IMAGE DENOISING METHOD USING CNN AND RESIDUAL BLOCKS

Group number: 18
Group member:
1. An Le - 93258317 – antl2@uci.edu
2. Son Le – 93351358 – lesc@uci.edu

I. Introduction

Abstract- Low-light photography is always challenging because of noise. Usually, cameras will take photos with high ISO sensitivity to compensate low brightness, but at the same time amplify noise signal. Photographers can also keep ISO low by increasing exposure time or aperture, however, result photos can be blur due to motion or depth of field. Therefore, many post-processing methods have been developed to restore original quality of noised imaging, and this has been an active area of research for years. In recent years, researchers in convolution neural network and deep learning have proposed many successful deep learning methods to denoise photos. Our project aims to explore denoising problem and propose a pipeline based on materials presented in CS 175 project course at UCI. Using presented dataset, our network learns and processes directly on RGB images with real life noise and produce restored images. We evaluate and compare our method to BM3D[4] method.

2. Related Work:

Being a long-time developed field, image denoising has many state of the art methods without using deep learning. BM3D[4], WNMM[5] are novel image denoising methods based on sparsity or low rank. Among them, BM3D is often used as a benchmark for comparing denoising algorithms. In our evaluation, we will compare our method's performance to a set of BM3D pre-denoised images.

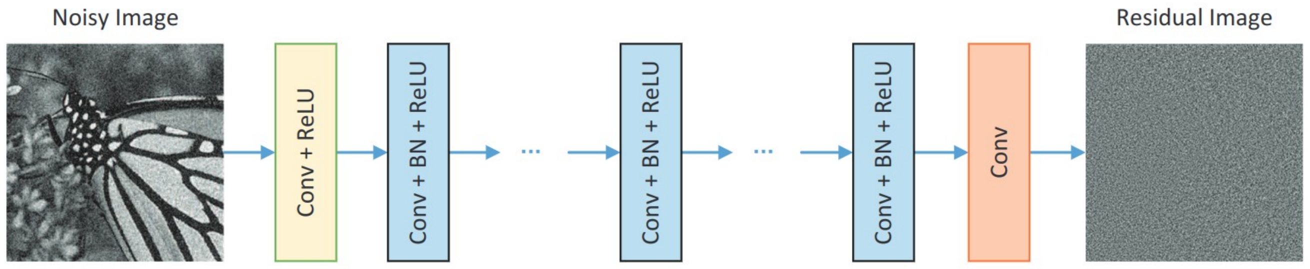


Figure 1: DnCNN Architecture

Moreover, with recent trend in deep learning, many highly effective neural network pipelines have been proposed to outperform all traditional methods. DnCNN[1], Fig. 1, is one of the first architectures that uses convolution network. The network consists of 17 fully convolution layers with the use of BatchNorm and ReLU to learn noisy images and estimate noise maps to restore original image quality. Results are compared to other popular methods, including BM3D. Although DnCNN outperforms BM3D, the network is trained on images with noise generated using Gaussian distribution. Thus, the performance in real world imaging is not yet evaluated.

Another simple and successful method is proposed in the paper ‘Learning to See in The Dark’ which also uses fully convolution network but with up-sampling and down-sampling layer. The pipeline take raw images with real noise as input and produce the final RGB output. Results are compared against BM3D denoising as well as other methods.

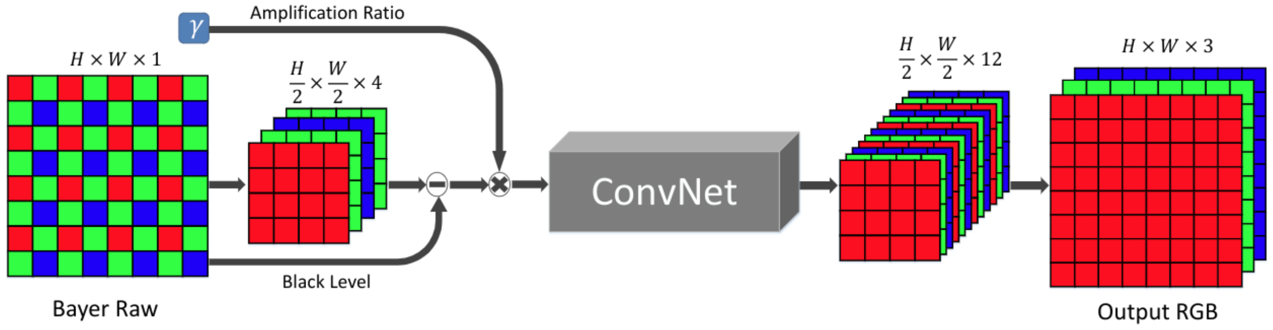


Figure 2: The pipeline proposed in the paper 'Learning to see in the dark'

Given the highly active area of research and the above successful pipelines, our project does not aim to propose any new method to outperform developed ones. Instead, we apply what we have learned in CS 175 course as well as online materials to form a simple, effective and understandable pipeline. However, we also challenge ourselves to train the pipeline on real world noisy images and then compare the performance with BM3D, a state of the art algorithm.

3. Data Sets

As mentioned, we use real noisy images in our project as challenge. The RENOIR data set, by Josue Anaya et al., contains real-world noisy images taken using consumer cameras. The data set has a total of 1500 photos including 300 pairs of ground truth and noisy images with five different noise levels. The data set is popular and among the best benchmark ones available to public use. However, since the size of each image is approximate 5200 x 3400, 55 MB, we will limit our project to use the aligned “Canon T3i” sub collection. The collection has 40 batches, and each batch contains noisy and ground truth images.

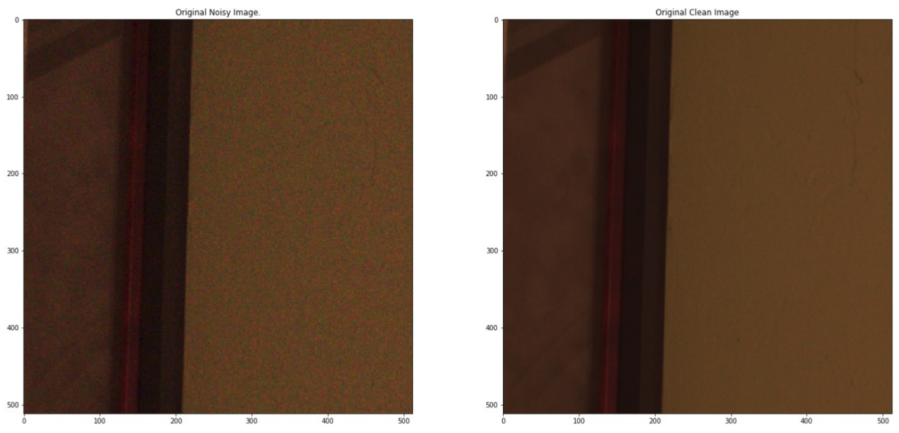


Figure 3: A 512 x 512 patch of noisy and clean images from 'Canon T3i' subcollection, RENOIR data set

Besides, to compare our method’s performance against BM3D, we also use a set of images available at BM3D official page. The Lena image has one ground truth, 15 noisy variation with different generated noise levels, and 15 corresponding denoised image using BM3D method (CBM3D.m). Further explanation of how we use this dataset will be presented in later section.

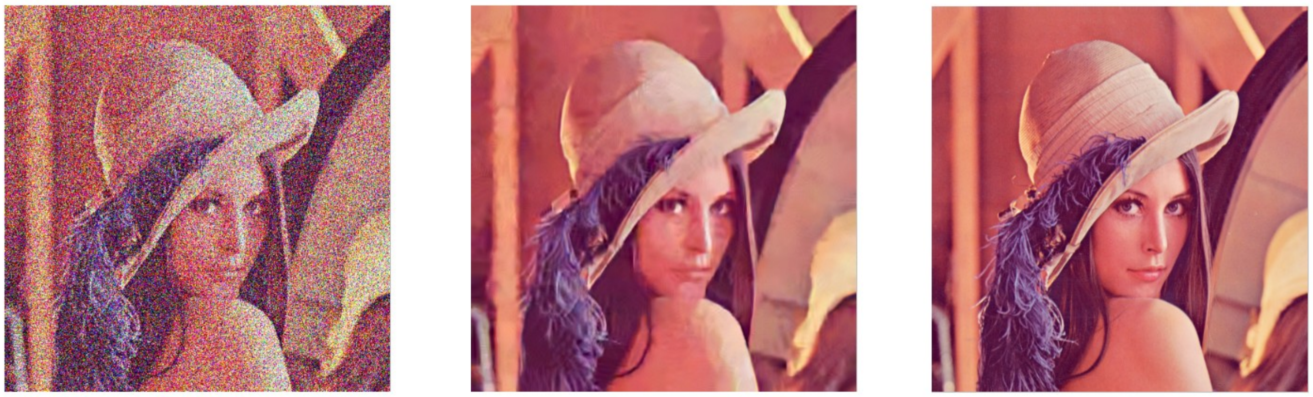


Figure 4: Set of images from BM3D official page. (left) Noisy Image with noise level 100, (middle) Denoised image using BM3D method, (right) Original clean image

Dataset	Subset	Source	Note
RENOIR data set	‘T3i’	http://ani.stat.fsu.edu/~abarbu/Renoir.html	- 90% for training data, 10% for testing data. - Total size is 4.3 GBs
BM3D images	‘Lena’	http://www.cs.tut.fi/~foi/GCF-BM3D/	- Used to compare our method with BM3D. - 15 set of images with different noise levels downloaded by right-clicking on links in table 2 on BM3D page. - Total size is approximately 20 MBs

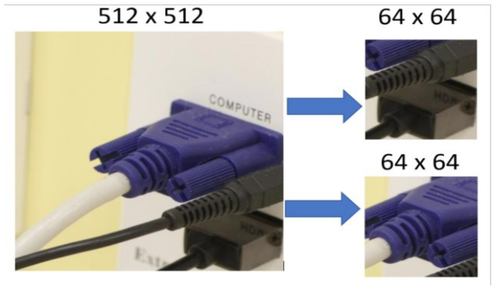
The provided link is original source………..

4. Description of Technical Approach

a. Data Preparation

For training, we use 36 pairs of image from T3i set. We choose to not crop the images into smaller patch, but instead, we use PyTorch built-in transformations to produce augmented patches during training time. Moreover, we randomly choose 4 batches as testing data to tune the hyper parameters of our network.

For data augmentation, we apply PyTorch RandomCrop and ColorJitter each time data loader gets a datapoint for training epoch. For an original image size of 5200 x 3400, a RandomCrop transformation with size 128 x 128 may yield us approximately (5200 x 3400 / 128 / 128) = 1000 possible regions. We also understand that real life images always have different brightness, contrast and saturation levels. Therefore, we use ColorJitter to apply random color transformation to training data. Also, we notice that the majority of images are dark and black. Therefore, we use uniform probability to give 40% chance that a generated patch will be increased greatly in brightness. During the training process, we In general, by using RandomCrop, ColorJitter and shuffle data loader, we are able to get many possible combination of training data for many epochs.



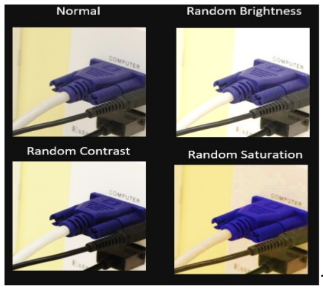
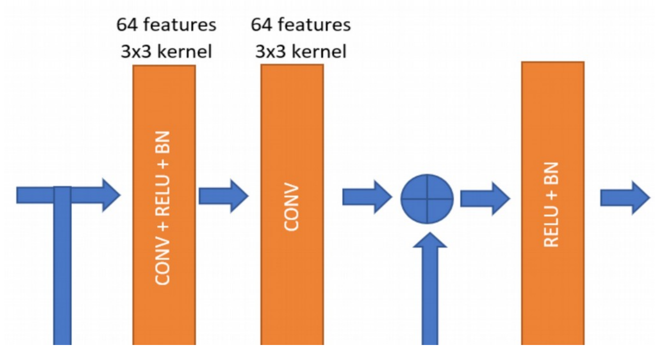


Figure 5: (Left) RandomCrop and (Right) ColorJitter Illustration

b. Network Architecture

We use Residual Block which was presented in the course as a different approach in our architecture compared to DnCNN and other methods available online as far as we know. Fig 6 is the residual block that we use. It is commonly believed that residual learning helps avoid vanishing gradient and thus deeper neural network can converge faster. Another important technique which also aids training process is ReLU activation function and Batch Normalization. We utilize both of these trendy techniques in our proposed network.



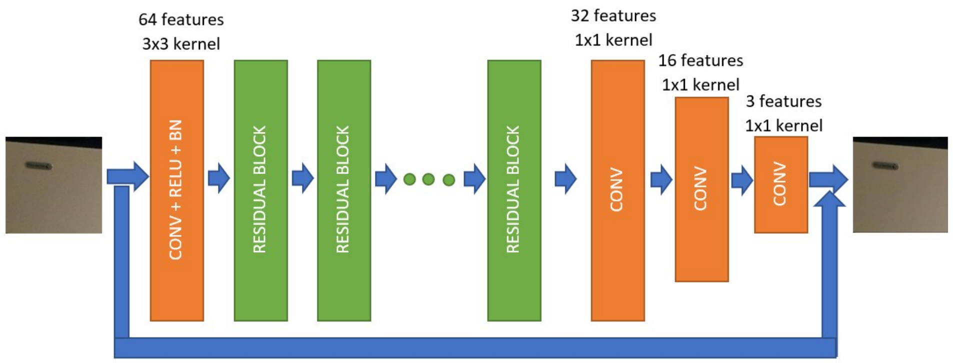


Figure 7: Our proposed network architecture

With five residual blocks, our network consists of 14 convolution layers. Our network take in RGB image as input, and produce RGB image as output. We also use skipping connection from the input to the last convolution layer. This skipping connection can also be interpreted as learning the residual or noisy map of images, which is explained in later section. The hyper parameters of our network include:

1. output features (default=64)
2. number of residual blocks (default=5)
3. input size(default =128x128)
4. ColorJitter (Default=True)
5. RandomCrop (default=True)
6. Kernel size(default=3x3)

c. Objective Function

In DNN paper, Zhang et al. states that a noisy observation can be expressed as:

$$y = x + v$$

where y is the noisy image, x is the clean image and v is the noise. In our case, our network will have to estimate v' which is the noise in order to produce an approximately denoised x'.

$$x' = y - v'$$

We implement this idea as the skipping connection mentioned in our architecture. In fact, the output after the last convolution network is subtracted from the original noisy image. Using this technique helps us avoid vanishing gradient, and also instead of estimating the final image, the network basically estimates the residual or the noise map v'. The code below will illustrate the residual idea.

```
def forward(self, x)
    output = self.model(x)
    output = x – output
    return output
```

Finally, our objective function is L1Loss since we are comparing the noisy image and the estimated image by pixels. The L1Loss also helps improve overall training time.

$$L1Loss = \frac{1}{N} \sum_{i=1}^{N} (y_{output} - y_{truth})$$

d. Training

We use Adam optimizer, a recent trend in deep learning, to converge faster. There are several parameters relating to the training process:

1. batch_size (default = 5)
2. shuffle (default = True)
3. numer of epochs (default = 500)
4. learning rate (default = 1e-03)
5. learning rate decay multiplier (default = 0.985)

4. Software

A table will help us organize this section.

File	Sections	Our code / External code	Description
Model.py	Class ResidualBlock	Code available from class materials	We use the code presented in class to form our residual block. However, we have modified the order of BatchNorm2d and ReLU.
Model.py	Class MyModel	Our code	This class defines the overall structure of our network.
Dataset.py	Functions about data preparation	Our code	Load images and prepare Pytorch Dataset
Helpers.py	Psnr	External code	To measure PSNR value
Helpers.py	Functions about image comparison and displays	Our code	To use in notebook lab to display data and metrics.
Notebook.ipynb	Main notebook	Our code	To train, test and evaluate the model.

List of external software and libraries used:

Glob	sudo pip3 install glob3	Library used to interact with files
Python	https://www.python.org/	Programming language
PyTorch	https://pytorch.org/	Deep learning framework
PSNR	https://github.com/aizvorski/video-quality/blob/master/psnr.py	PSNR measurement

5. Experiments and Evaluation

a. Measurement

Peak signal-to-noise ratio (PSNR) is used to measure the quality of reconstruction of lossy compression codecs (e.g., for image compression). We use this metrics to measure the improvement of our denoised output over the noisy input. According to Wikipedia, the formula is as follow:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

PSNR(x, y) is a value calculated from two comparable images. The higher PSNR (in db), the better an image is denoised. For example, PSNR(original, original) will be 100 db. PSNR(original, noisy) will be much lower, and PSNR(original, denoised) should be higher if there is any improvement.

b. Training experiments

We train the network on various combination of hyper parameters to determine the best one. The train data set is 36 images, and the test data is 4 images. The detail PSNR values of test images will help determine the best parameter values (green color)

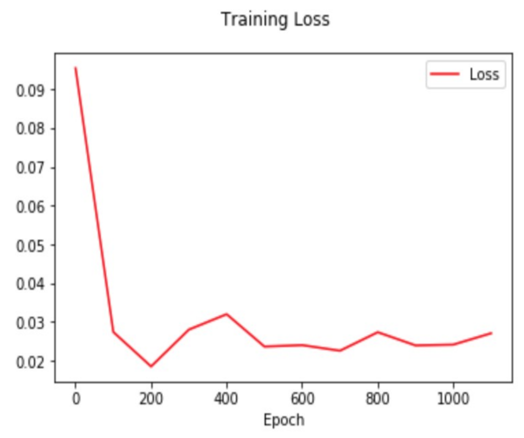
		PSNR values of Test Data Image							
Hyper-parameter	**Value**	**Noisy 0**	**Denoised 0**	**Noisy 1**	**Denoised 1**	**Noisy 2**	**Denoised 2**	**Noisy 3**	**Denoised 3**
Input Image Size	**32x32**	75.48	**83.7**	81.32	**88.07**	78.64	**86.33**	76.85	**82.53**
Input Image Size	64x64		82.99		87.43		85.59		82.39
Input Image Size	128x128		83.27		37.97		85.9		82.52
Output Features	**64**	75.48	**83.7**	81.32	88.07	78.64	**86.33**	76.85	82.53
Output Features	32		83.36		**88.27**		85.66		82.3
Output Features	96		83.36		87.05		86.09		**82.54**
Num of Res Blocks	**5**	75.48	**83.7**	81.32	88.07	78.64	**86.33**	76.85	82.53
Num of Res Blocks	3		83.57		**88.47**		86.06		**82.68**
Num of Res Blocks	9		8315		87.83		85.75		82.49
Learning rate	**1.00E-03**	75.48	**83.7**	81.32	**88.07**	78.64	**86.33**	76.85	**82.53**
Learning rate	1.00E-02		77.55		82.68		80.35		77.21
Learning rate	1.00E-04		80.38		81.64		82.75		79.99
learning rate mult	**0.985**	75.48	**83.7**	81.32	**88.07**	78.64	**86.33**	76.85	**82.53**
learning rate mult	0.75		76.28		80.19		78.76		76.46
learning rate mult	0.99		8.3		86.83		85.69		82.37
RandomCrop	**On**	75.48	**83.7**	81.32	**88.07**	78.64	**86.33**	76.85	**82.53**
RandomCrop	Off		82.88		87.58		85.62		82.33

ColorJitter	**On**	75.48	**83.7**	81.32	**88.07**	78.64	**86.33**	76.85	**82.53**
ColorJitter	Off		83.53		87.85		86.72		82.54

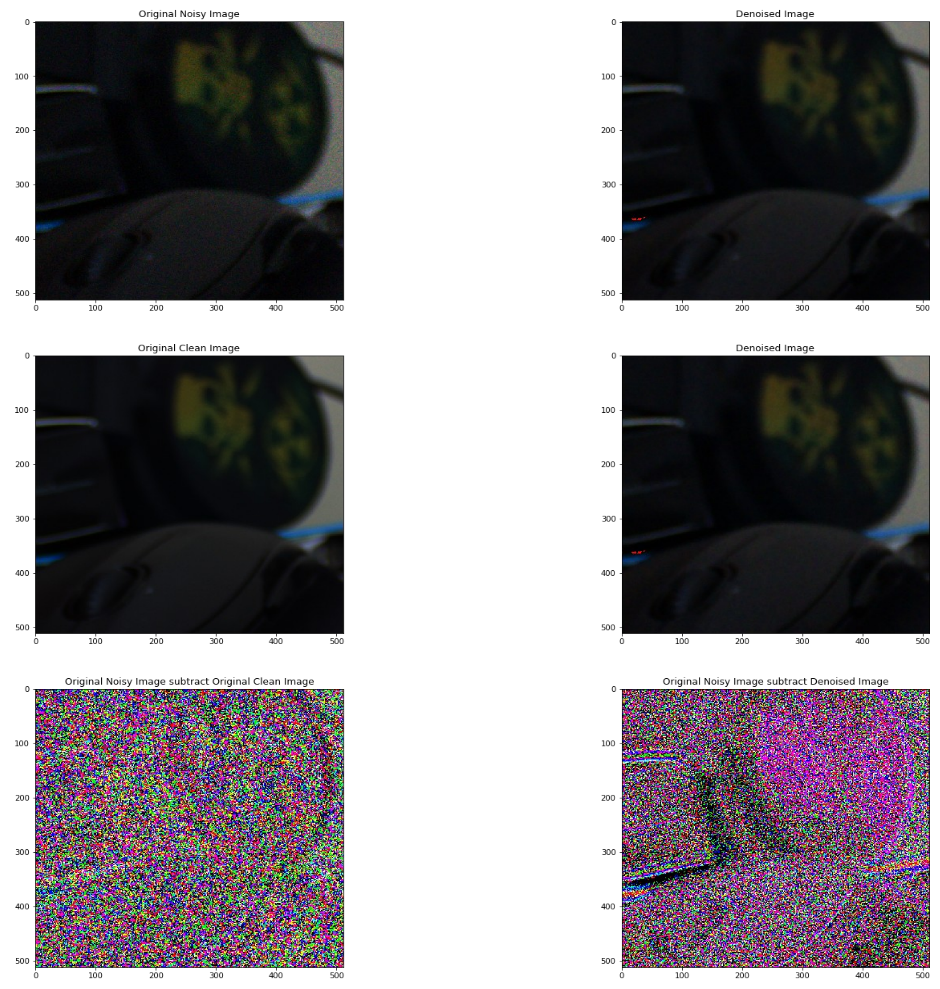
b. Final model evaluation

With the chosen parameters, we proceed to train the network for 1200 epochs and obtain the following result.

PSNR values of Test Data Image							
Noisy 0	**Denoised 0**	**Noisy 1**	**Denoised 1**	**Noisy 2**	**Denoised 2**	**Noisy 3**	**Denoised 3**
75.48	**82.72**	81.32	**87.46**	78.64	**85.60**	76.85	**82.07**



Although we use the same parameters, but the model network trained for 1200 epochs has greater training loss than 200 epochs. Also, the PSNR values over test images show a decrease in improvement with more epochs. We make assumption that with more epochs, data augmentation with color jitter tends to direct the neural network away from fitting the actual train and test data set. Moreover, there are red color artifacts appear in the denoised image. Our group has addressed this problem in the presentation, but despite many approaches, we have not found a solution so far .


Original Noisy Image
Denoised Image
Original Clean Image
Denoised Image
Original Noisy Image subtract Original Clean Image
Original Noisy Image subtract Denoised Image

c. Performance comparison

We extend our evaluation by comparing our network to the state of the art method BM3D. We use Lena set downloaded from BM3D website, and obtain the following PSNR improvements. It is clear that BM3D is a winner here. However, our network has produced good result for all noise levels, except one. It is also important to note that the noise here is generated using Gaussian distribution, not a real life noise by the specific camera T3i that our neural network is trained on. We have high hope that with further training, deeper model and better data set will help us improve the performance.

Num	Noisy PSNR	Our Denoised PSNR	BM3D PSNR
0	76.29	77.98	**83.33**
1	58.41	64.73	**75.24**
2	72.82	76.58	**82.06**
3	70.38	75.28	**81.14**
4	68.51	74.11	**80.40**
5	67.00	73.06	**79.72**
6	65.74	72.12	**79.04**
7	64.68	71.27	**78.24**
8	82.28	**79.40**	**85.92**
9	62.96	69.76	**78.01**
10	61.62	68.46	**77.34**
11	60.56	67.33	**76.76**
12	60.11	66.82	**76.49**
13	59.70	66.34	**76.25**
14	59.00	65.48	**75.78**

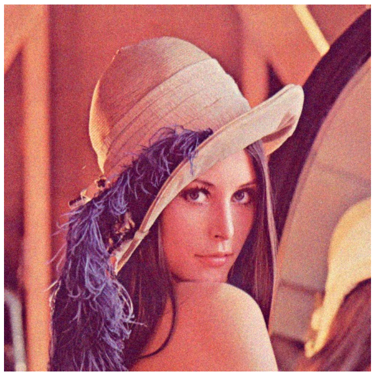


Figure 9: Noisy Image (Num=2)

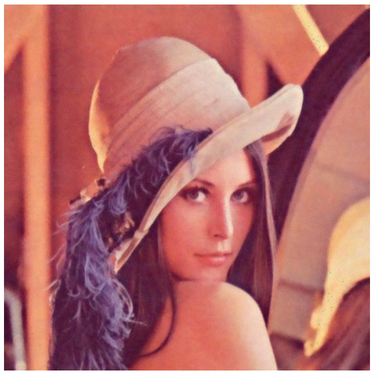


Figure 10: Our denoised image (Num=2)

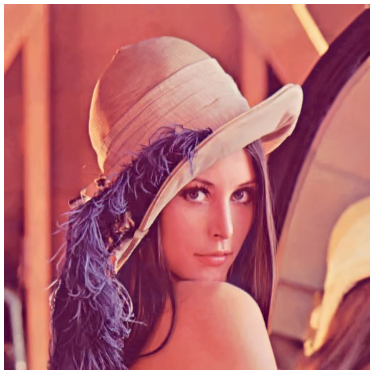


Figure 8: BM3D denoised image (Num=2)

6. Discussion and Conclusion

Starting from scratch and without prior knowledge to PyTorch and deep learning, we have learned and gained a lot of experience from the project. We learn that deep neural network is not as easy as putting bricks together and expect it to work effectively. We had to explore and finely tune our model in order to achieve good result. We also learn that data sets play an imperative role in machine learning, and only with good data set can we obtain feasible result. Moreover, despite all the effort we put into the project, our output occasionally has color artifacts. We have not yet understood and solved this problem thoroughly. We have even tried to find a solution to this color problem by looking at DnCNN implementation, but we cannot figure out any reason. These color artifacts are the biggest limitation of our approach. On the other hand, our project performs relatively well on the data set provided. If we are in charge of a research lab, what we would do next is to gather more data before trying deeper and more complicated model. After all, we believe data set is more important than the model itself.

WORK CITED

1. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising (TIP2017), Zhang et al.
https://arxiv.org/pdf/1608.03981v1.pdf
2. Learning to See in the Dark, Chen Chen, Qifeng Chen, Jia Xu, Vladlen Koltun
https://arxiv.org/abs/1805.01934
3. Peak signal-to-noise ratio, From Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
4. Image restoration by sparse 3D transform-domain collaborative filtering (SPIE Electronic Imaging 2008), Dabov et al.
http://www.cs.tut.fi/~foi/GCF-BM3D/SPIE08_deblurring.pdf
http://www.cs.tut.fi/~foi/GCF-BM3D/
5. Weighted Nuclear Norm Minimization with Application to Image Denoising (CVPR2014), Gu et al.
http://www4.comp.polyu.edu.hk/~cslzhang/code/WNNM_code.zip
6. RENOIR - A Dataset for Real Low-Light Image Noise Reduction (Arxiv 2014), Anaya, Barbu.
https://arxiv.org/pdf/1409.8230.pdf