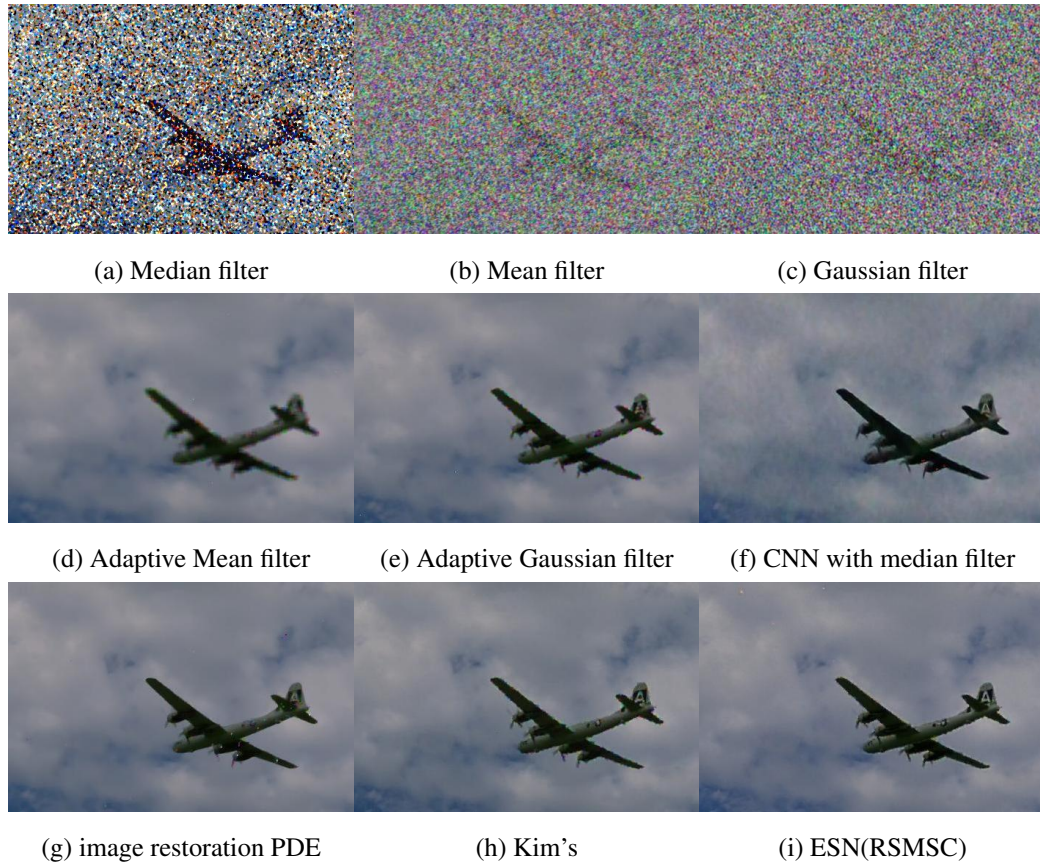


Figure 25: The result images with 80% noise.(Airplane)

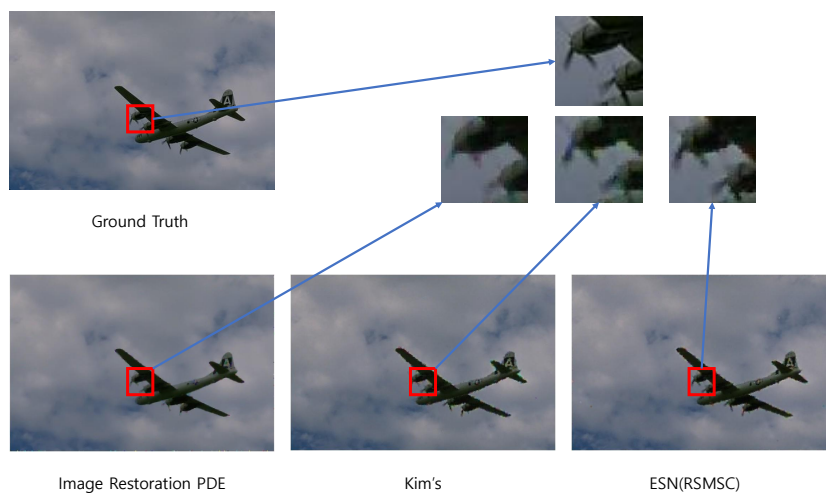


Figure 26: The result images with 80% noise.(Airplane)

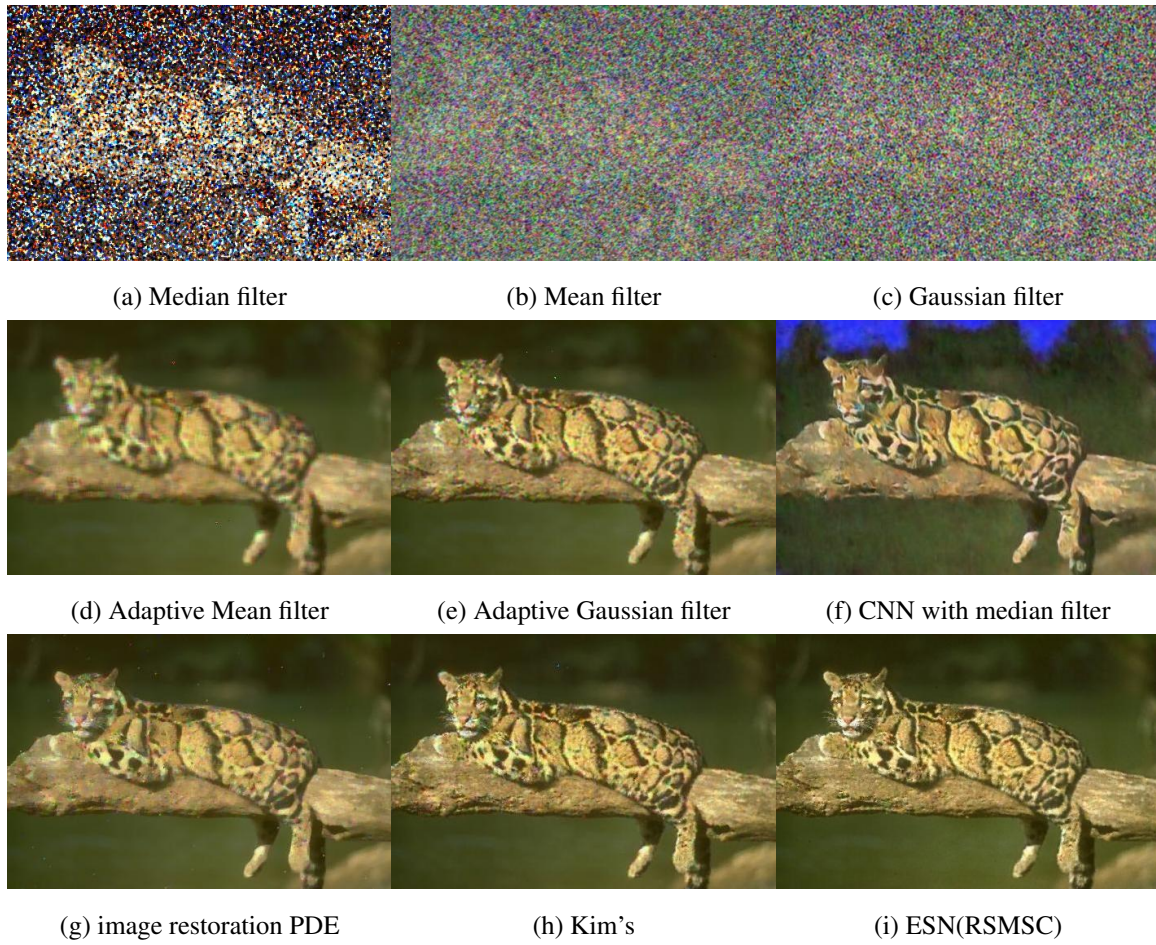


Figure 27: The result images with 80% noise.(Cheetah)

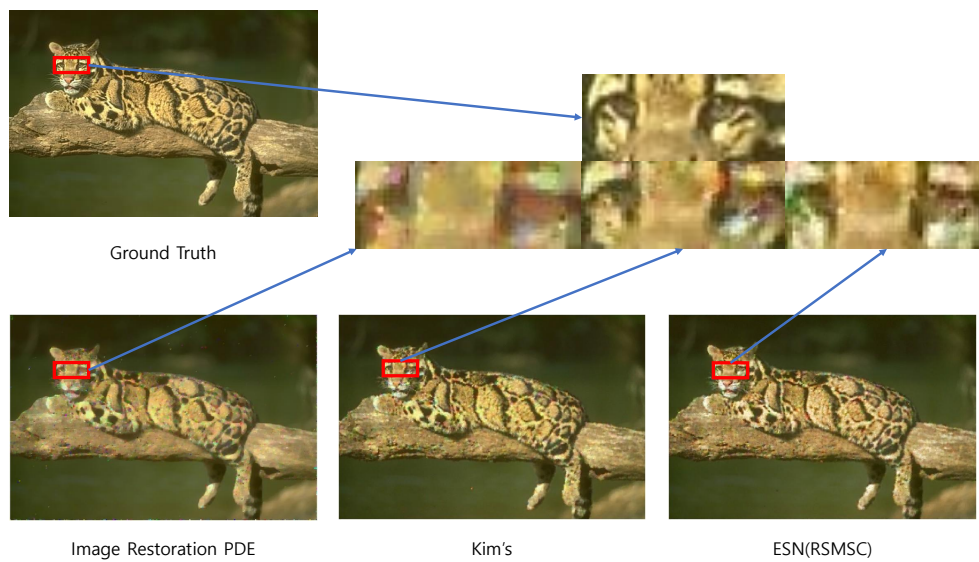


Figure 28: The result images with 80% noise.(Cheetah)

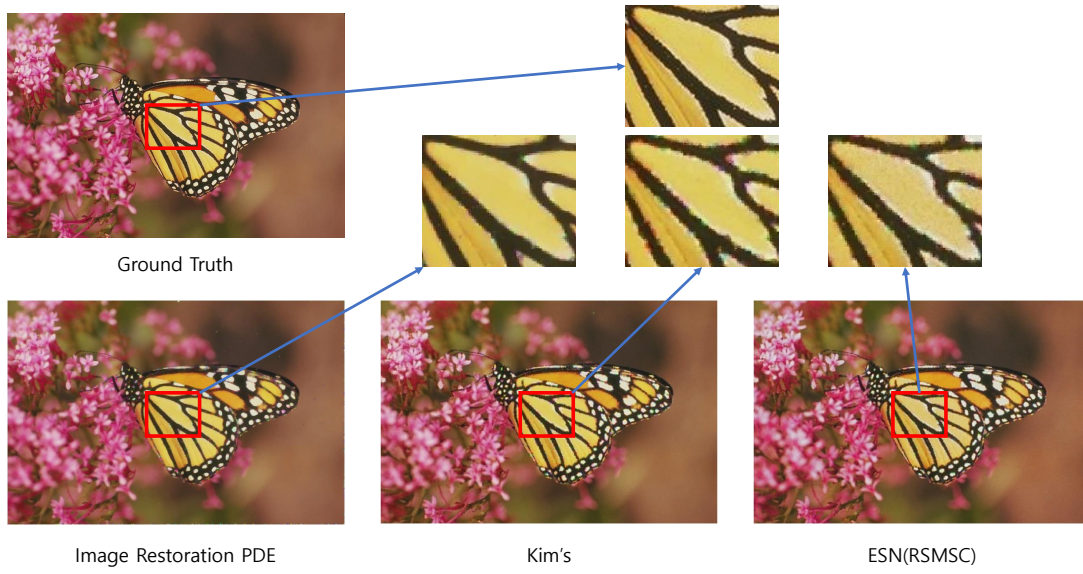


Figure 29: The result images with 80% noise.(Butterfly)

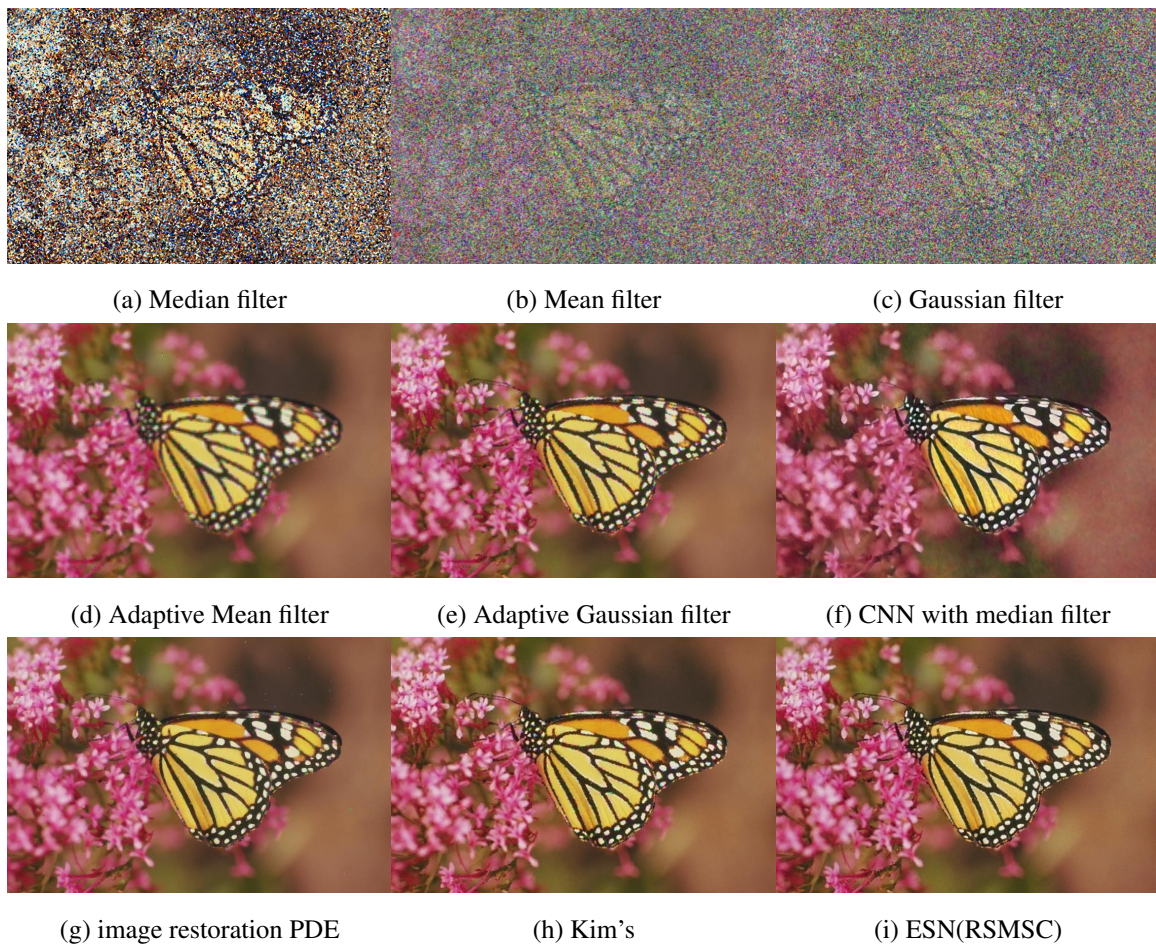
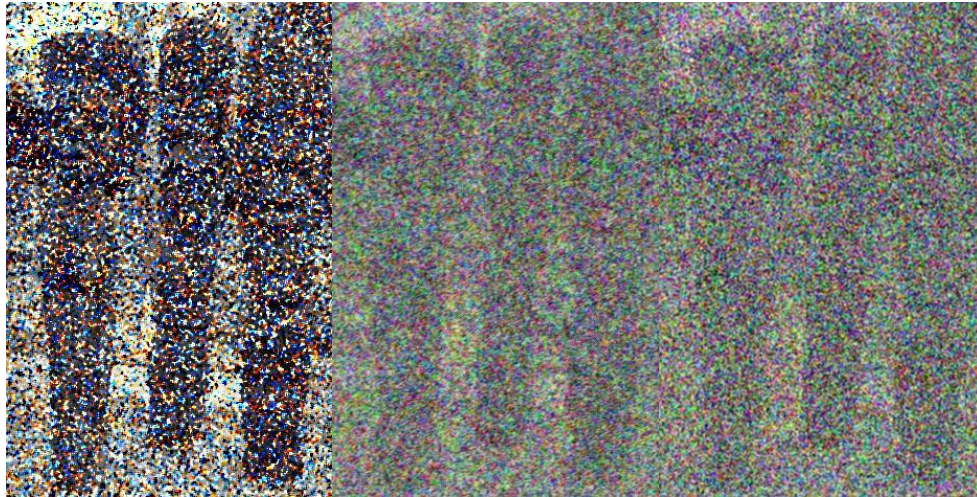


Figure 30: The result images with 80% noise.(Butterfly)



(a) Median filter

(b) Mean filter

(c) Gaussian filter



(d) Adaptive Mean filter

(e) Adaptive Gaussian filter

(f) CNN with median filter



(g) image restoration PDE

(h) Kim's

(i) ESN(RSMSC)

Figure 31: The result images with 80% noise.(Status)

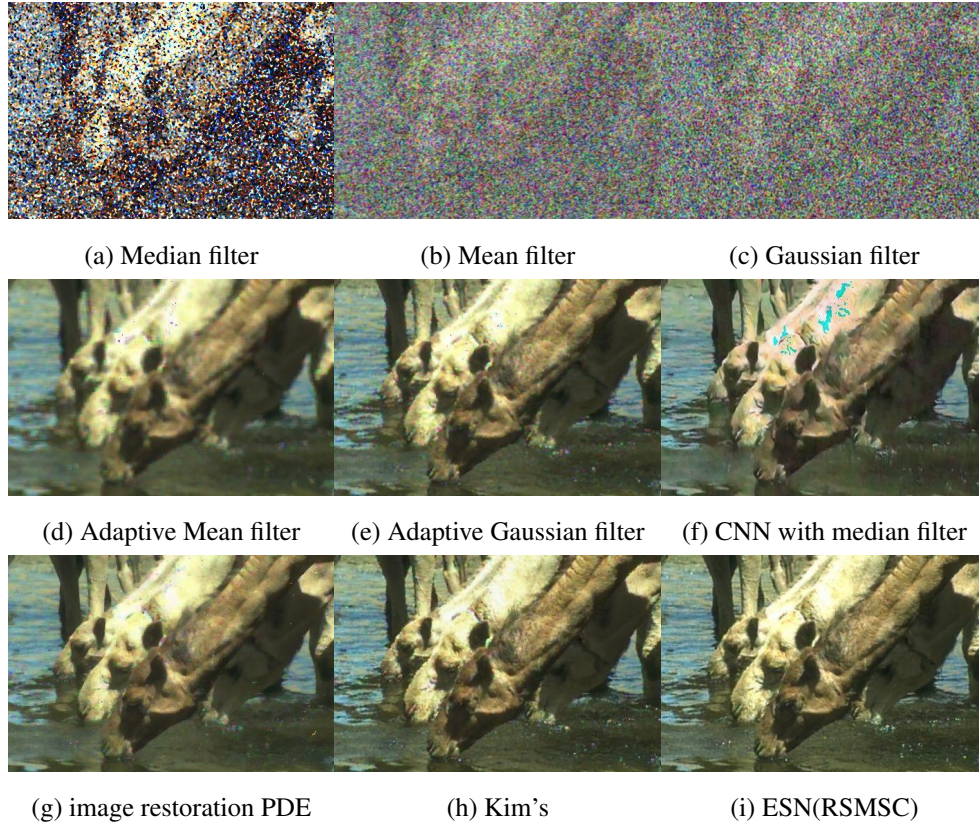


Figure 32: The result images with 80% noise.(Camel)

		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	17.1949	17.0207	13.2709	33.4803	24.6801
	RMSE	0.1381	0.1409	0.2170	0.0212	0.0583
	SSIM	0.1846	0.1099	0.2259	0.9694	0.9383
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	33.7118	40.6265	39.1251	41.6450	
	RMSE	0.0357	0.0093	0.0111	0.0083	
	SSIM	0.9802	0.9922	0.9891	0.9909	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	10.0962	15.5592	11.4299	28.3966	22.4976
	RMSE	0.3127	0.1667	0.2682	0.0380	0.0750
	SSIM	0.0258	0.0681	0.1150	0.8672	0.9049
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	33.1430	36.3365	35.4601	36.7414	
	RMSE	0.0381	0.0152	0.0169	0.0146	
	SSIM	0.9729	0.9822	0.9763	0.9779	

Table 6: The error results comparing other methods(Airplane)

Cheetah						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	14.9531	13.4197	13.0186	24.3609	21.8690
	RMSE	0.1788	0.2133	0.2234	0.0605	0.0806
	SSIM	0.3665	0.2501	0.2258	0.9268	0.8251
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	26.6908	28.5834	30.3032	32.8087	
	RMSE	0.0802	0.0372	0.0305	0.0229	
	SSIM	0.9103	0.9742	0.9816	0.9897	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	8.9269	11.5213	11.1766	22.4293	19.9140
	RMSE	0.3578	0.2654	0.2762	0.0756	0.1010
	SSIM	0.0912	0.1239	0.1122	0.8733	0.7669
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	17.9008	25.1183	26.7132	28.1163	
	RMSE	0.2206	0.0555	0.0462	0.0393	
	SSIM	0.8278	0.9426	0.9605	0.9704	
Butterfly						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	14.6588	14.4087	13.9653	26.1719	24.4800
	RMSE	0.1850	0.1904	0.2003	0.0491	0.0597
	SSIM	0.4530	0.3582	0.3283	0.9579	0.8987
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	28.3151	32.4354	32.1753	34.0846	
	RMSE	0.0665	0.0239	0.0246	0.0198	
	SSIM	0.9452	0.9898	0.9884	0.9920	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	9.0440	12.6151	12.1954	23.7028	21.8439
	RMSE	0.3530	0.2340	0.2456	0.0653	0.0809
	SSIM	0.1287	0.1765	0.1606	0.9199	0.8439
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	26.2850	27.6185	28.3437	29.8326	
	RMSE	0.0840	0.0416	0.0383	0.0322	
	SSIM	0.9148	0.9721	0.9738	0.9807	

Table 7: The error results comparing other methods(Cheetah and Butterfly)

Status						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	14.7915	13.2267	12.8724	20.5678	19.7567
	RMSE	0.1821	0.2181	0.2272	0.0937	0.1028
	SSIM	0.2419	0.1606	0.1530	0.5626	0.5795
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	24.5522	25.1667	26.9897	29.1049	
	RMSE	0.1026	0.0552	0.0447	0.0351	
	SSIM	0.8272	0.8277	0.8919	0.9370	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	8.8632	11.3696	11.0474	18.9474	17.7616
	RMSE	0.3604	0.2701	0.2803	0.1129	0.1294
	SSIM	0.0555	0.0857	0.0785	0.4972	0.4880
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	22.3835	22.4563	23.3142	24.5742	
	RMSE	0.1316	0.0754	0.0683	0.0591	
	SSIM	0.7192	0.6822	0.7641	0.8212	
Camel						
		median filter	mean filter	gaussian filter	AM	AG
60%	PSNR	15.4575	13.8839	13.4781	24.8134	22.3765
	RMSE	0.1687	0.2022	0.2119	0.0575	0.0761
	SSIM	0.3097	0.2049	0.1879	0.8184	0.7496
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
60%	PSNR	27.7171	29.2633	30.4913	33.0169	
	RMSE	0.0712	0.0344	0.0299	0.0223	
	SSIM	0.8935	0.9365	0.9552	0.9742	
		median filter	mean filter	gaussian filter	AM	AG
80%	PSNR	9.1941	12.0469	11.7036	21.9878	19.9130
	RMSE	0.3470	0.2498	0.2599	0.0795	0.1010
	SSIM	0.0715	0.1042	0.0947	0.7323	0.6653
		CNN	PDE	Kim's [12]	ESN(RSMSC)	
80%	PSNR	25.4638	25.7634	26.9098	28.3022	
	RMSE	0.0923	0.0515	0.0451	0.0384	
	SSIM	0.8271	0.8645	0.9002	0.9212	

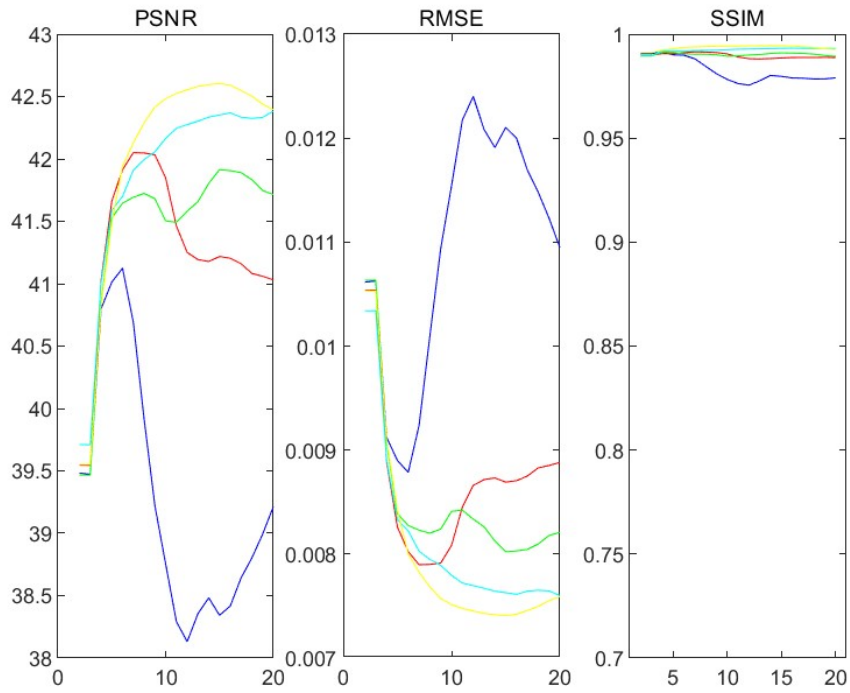
Table 8: The error results comparing other methods(Status and Camel)

The result with the reservoir state matrix sequential concatenation and no leaky integrator neurons(4.3)

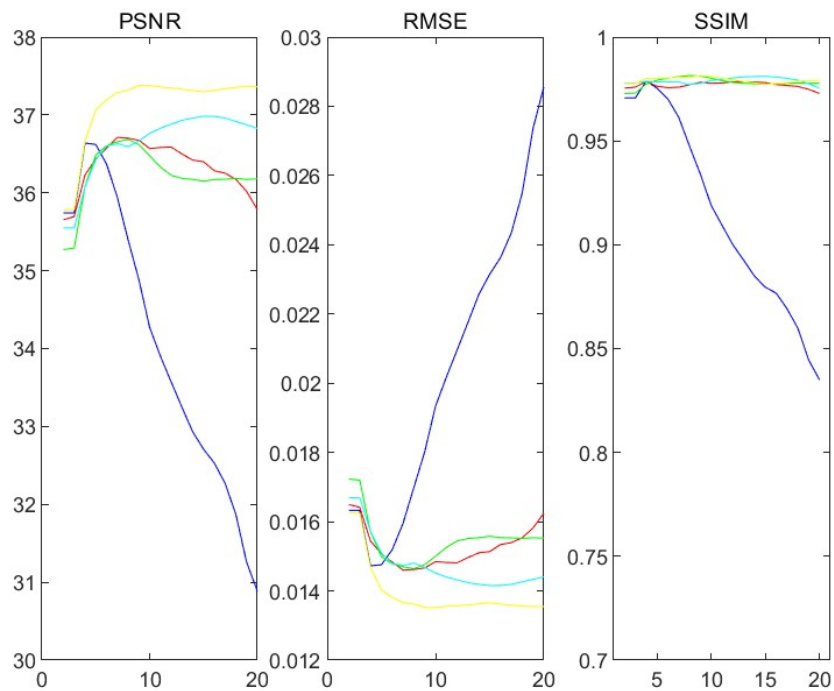
		60% noise, 5 × 5 patch			80% noise, 7 × 7 patch		
		Vanilla	RSMSC	NoLIN	Vanilla	RSMSC	NoLIN
airplane	PSNR	37.8147	41.6450	39.0332	31.8228	36.7414	33.3823
	RMSE	0.0129	0.0083	0.0112	0.0256	0.0146	0.0214
	SSIM	0.9705	0.9909	0.9764	0.9135	0.9779	0.9481
cheetah	PSNR	30.2185	32.8087	30.4329	25.1251	28.1163	25.4582
	RMSE	0.0308	0.0229	0.0301	0.0554	0.0393	0.0533
	SSIM	0.9695	0.9897	0.9722	0.9199	0.9704	0.9350
butterfly	PSNR	30.4722	34.0846	30.9037	24.8842	29.8326	25.4788
	RMSE	0.0299	0.0198	0.0285	0.0570	0.0322	0.0532
	SSIM	0.9763	0.9920	0.9813	0.9333	0.9807	0.9427
status	PSNR	27.1667	29.1049	27.5919	22.6137	24.5742	22.8554
	RMSE	0.0438	0.0351	0.0417	0.0740	0.0591	0.0720
	SSIM	0.8897	0.9370	0.9037	0.7294	0.8212	0.7379
camel	PSNR	30.0197	33.0169	30.6378	24.9127	28.3022	25.2870
	RMSE	0.0316	0.0223	0.0294	0.0568	0.0384	0.0544
	SSIM	0.9327	0.9742	0.9434	0.8360	0.9212	0.8480

Table 9: The result comparing RSMSC and NoLIN

The result depending on the number of training datasets(4.4)

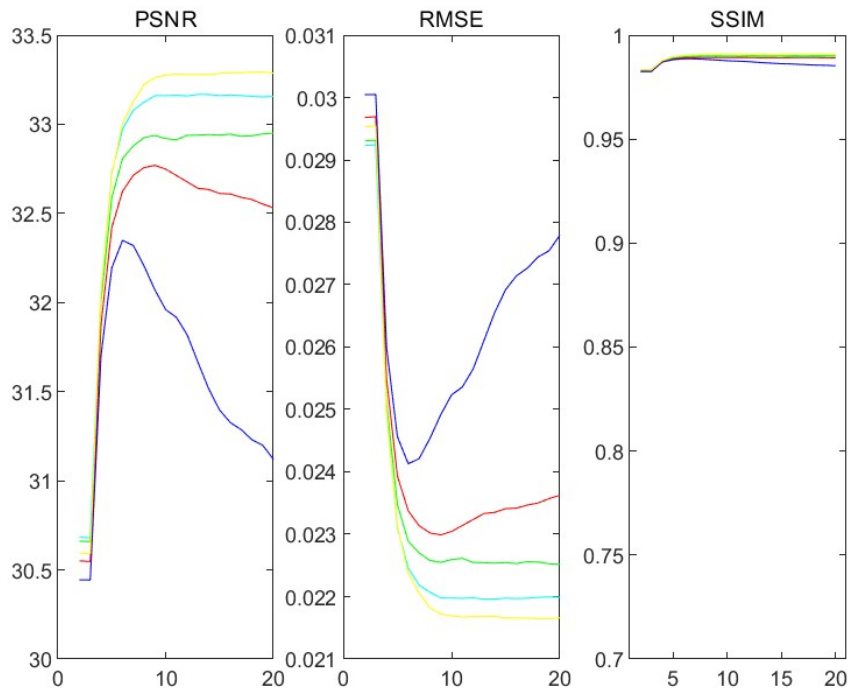


(a) 60% noise

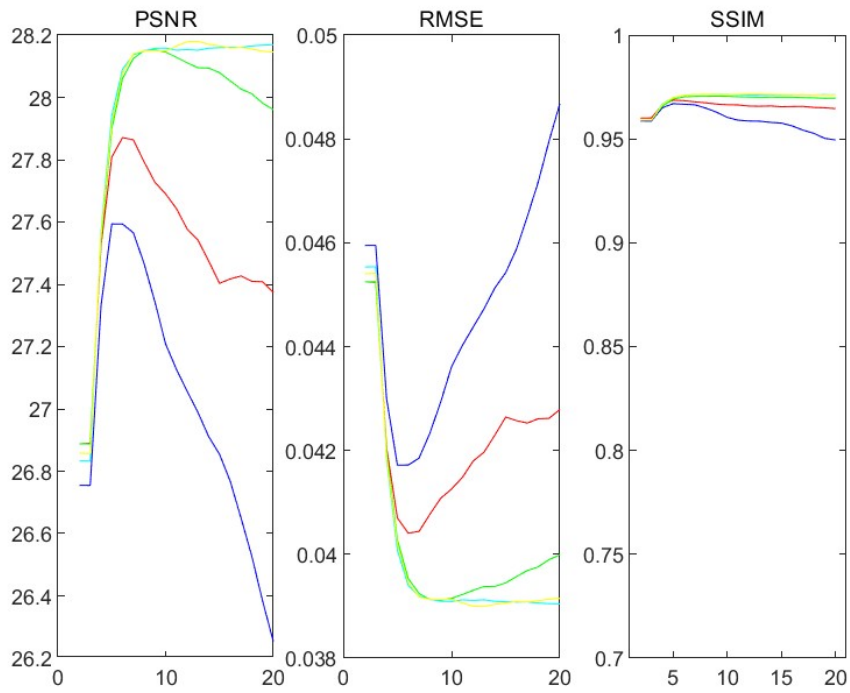


(b) 80% noise

Figure 33: The result comparison depending on the number of training datasets. (Airplane)

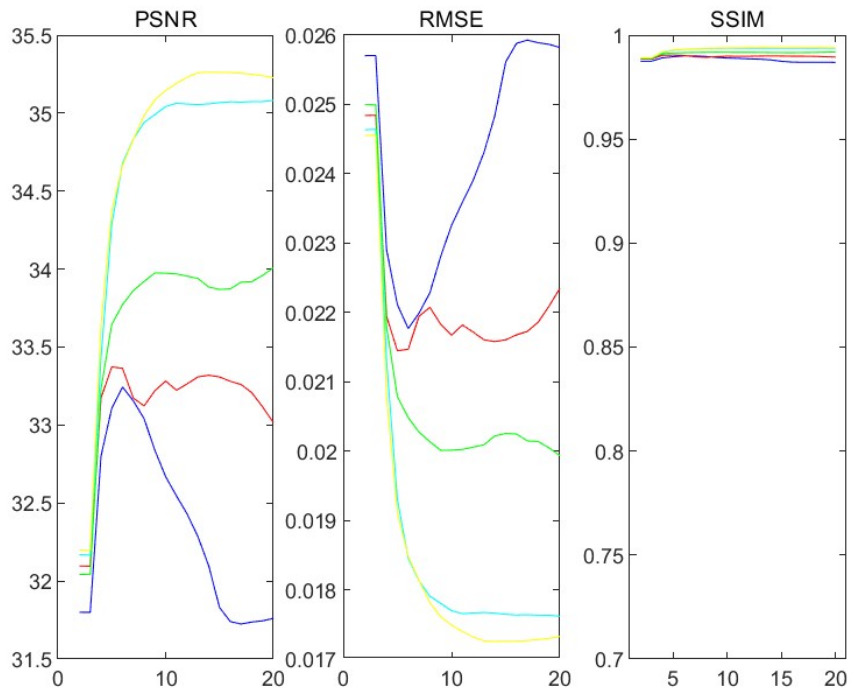


(a) 60% noise

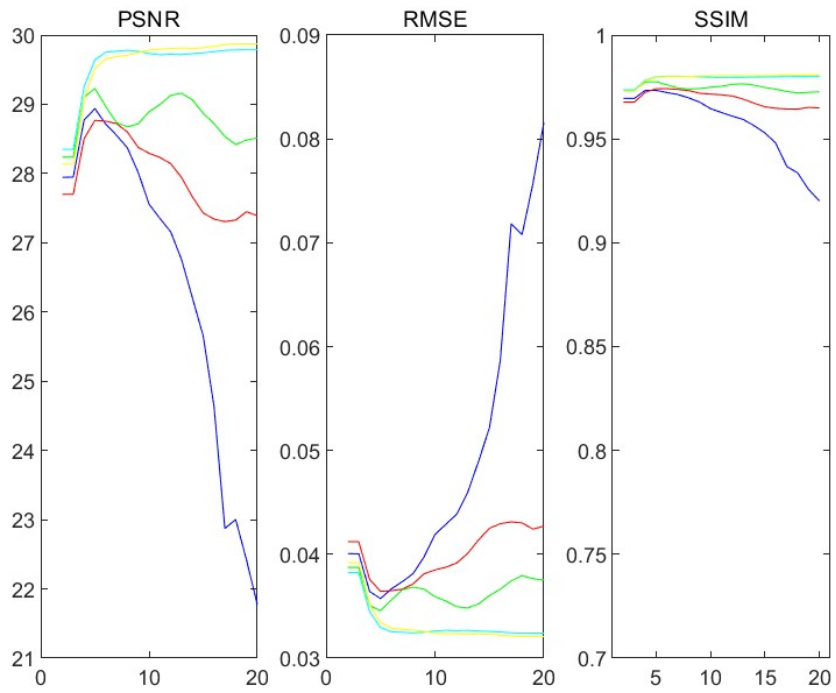


(b) 80% noise

Figure 34: The result comparison depending on the number of training datasets. (Cheetah)

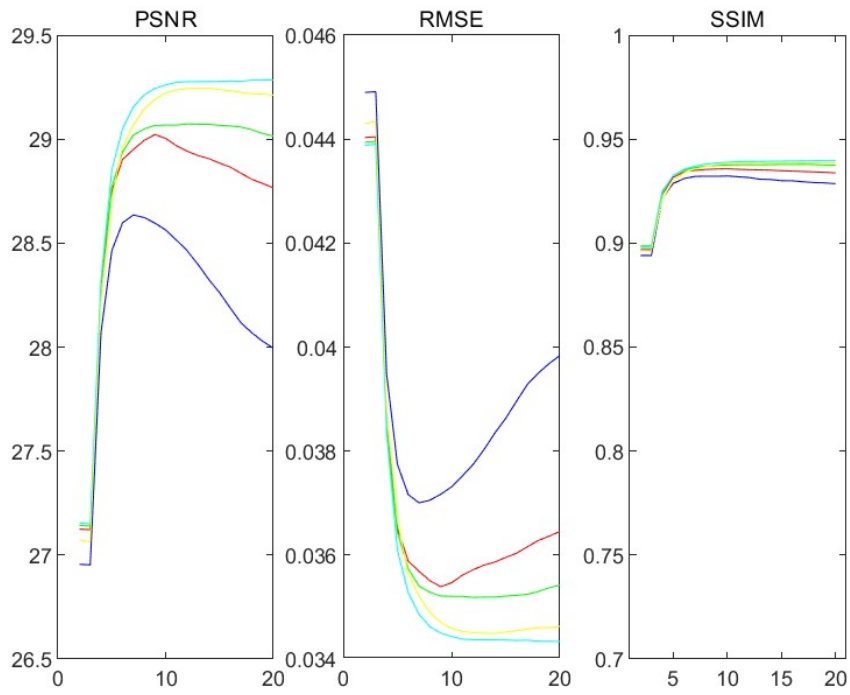


(a) 60% noise

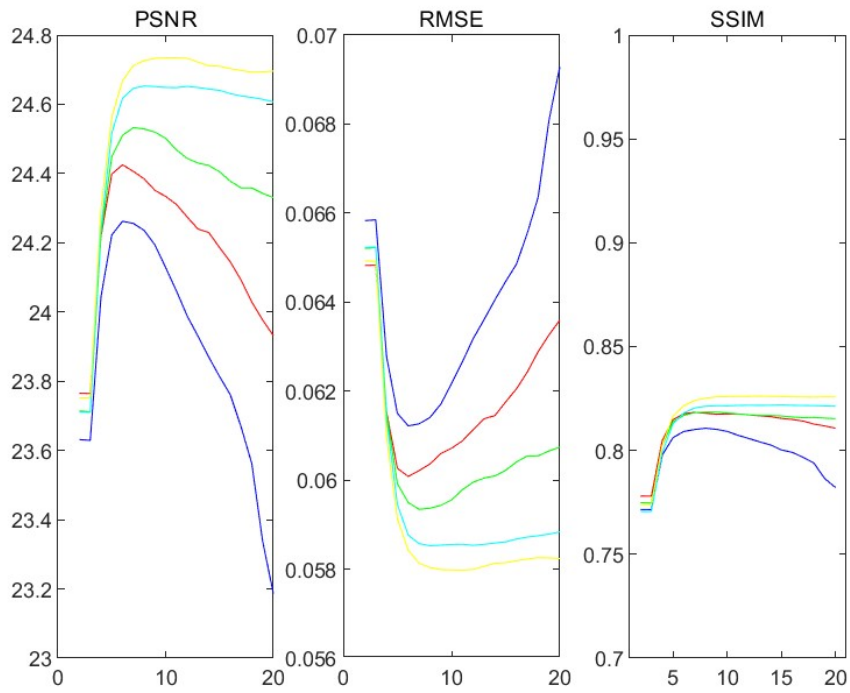


(b) 80% noise

Figure 35: The result comparison depending on the number of training datasets. (Butterfly)

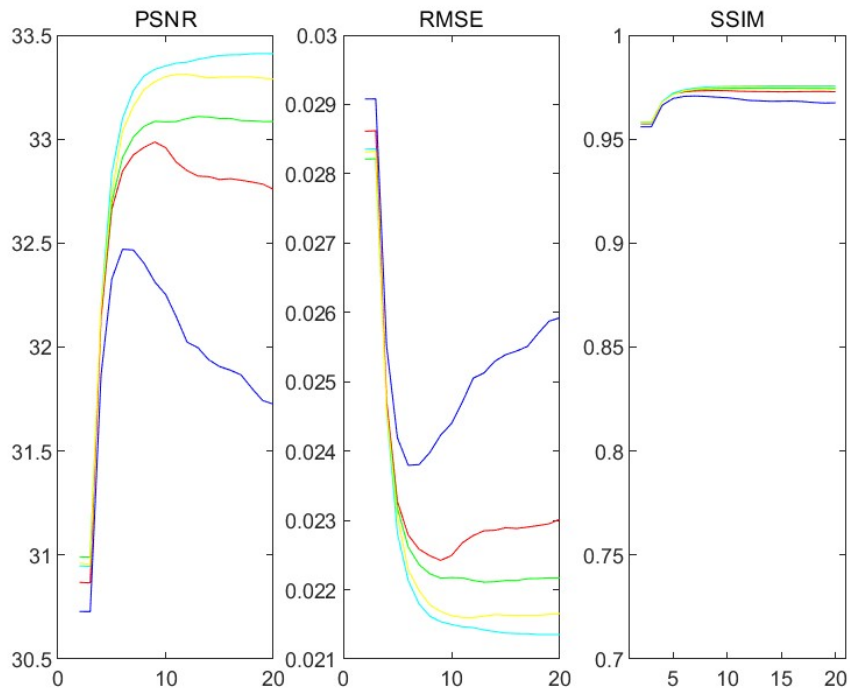


(a) 60% noise

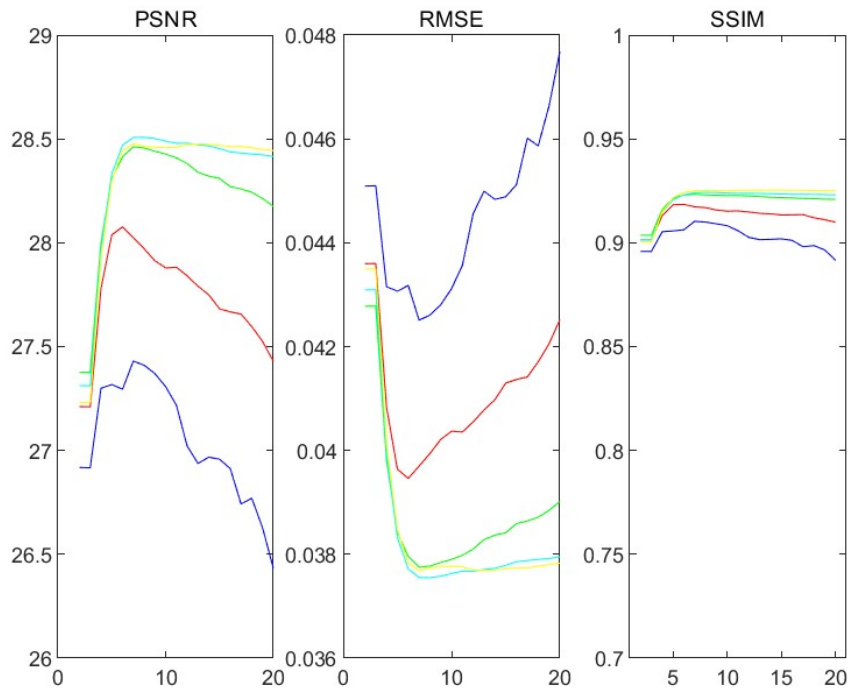


(b) 80% noise

Figure 36: The result comparison depending on the number of training datasets. (Statue)



(a) 60% noise



(b) 80% noise

Figure 37: The result comparison depending on the number of training datasets. (Camel)

References

- [1] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, “An overview of reservoir computing: theory, applications and implementations,” in *Proceedings of the 15th european symposium on artificial neural networks*. p. 471-482 2007, 2007, pp. 471–482.
- [2] H. Jaeger, “The “echo state” approach to analysing and training recurrent neural networks-with an erratum note,” *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.
- [3] L. Liang, S. Deng, L. Gueguen, M. Wei, X. Wu, and J. Qin, “Convolutional neural network with median layers for denoising salt-and-pepper contaminations,” *Neurocomputing*, vol. 442, pp. 26–35, 2021.
- [4] B. R. Hunt, “The application of constrained least squares estimation to image restoration by digital computer,” *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 805–812, 1973.
- [5] W. H. Richardson, “Bayesian-based iterative method of image restoration,” *JoSA*, vol. 62, no. 1, pp. 55–59, 1972.
- [6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.
- [7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
- [12] Y. Kim, “Effectiveness of task-specific nonlinear modeling in reservoir computing for impulse noise removal,” 2022.

- [13] T. F. Chan and J. Shen, “Variational image inpainting,” *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 58, no. 5, pp. 579–619, 2005.
- [14] P. Perona and J. Malik, “Scale-space and edge detection using anisotropic diffusion,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

Acknowledgements

I want to thank Yoojeong kim, who is my superior and wrote [12]. And my supervisor Yunho Kim helps write this thesis.

