Comparison of CNN based and self-similarity based denoising methods

Minchuan Zhou
Stanford University
mczhou@stanford.edu

Abstract

Convolutional networks and traditional self-similarity based methods are both state-of-the-art denoising methods. We implement the BM3D model and DnCNN network and show their favorable performance for denoising a Gaussian noise at a given noise level, compared to NLM. We also make use of dilated convolution in DnCNN and show that in this way we can achieve comparable denoising performance while greatly reducing the computational cost. We have also trained a DnCNN-b network for blind denoising for all noise levels between 0 and 55, which shows similar denoising results without knowledge of what the actual noise level is in the noisy image.

1. Introduction

Denoising images has been an important research topic in computational imaging, especially in low level vision as denoising is an important step before such images are processed for further application. Many new deep learning based methods are proposed as the rise of new deep learning techniques, for example, convolutional neural network (CNN). In the meanwhile, traditional methods are still playing important roles in many aspects.

The general problem of image restoration can be written in the form of $y = Ax + \eta$, where A is any linear operator, x is the original image, η is the noise term, and y is the resulting image. In the case of denoising, A is an identity operator, and y is the noisy image. In addition, we can assume that the noise is Gaussian noise with standard deviation σ. The purpose is to recover the original clean image x from the noisy image y. In class, we have learnt denoising techniques such as Gaussian filter, bilateral filter and non-local means (NLM) [1], among which NLM gives the best denoising result. NLM averages pixels with a similar neighborhood. Like other traditional methods, NLM can produce a high denoising quality but has the drawback of low computational efficiency and requires manual choice of some parameters, such as the patch size, search window size, and cut-off distance. More advanced traditional methods are also being developed, such as block-matching and 3D filtering (BM3D) [2].

On the other hand, CNN is widely used in many computational imaging aspects, such as denoising, demosaicing, superresolution, etc. As a discriminative learning method, CNN models do not require choosing the parameters manually, also require no iterative optimization in the test phase therefore is highly computationally efficient. In particular, the denoising CNN (DnCNN) [3] separates the noise from a noisy image by feed-forward CNN using residual learning and batch normalization. The advantage of CNN is two-fold: first, the deep structure of CNN is suitable for exploiting image characteristics; second, CNN can be trained on GPUs, which improves the run time performance significantly.

In this report, we implement the state-of-the-art denoising techniques, both the traditional method (BM3D) and deep learning method (DnCNN), and compare the denoising performance with the NLM method we learnt in class. In particular, we compare the quantitatively the metrics of peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) and also qualitatively the visual quality.

2. Related work

2.1. Traditional denoising method

We have learnt a variety of traditional denoising methods in class. Among all the traditional models, nonlocal self-similarity models are very important. The NLM is one of the simplest and most effective of the self-similarity models.

BM3D is one of the state-of-the-art denoising methods. BM3D, first proposed in Ref. [2], is also a non-local denoising method, exploiting self-similarity in the image. Different from NLM, BM3D is a transfer-domain denoising method. BM3D makes use of the sparsity of the image in the transform domain, e.g. DCT transform. The sparsity is enhanced by blocking match, which is to group similar 2D blocks to 3D arrays. In the paper, the denoising performance of BM3D is shown to be better than other denoising methods such as BLS-GSM [4] and K-SVD [5].

The basic idea is to divide the image into square blocks of fixed size and group similar 2D blocks to 3D arrays by block matching and then apply 3D collaborative filtering.

Both BM3D and NLM works very well for images with regular and repetitive patterns.

2.2. Deep neural network denoising

The other kind of state-of-the-art denoising method is deep neural network denoising. Compared to self-similarity-based methods, these neural networks work better for images with irregular structures. In 2017, B. Ahn and N. I. Cho proposed a block-matching convolutional neural network (BMCNN) method that combines BM3D and CNN denoising [6].

A variety of neural network based denoising models are developed recently. For example, H. C. Burger et. al, proposed multi-layer perceptron (MLP) [7] for image denoising at specific noise level in 2012, which exhibits comparable performance as BM3D. In 2017, K. Zhang et. al, proposed the DnCNN model that can perform Gaussian denoising, using techniques such as residual learning [8] and batch normalization [9] are used in the model. In addition, to decrease the computational cost, we use dilated convolution in the CNN [10], which enables similar denoising performance with fewer number of convolution layers. In the following we discuss these techniques briefly.

DnCNN model with residual learning [8] estimates the Gaussian noise instead of the clean image. Suppose that the recovery of the clean image from the noisy image is denoted by the mapping $\mathcal{H}(y) = y - \eta$. Instead of estimating $\mathcal{H}(y)$, in residual learning, we estimate $\mathcal{F}(y) = y - \mathcal{H}(y) = \eta$. Residual learning can address the degradation problem in training deep neural networks.

When training a CNN, the distribution of a layer's inputs changes with the parameters of its previous layers. This internal covariate shift slows down the training. To overcome this issue, we can use batch normalization [9], which normalizes the inputs in a mini-batch by subtracting the mean and dividing by the standard deviation of the batch. Batch normalization brings in two additional parameters (scale and shift) to the layer.

The DnCNN model usually requires many layers. Using dilated convolution in the layers [10] can enlarge the receptive field and achieve comparable performance as the model with more layers. This helps reduce the computational cost.

3. Method

3.1. Non-local means (NLM)

The idea of NLM is a weighted average of all pixels in the image to get a denoised image z :

$$z = \frac{\sum_j \omega_{ij} y_j}{\sum_j \omega_{ij}} \tag{1}$$

and ω_{ij} is the weights that measures the similarity between the neighborhood around the i th pixel and the j the pixel:

$$\omega_{ij} = \frac{1}{Z_i} \exp\left(\frac{\sum_m k_m [v(y_i)_m - v(y_j)_m]^2}{h^2} \right), \quad Z_i = \sum_j \exp\left(\frac{\sum_m k_m [v(y_i)_m - v(y_j)_m]^2}{h^2} \right) \tag{2}$$

Here $v(y_i)$ is the neighborhood around the *i* the pixel in the image, m denotes the pixels in the neighborhood, k_m is a Gaussian kernel, and h controls the decay of the weight.

In the project, we use the restoration.denoise_nl_means function in the skimage module of Python.

3.2. BM3D

In BM3D denoising, block matching is realized by finding blocks that are similar to a reference block. The similarity is determined by finding the distance d between the two blocks. Given a threshold distance τ_{th} , two blocks with a distance smaller than the threshold distance ($d \le \tau_{th}$) is said to be similar. The similar blocks are stacked to form a 3D array, denoted by $\boldsymbol{Z}$. Then the collaborative 3D filtering is carried out to suppress the noise – we first DCT transform the 3D array, filter it using either hard-threshold filter or Wiener filter, and finally perform inverse-DCT transform. The idea is to shrink the coefficients in the DCT transform domain and filter out the noise components. Finally, an estimate of the clean image is obtained by weighted averaging the overlapping blocks.

When denoising color image with BM3D, we first transform the RGB image to YCrCb space, and the grouping is based on the information in the luminance channel only, since the luminance channel contains most of the information. The transformation matrix from RGB to YCrCb is:

$$A_{RGB2YCrCb} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.500 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.500 \end{pmatrix} \tag{3}$$

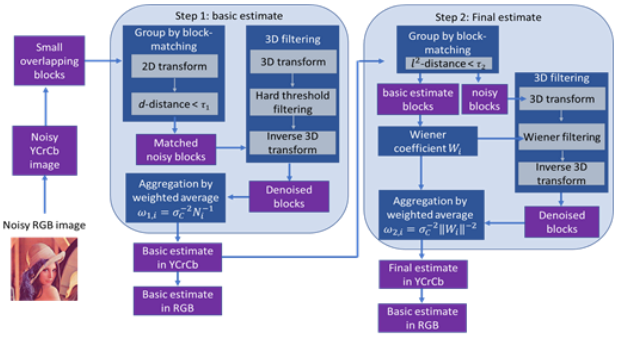


Figure 1: Flow chart of BM3D denoising algorithm.

Suppose we add independent Gaussian noise to RGB channels with a standard deviation of σ, then the variance of the noise in YCrCb channels are:

$$\begin{pmatrix}\sigma_Y^2\\ \sigma_{Cr}^2\\ \sigma_{Cb}^2\end{pmatrix}=\begin{pmatrix}0.299 & 0.587 & 0.114\\ 0.500 & -0.419 & -0.081\\ -0.169 & -0.331 & 0.500\end{pmatrix}\begin{pmatrix}\sigma^2\\ \sigma^2\\ \sigma^2\end{pmatrix}=\begin{pmatrix}0.669^2\\ 0.657^2\\ 0.623^2\end{pmatrix}\sigma^2 \quad (4)$$

We first slice the noisy YCrCb image in a set of overlapping blocks. Then the estimate consists of two steps – basic estimate and final estimate – as shown in the flowchart in **Figure 1**.

In the first step (Step 1), the blocks are grouped using the similarity criteria based on the *d*-distance between two blocks B_i and B_j defined as

$$d(B_i,B_j)=\frac{\left\|T_{2D}(B_i)-T_{2D}(B_i)\right\|_2^2}{N_{block1}^2} \quad (5)$$

where T_{2D} denotes a 2D transform and N_{block1} is the block size used in Step 1. Then we apply 3D transform to the matched noisy blocks and use hard-threshold filtering in the transform domain, based on assumed sparsity there. Denoised blocks can be obtained from inverse 3D transform. Then the blocks are stitched together to form the basic estimate using weighted average with a weight of

$$\omega_{1i}=\sigma_C^{-2}N_i^{-1} \quad (6)$$

where σ_C is the standard deviation for channels $C=Y,Cr,Cb$ as defined in Eq. (4), and N_i is the number of nonzero coefficients kept in the transform domain.

In the second step (Step 2), since we already have a rough estimate of the clean image, we use the basic estimate to perform the block-matching, and create 3D arrays (B_{noisy} and B_{basic}) from the noisy image and the basic estimate, respectively. This improves the grouping compared to Step 1. Then we apply 3D filtering to B_{noisy} in the transform domain. The difference with Step 1 here is that we use empirical Wiener filtering, where the Wiener shrinkage coefficients is calculated using B_{basic}:

$$W_C=\frac{\left|T_{3D}(B_{basic})\right|^2}{\left|T_{3D}(B_{basic})\right|^2+\sigma_C^2}. \quad (7)$$

Finally, we use weighted average to get the final estimate. The use of Wiener filtering gives more accurate filtering than hard-threshold filtering in Step 1. As a result, the final estimate outperforms the basic estimate.

We implement the full BM3D model in Python for color image denoising. The results are compared to other methods in Section 4.

3.3. DnCNN methods

We implement two different structures of neural networks: one is the DnCNN proposed in Ref. [3]; the other one makes use of dilated convolution to decrease the computational cost, which we call DnCNN-d.

The first network (DnCNN) is schematically illustrated in **Figure 2** (a). A total of 17 layers are used here. We use 64 filters of size $3\times3\times c$ in the 1st layer, 64 filters of size $3\times3\times64$ in the 2nd to 15th layer, and finally c filters of size $3\times3\times64$ in the 16th layer. Zero padding are used in the convolution so that the output has the same size as the input. The receptive field of the model is 33×33. Here c is the number of channels in the image. In our color image case, $c=3$. Batch normalization is used in the layers to help with the issue of internal covariate shift that slows down the training process.

The architecture of the second network (DnCNN-d) is shown in **Figure 2** (b). We use dilated convolution in the network [10] to increase the receptive field. A dilated convolution with a dilation rate of *n* is denoted by DConv-*n*. With dilated convolution, a DnCNN-d network with 7 layers has a receptive field of 33×33, same as that of DnCNN with 17 layers.

We use Keras API in Python for both models. The mean squared error between the CNN output and the target clean image is used as the loss function for training. We use Adam optimizer in the training, which is an adaptive learning rate optimization algorithm. For training the networks, we use the 400 images train and test images from BSDS500 [11], while the rest 100 images are left for evaluation.

To train the network for denoising at a given noise level, we add Gaussian noise with standard deviation at the given noise level, so we have a pair of clean and noisy images. The noise in RGB channel are added independently. We then generate overlapping patches of size $50\times50\times c$ with a 10×10 stride (no overlap in the third dimension) for each image. A total of 492,800 patches are generated for the set of clean images, as well as the set of noisy images, which are then grouped into 3850 mini-batches of 128 patches. The input of the training is patch pairs – a clean patch and a noisy patch.

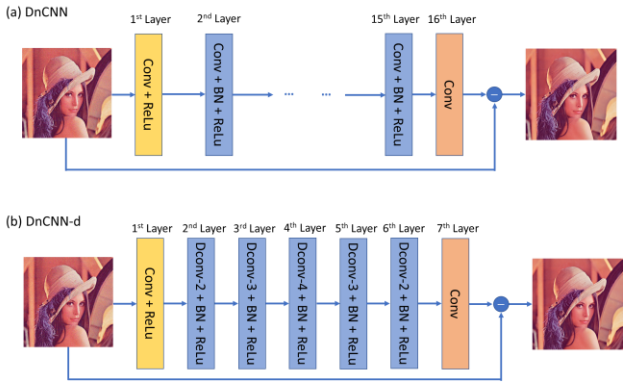


Figure 2: The architectures of (a) DnCNN network and (b) DnCNN-d network.

3.4. Comparison metrics

Two popular objective image quality metrics are PSNR and SSIM. PSNR between two images x and y can be calculated from

$$PSNR = 10\log_{10}\left(\frac{\max(I_{original})^2}{MSE_{xy}}\right). \quad (8)$$

The SSIM can be calculated using

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}. \quad (9)$$

where μ_α and σ_α are the mean and standard deviation of image $\alpha = x, y$, and σ_{xy} is the covariance of x and y. $c_i = (k_i L)^2, i = 1,2$ with $k_1 = 0.01$, $k_2 = 0.03$, and L being 255.

4. Results and discussion

4.1. Training time and testing time

The traditional methods (NLM and BM3D) does not require training, while the two neural networks (DnCNN and DnCNN-d) needs to trained first. As there are fewer layers in DnCNN-d, the training time is almost half of the training time for DnCNN, as shown in **Table 1**. The average testing time is also less for DnCNN-d. In terms of testing time, DnCNN-d requires shortest testing time, and the traditional methods are much slower than neural networks. We also notice that the BM3D method we implement is slower than the other methods. We will need to modify our implementation so that it is more computational efficient in the future.

	Training time (ms/step)	Average testing time (sec)
DnCNN	803	1.583
DnCNN-d	423	0.9528
NLM	-	32.78
BM3D	-	243.5

Table 1: Comparison of training time and testing time with DnCNN and DnCNN-d.

4.2. Number of epochs

The loss function decreases with the number of epoch. As can be seen from **Figure 3**, the loss function is converging around 25 epochs. Therefore, for the rest of the report, we use 25 epochs in the training.

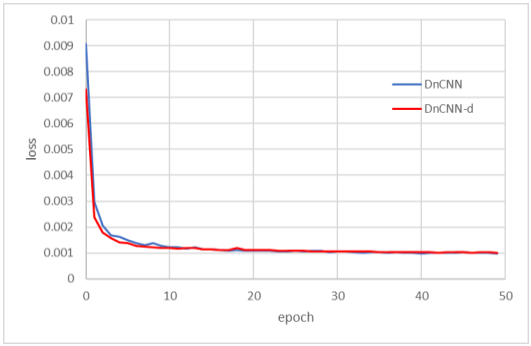


Figure 3: The architectures of (a) DnCNN network and (b) DnCNN-d network.

4.3. Gaussian denoising performance

For evaluation of denoising performance, we use the rest 100 images in CBSD500 dataset. We calculate the average PSNR and SSIM, and also the average testing time of the denoised images for each denoising method we discussed above, the results of which are summarized in Table 2. In terms of PSNR and SSIM, the deep neural network based methods have better performance than traditional denoising techniques. In particular, DnCNN performs the best. DnCNN-d has similar performance with DnCNN, which confirms that dilated convolution can be used to reduce the computational cost of DnCNN denoising, given that the training and testing time are both reduced greatly as discussed in Sec. 4.1. Comparing the two traditional methods, we find that BM3D gives overall better denoising results than NLM.

	NLM	BM3D	DnCNN	DnCNN-d
PSNR(dB)	27.90	28.20	31.00	30.85
SSIM	0.7611	0.8126	0.9324	0.9318

Table 2: Comparison of average PSNR, SSIM, and testing time of NLM, BM3D, DnCNN and DnCNN-d using test images in CBSD500 dataset with noise level of 25.

	NLM	BM3D	DnCNN	DnCNN-d
PSNR(dB)	24.25	24.54	27.56	27.62
SSIM	0.6246	0.6509	0.8668	0.8658

Table 3: Comparison of average PSNR and SSIM of NLM, BM3D, DnCNN and DnCNN-d using test images in CBSD500 dataset with noise level of 50.

		NLM	BM3D	DnCNN	DnCNN-d
Clay pot	PSNR	32.69	32.64	35.09	34.90
	SSIM	0.8227	0.8368	0.9574	0.9550
Stone carving	PSNR	22.55	24.01	27.68	27.53
	SSIM	0.8314	0.8578	0.9357	0.9396

Table 4: Comparison of PSNR and SSIM for of two image examples in the CBSD500 dataset with noise level 25.

Examples of testing images and denoised results are shown in Figure 4. The quantitative comparison is shown in Table 4. As can be seen from both PSNR and SSIM, neural networks perform the best. In the case of the clay pot image, which has less details and more regular patterns, the NLM method works well, but lots of details in the image are removed. The result of BM3D also loses some details. The two neural networks give similar results. For the stone carving, which has a lot of details and irregular patterns, there is still significant amount of noise in the NLM result. The BM3D result appears to be over-smoothed. NLM and BM3D has better performance when the image has regular and repetitive patterns.

4.4. Blind denoising

The network trained for denoising at noise level 25 cannot properly denoise images with noise level 50, as shown in the **Figure 5**. Similar in BM3D, if we set $\sigma = 25$, there are still significant amount of noise in the denoised image. The PSNR of the denoised result of DnCNN-d and BM3D are 22.01 and 21.82, respectively. These methods do not perform well if we do not know a priori what the noise level of the image is. In order to perform 'blind' denoising, we train a DnCNN-b network using the same structure as DnCNN-b. We set the noise level of the training images to a uniform random distribution between 0 and 55. The denoised image is shown in **Figure 5** (c). In addition, we show the results of the denoising performance of the network at noise level 25 and 50 as summarized in **Table 5** and **Table 6**, respectively. As can be seen, DnCNN-b has a similar performance as the DnCNN-d network trained at the correct noise level, and gives much better results than the DnCNN-d network trained at an incorrect noise level.

In **Figure 6**, we show the results of DnCNN-b denoising applied to an example image with noise level 15, 25, and 50, respectively. Our trained DnCNN-b works well for noise level below 50. We notice that When the noise level is 50, the denoised images show some artefacts and the result is over-smoothed.

5. Discussion and future work

We have implemented two state-of-art denoising methods: a deep learning method DnCNN and a traditional BM3D method. We have shown that DnCNN performs better than traditional denoising methods BM3D and NLM; and BM3D has an overall better performance than NLM. For DnCNN,

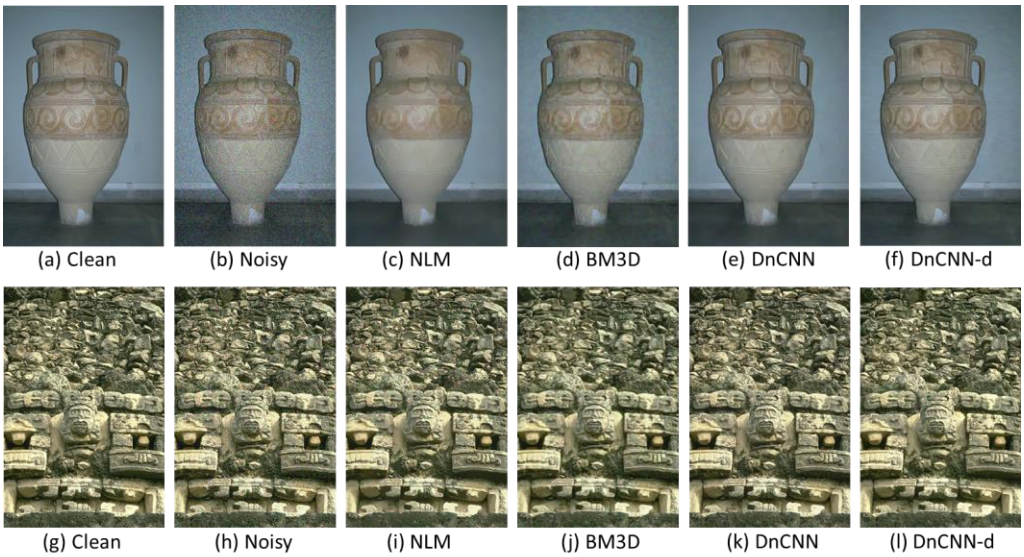
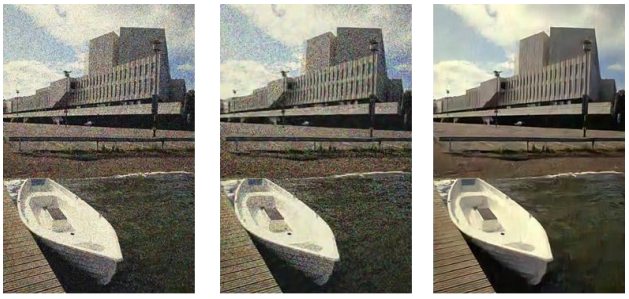


Figure 4: Color image denoising results of two image examples from the CBSD500 with noise level 25.



(a) DnCNN-d (b) BM3D (c) DnCNN-b

Figure 5: Resulting of denoising images at noise level 50 using (a) DnCNN-d trained at noise level 25, (b) BM3D model with $\sigma = 25$, and (c) DnCNN-b.

`

Figure 6: Resulting of denoising images at noise level 15, 25, and 50 using DnCNN-b.

	DnCNN-b	DnCNN-d trained for σ=25	DnCNN-d trained for σ=50
PSNR (dB)	30.77	30.85	27.77
SSIM	0.9293	0.9318	0.8415

Table 5: Average PSNR and SSIM of DnCNN-d trained for noise level 25 and DnCNN-d using 100 test images in CBSD500 dataset with noise level of 25.

	DnCNN-b	DnCNN-d trained for σ=50	DnCNN-d trained for σ=25
PSNR (dB)	27.41	27.62	21.65
SSIM	0.8641	0.8658	0.5992

Table 6: Average PSNR and SSIM of DnCNN-d trained for noise level 50 and DnCNN-d using 100 test images in CBSD500 dataset with noise level of 50.

we have trained two different structures of networks (DnCNN and DnCNN-d) for noise levels 25 and 50, respectively. DnCNN-d makes use of dilated convolution to reduce the computational cost without compromising the performance of the network. We have also trained a DnCNN-b network for blind denoising, which applies to all noise levels between 0 and 55.

We also note that DnCNN requires the shortest testing time, while BM3D is the least computational efficient. In the future, we will modify our implementation to reduce the computational cost of BM3D.

One thing we found in training DnCNN is that the denoising performance is affected greatly by the training data we chose. Also, we notice that the denoised image from a high noise level image shows some artefacts and is over-smoothed. In the future, we would like to use different training sets and access the performance of the networks to try to mitigate this issue.

Self-similarity based denoising methods works better for images with regular patterns, but may over-smooth images with a lot of details; DnCNN methods, on the other hand, works very well for images with irregular patterns. Combining the two methods may give a denoising method that outperforms each of them. We would like to investigate as future work as well.

Acknowledgement

We would like to thank Hayato Ikoma for his support in this work.

References

[1] A. Buades, B. Coll and J. -. Morel, "A non-local algorithm for image denoising," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, pp. 60-65 vol. 2.

[2] Dabov, Foi, Katkovnik, Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering", IEEE Trans. Im. Proc. 2007.

[3] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," IEEE Transactions on Image Processing, Volume: 26, Issue: 7, July 2017.

[4] J. Portilla, V. Strela, M. Wainwright, and E. P. Simoncelli, "Image denoising using a scale mixture of Gaussians in the wavelet domain," IEEE Trans. Image Process., vol. 12, no. 11, pp. 1338-1351, November 2003.

[5] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," IEEE Trans. on Image Process., vol. 15, no. 12, pp. 3736-3745, December 2006.

[6] B. Ahn, N. I. Cho, "Block-matching convolutional neural network for image denoising", CoRR, vol. abs/1704.00524, 2017, [online] Available: http://arxiv.org/abs/1704.00524.

[7] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with BM3D?" *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2392–2399.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International Conference on Machine Learning*, 2015, pp. 448–456.

[10] T. Wang, M. Sun and K. Hu, "Dilated Deep Residual Network for Image Denoising," *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Boston, MA, 2017, pp. 1272-1279.

[11] The Berkeley Segmentation Dataset and Benchmark, https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/