DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
ECE 211A: DIGITAL IMAGE PROCESSING I

INSTRUCTOR: Prof. Achuta Kadambi
TA: Yunhao Ba

NAME: Qiong Hu
UID: 405065032

HOMEWORK 1: IMAGE DECONVOLUTION

PROBLEM	TOPIC	MAX. POINTS	GRADED POINTS	REMARKS
2.1	Blurry Image	1.0		
2.2	Metric Baseline	1.0		
3.1	Naive Deconvolution Algorithm	1.0		
3.2	Naive Deconvolution Results	1.0		
3.3	Naive Deconvolution Analysis	1.0		
4.1	Wiener Filter Algorithm	1.0		
4.2	Ideal Wiener Filter Results	1.0		
4.3	Power Spectral Density	1.0		
4.4	SNR Approximation	1.0		
4.5	Approximation Results	1.0		
	Total	10.0		

1 Problem Setup

Under LSI conditions, deblurring can be posed as follows:

$$y(n_1,n_2) = h(n_1,n_2) * * x(n_1,n_2) + e(n_1,n_2), \tag{1}$$

where $y(n_1,n_2)$ is the observed image, $x(n_1,n_2)$ is the original image, $h(n_1,n_2)$ is the kernel, and $e(n_1,n_2)$ is the noise.

To recover the original image, we usually apply deconvolution algorithms to the observation. In this homework, we will implement several deconvolution algorithms and compare the performance of them. To evaluate the performance quantitatively, we will adopt two image similarity metrics: (1) peak signal-to-noise ratio (PSNR), and (2) structural similarity (SSIM) index.

PSNR between two images, $x_1(n_1,n_2)$ and $x_2(n_1,n_2)$, can be calculated as follows:

$$\begin{aligned} PSNR(x_1,x_2) &= 10\log_{10}(\frac{R^2}{MSE(x_1,x_2)}) \\ MSE(x_1,x_2) &= \frac{1}{N_1N_2}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}[x_1(n_1,n_2)-x_2(n_1,n_2)]^2, \end{aligned} \tag{2}$$

where R is the data range of the images, and (N_1, N_2) is the image size.

SSIM between $x_1(n_1,n_2)$ and $x_2(n_1,n_2)$ is calculated based on three measurements, including luminance (l), contrast (c), and structure (s):

$$\begin{aligned} SSIM(x_1,x_2) &= [l(x_1,x_2)]^\alpha \cdot [c(x_1,x_2)]^\beta \cdot [s(x_1,x_2)]^\gamma \\ l(x_1,x_2) &= \frac{2\mu_{x_1}\mu_{x_2}+C_1}{\mu_{x_1}^2+\mu_{x_2}^2+C_1} \\ c(x_1,x_2) &= \frac{2\sigma_{x_1}\sigma_{x_2}+C_2}{\sigma_{x_1}^2+\sigma_{x_2}^2+C_2} \\ s(x_1,x_2) &= \frac{\sigma_{x_1x_2}+C_3}{\sigma_{x_1}\sigma_{x_2}+C_3}, \end{aligned} \tag{3}$$

where α, β, and γ are the weights for the three measurements, μ_{x_1}, μ_{x_2}, σ_{x_1}, σ_{x_2}, and $\sigma_{x_1x_2}$ are the local means, standard deviations, and correlation coefficient for images $x_1(n_1,n_2)$ and $x_2(n_1,n_2)$, and C_1, C_2, and C_3 are three variables to stabilize the division.

You can refer to PSNR and SSIM functions in the scikit-image package[1] for more details. Make sure you have changed the corresponding parameters of the SSIM function in scikit-image to match the implementation of [3]. Remember to normalize the images before reporting the PSNR and SSIM scores.

[1] `https://scikit-image.org/docs/dev/api/skimage.metrics.html`

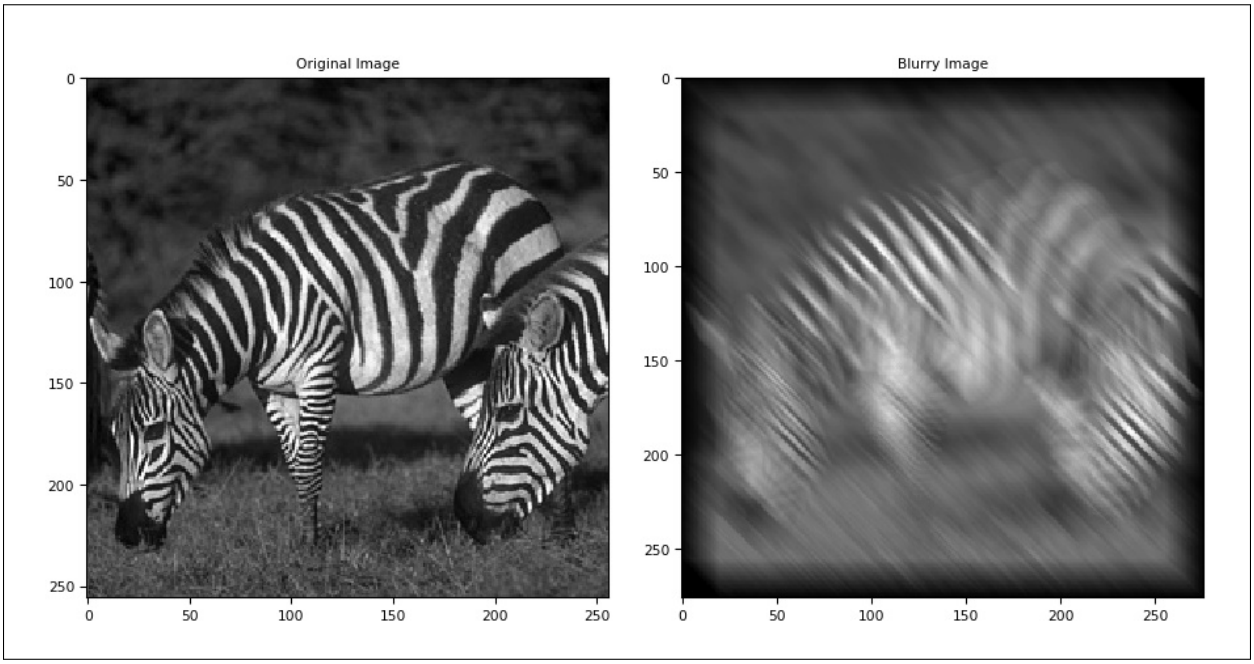
2 Image Preparation

To test the performance of different deconvolution algorithms, we need to generate a blurry image using a known blur kernel. Here are the procedures:

1. Pick an image from the BSDS500 dataset[2], and convert it to gray scale. Crop a 256×256 region from the selected image. Denote this image as $x(n_1, n_2)$.
2. Perform 2D convolution (with zero padding) on the cropped image using an identity matrix of size 21 as the kernel, $h(n_1, n_2)$. Remember to normalize the kernel to make sure that it sums up to one before convolution. Denote the obtained image as $y_{noiseless}(n_1, n_2)$.
3. Calculate the standard deviation of $x(n_1, n_2)$ and add Guassian noise with zero mean and standard deviation of $0.01 * std(x(n_1, n_2))$ to $y_{noiseless}(n_1, n_2)$. Denote the noisy image as $y_{noisy}(n_1, n_2)$.

2.1 Blurry Image (1.0 points)

Plot $x(n_1, n_2)$, $y_{noiseless}(n_1, n_2)$ and $y_{noisy}(n_1, n_2)$. What are the sizes of $y_{noiseless}(n_1, n_2)$ and $y_{noisy}(n_1, n_2)$?



[2] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

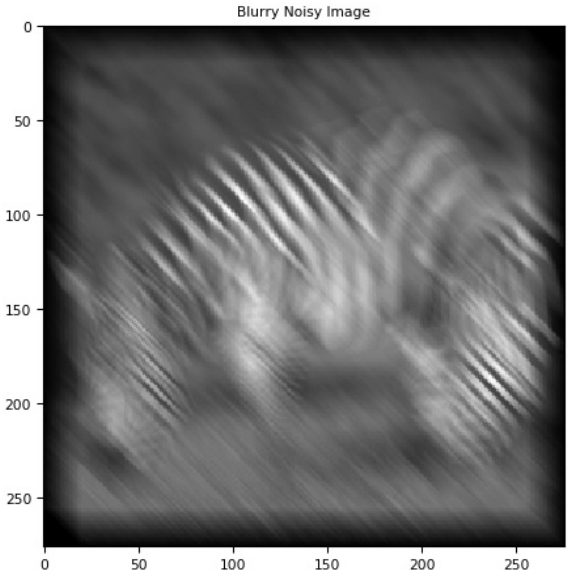


Figure 1: Images of $x(n_1,n_2)$, $y_{noiseless}(n_1,n_2)$ and $y_{noisy}(n_1,n_2)$.
The sizes of $y_{noiseless}(n_1,n_2)$ and $y_{noise}(n_1,n_2)$ are both 276×276, $\because 256+21-1=276$.

2.2 Metric Baseline (1.0 points)

Briefly describe the differences between PSNR metric and SSIM metric. Report PSNR(x, $y_{noiseless}$), SSIM(x, $y_{noiseless}$), PSNR(x, y_{noisy}), and SSIM(x, y_{noisy}). You can crop the center 256×256 regions from $y_{noiseless}(n_1,n_2)$ and $y_{noise}(n_1,n_2)$ when calculating PSNR and SSIM.

The difference between PSNR and SSIM is that, PSNR estimates absolute errors, on the other hand, SSIM is a perception-based model that considers image degradation as perceived change in structural information, while incorporating important perceptual phenomena, including both luminance masking and contrast masking terms[4].

In theoretical and experimental studies that compare PSNR and SSIM[1, 2], the studies reveal that PSNR is more sensitive to additive Gaussian noise than SSIM, while the opposite is observed for jpeg compression. Both measures have slightly similar sensitivity to Gaussian blur and jpeg2000 compression. And in all cases, PSNR and SSIM are more sensitive to additive Gaussian noise than Gaussian blur, jpeg and jpeg2000 compression.

$$
\begin{aligned}
PSNR(x, y_{noiseless}) &= 16.6019 \\
SSIM(x, y_{noiseless}) &= 0.4107 \\
PSNR(x, y_{noisy}) &= 16.6020 \\
SSIM(x, y_{noisy}) &= 0.4104
\end{aligned}
$$

3 Naive Deconvolution

3.1 Naive Deconvolution Algorithm (1.0 points)

Implement a function to conduct naive deconvolution, and provide your codes in the box below. The function should take the blurry observation, $y(n_1, n_2)$, and the blur kernel, $h(n_1, n_2)$, as the input parameters, and return the recovered image, $\hat{x}(n_1, n_2)$. The discrete Fourier transform functions in NumPy[3] might be useful.

```
def naive_deconv(y, h):
    Y = np.fft.fft2(y)
    H = np.fft.fft2(h, s = y.shape)

    epsilon = 1e-10
    X = Y / (H + epsilon)
    x = np.fft.ifft2(X)
    x = np.real(x[0:256, 0:256])

    x_max = np.max(x)
    x_min = np.min(x)
    x = (x - x_min) / (x_max - x_min)
    return x
```

3.2 Naive Deconvolution Results (1.0 points)

Apply your naive deconvolution algorithm to both $y_{noiseless}(n_1, n_2)$ and $y_{noisy}(n_1, n_2)$. Plot the recovered images, and report their PSNR and SSIM scores with $x(n_1, n_2)$. Remember to crop the boundaries of the recovered images.

[3] https://docs.scipy.org/doc/numpy/reference/routines.fft.html

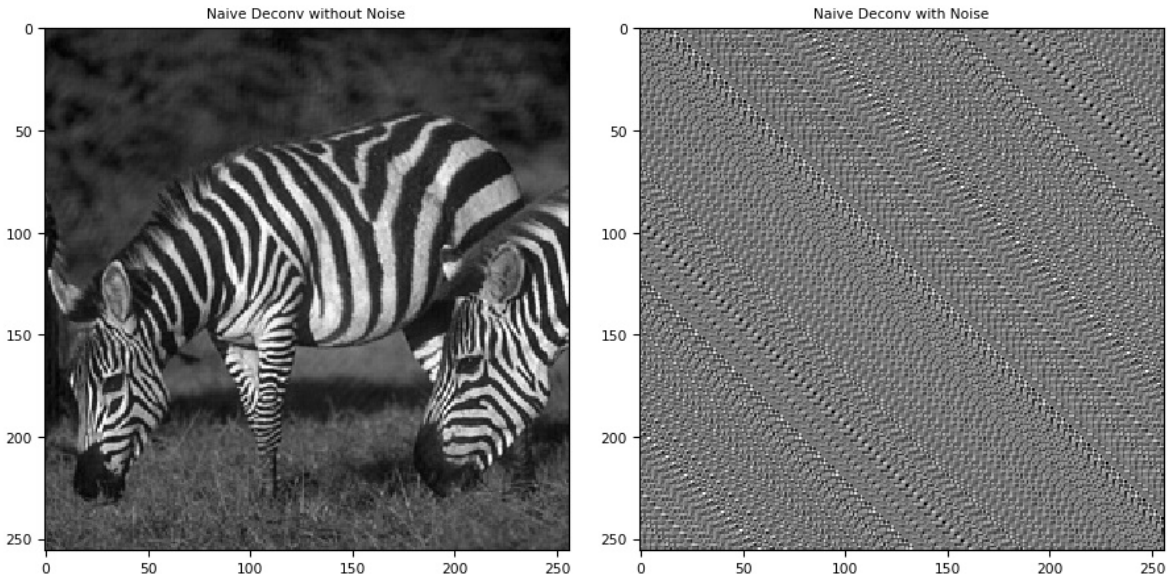


Figure 2: Images of $\hat{x}_{noiseless}(n_1,n_2)$ and $\hat{x}_{noisy}(n_1,n_2)$.

$$PSNR(x,\hat{x}_{noiseless}) = 23.2247$$
$$SSIM(x,\hat{x}_{noiseless}) = 0.9442$$
$$PSNR(x,\hat{x}_{noisy}) = 11.2365$$
$$SSIM(x,\hat{x}_{noisy}) = 0.0233$$

3.3 Naive Deconvolution Analysis (1.0 points)

Why the outputs of the above two cases are different? You need to derive the Fourier transform of the recovered images for this question.

For noisy images,

$$Y = X \times H + E,$$

where X, Y, H, E are the Fourier transform of the original image $x(n_1,n_2)$, noisy image $y(n_1,n_2)$, blur kernel $h(n_z,n_2)$, and the noise $e(n_1,n_2)$, respectively.

In Naive Deconvolution,

$$\hat{X} = \frac{Y}{H} = \frac{X \times H + E}{H} = X + \frac{E(\omega_1,\omega_2)}{H(\omega_1,\omega_2)},$$

where $\hat{X}(\omega_1,\omega_2) = \mathscr{F}[\hat{x}(n_1,n_2)]$ is the Fourier transform of the recovered image.

Only when the blur image is noiseless ($E = 0$), the recovered image is almost same as the

original image. However, when E exists, and $H(\omega_1, \omega_2)$ is close to 0 when in high frequencies, the noise will blow-up, so the recovered image looks like a mess.

In our case, the noise $e(n_1, n_2)$ is a Gaussian noise that is irrelevant to frequencies, and the blur kernel $h(n_1, n_2)$ is a normalized identity matrix, which functions as a low-pass filter. Therefore, the Naive Deconvolution for images with noise would blow up.

4 Wiener Filter

4.1 Wiener Filter Algorithm (1.0 points)

Express the recovered image from Wiener deconvolution in frequency domain, and implement your own Wiener filter function based on it. Place the code of your program in the box below.

```
# Element-wise square value of a matrix
def square(x):
    result = np.multiply(np.abs(x), np.abs(x))
    return result

def wiener_filter(y, h, snr):
    Y = np.fft.fft2(y)
    H = np.fft.fft2(h, s = y.shape)
    H2 = square(H)

    # Wiener filter
    epsilon = 1e-10
    G = H2/(H2 + 1/snr)
    X = Y/(H + epsilon) * G
    x = np.fft.ifft2(X)
    x = np.real(x[0:256, 0:256])

    x_min = np.min(x)
    x_max = np.max(x)
    x = (x - x_min)/(x_max - x_min)
    return x
```

4.2 Ideal Wiener Filter Results (1.0 points)

Apply your Wiener filter to $y_{noisy}(n_1, n_2)$. Plot your recovered image, and report its PSNR and SSIM scores. You can use the actual frequency-dependent $SNR(\omega_1, \omega_2)$ in this question.

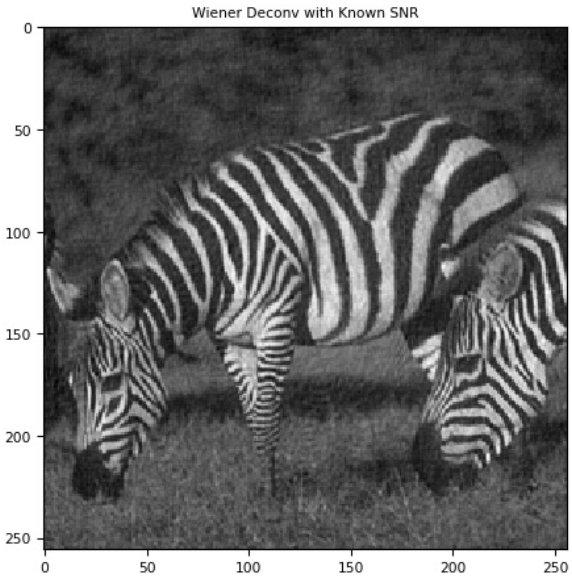


Figure 3: Image of $\hat{x}_{wiener\_exact}(n_1, n_2)$.

$$PSNR(x, \hat{x}_{wiener\_exact}) = 24.7730$$
$$SSIM(x, \hat{x}_{wiener\_exact}) = 0.8450$$

4.3 Power Spectral Density (1.0 points)

Normally, we do not have access to the frequency-dependent $SNR(\omega_1, \omega_2)$ in real applications. Therefore, people usually approximate the $SNR(\omega_1, \omega_2)$ from a predefined function. To explore how to estimate $SNR(\omega_1, \omega_2)$, let's first analyze the power spectrum of noise and real images. Plot the power spectral density of $x(n_1, n_2)$ and your added noise $e(n_1, n_2)$ in log scale[4]. Pick two other images with different scenes in the BSDS500 dataset, and plot these two images together with their log-scale spectral density.

[4]Remember to shift the zero-frequency component to the center of the spectrum by using "fftshift" in NumPy.

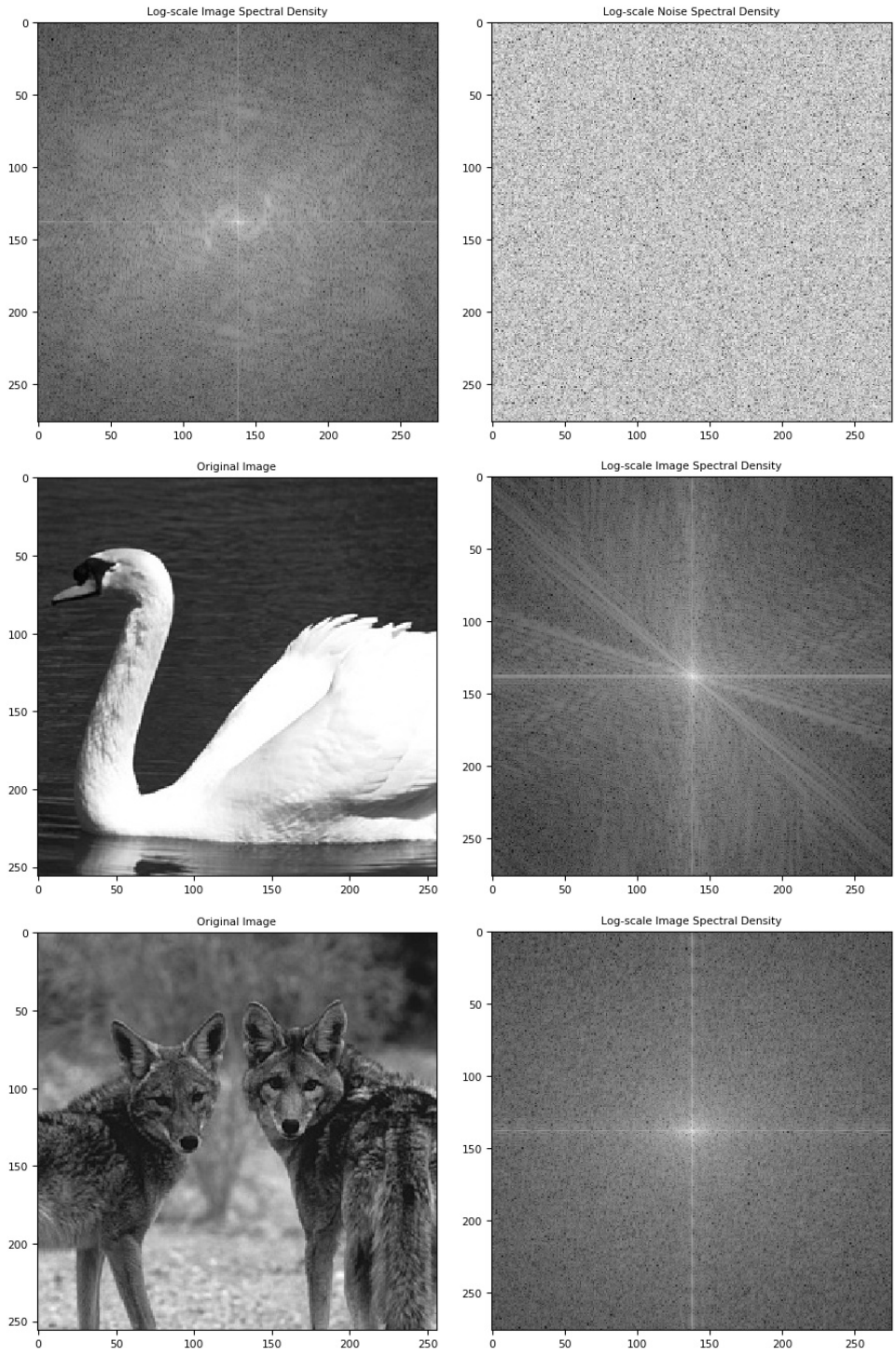


Figure 4: Images of log-scale power spectral density.

4.4 SNR Approximation (1.0 points)

Based on the above plots, describe the features of real images and noise. Which function would you use to approximate SNR in this case?

Based on the above plots, the power intensity of real images have the maximum value at the low frequency domain, while the power intensity of the Gaussian noise is irrelevant to the frequency.
Therefore, an approximation of SNR can be:

$$SNR(\omega_1, \omega_2) = \frac{1}{\omega_1^2 + \omega_2^2}.$$

A Log-scale diagram of this SNR function is shown in Figure 5.

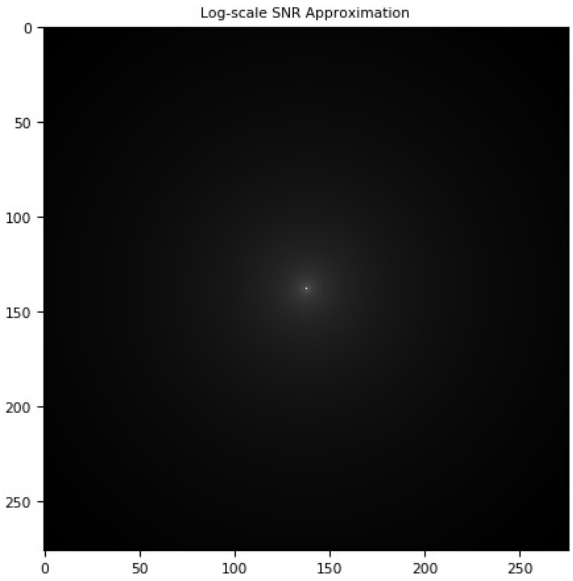


Figure 5: Image of log-scale SNR approximation.

(I also tried many other potential approximations and these extra results are attached at the end of the homework.)

4.5 Approximation Results (1.0 points)

Plot the deconvolution result using the above SNR approximation, and report your PSNR and SSIM scores.

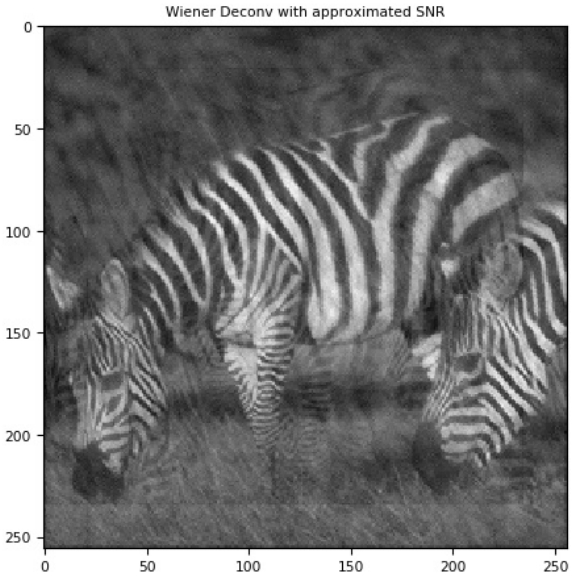


Figure 6: Image of $\hat{x}_{wiener\_approx}(n_1, n_2)$.

$$PSNR(x, \hat{x}_{wiener\_approx}) = 22.7635$$
$$SSIM(x, \hat{x}_{wiener\_approx}) = 0.7459$$

Extra Experiments

(1) The approximation of SNR function is:

$$SNR(\omega_1, \omega_2) = \omega_1^2 + \omega_2^2.$$

The Log-scale diagram of the SNR function and the recovered image are shown in Figure 7.

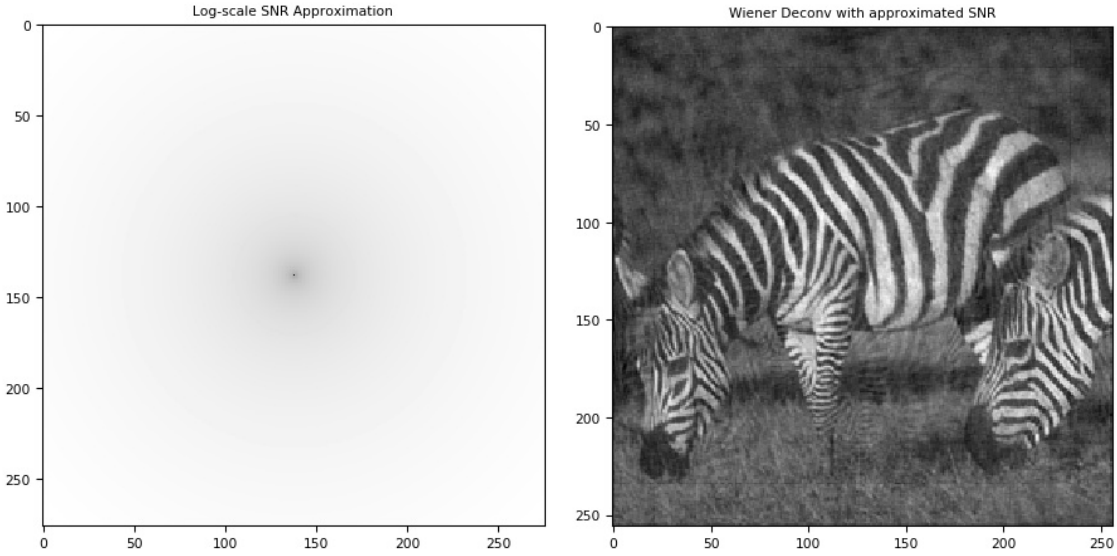


Figure 7: Image of log-scale SNR approximation and deconvolution result.

The performance evaluations are:

$$PSNR = 26.5565, SSIM = 0.7459.$$

(2) The approximation of SNR function is:

$$SNR(\omega_1, \omega_2) = max\left\{\frac{1}{\omega_1}, \frac{1}{\omega_2}\right\}.$$

The Log-scale diagram of the SNR function and the recovered image are shown in Figure 8.

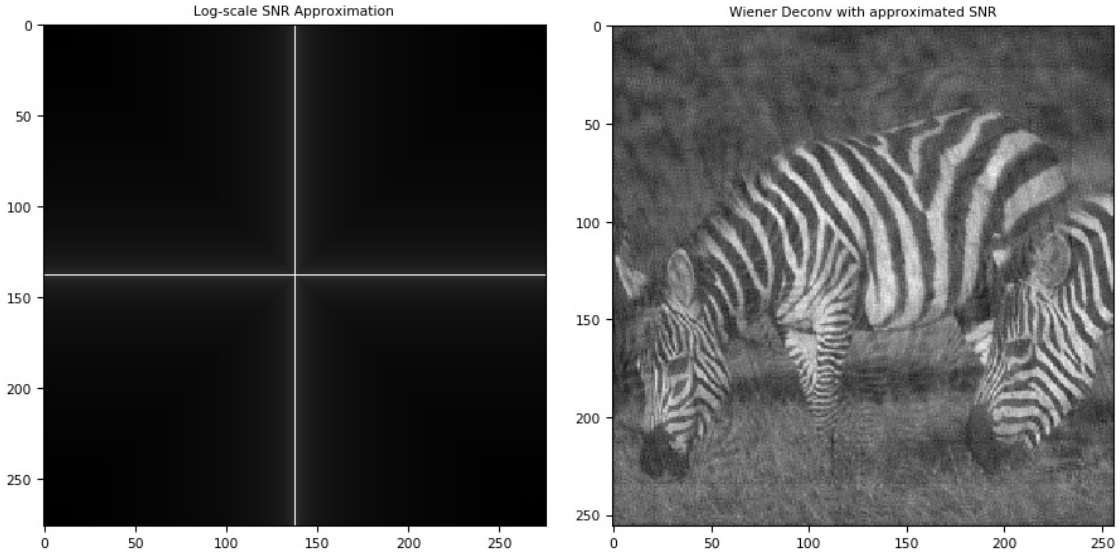


Figure 8: Image of log-scale SNR approximation and deconvolution result.

The performance evaluations are:

$$PSNR = 24.4619, SSIM = 0.7193.$$

(3) The approximation of SNR function is:

$$SNR(\omega_1, \omega_2) = \omega_1^{-0.6} + \omega_2^{-0.6}.$$

The parameter -0.6 comes from the simulation of the log-scale power spectral density at the $\omega_1 = 0, \omega_2$ varies or $\omega_2 = 0, \omega_1$ varies, as seen in Figure 9. The black and blue curve in the figure represent the log-scale psd value from the original sharp image and the noisy image, respectively.

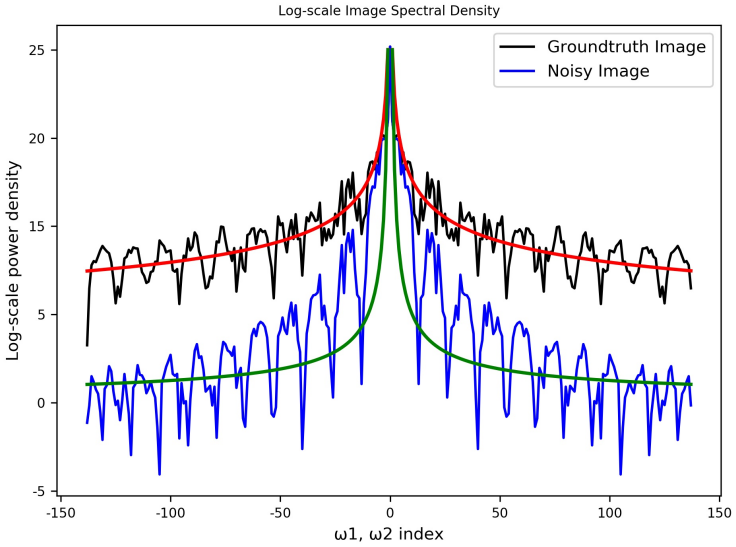


Figure 9: Simulation of log-scale power spectral density at low frequency, where the red curve follows the scale of $\omega^{-0.2}$, and the green curve follows the scale of $\omega^{-0.6}$.

From this new approximation function, the Log-scale diagram of the SNR function and the recovered image are shown in Figure 10.

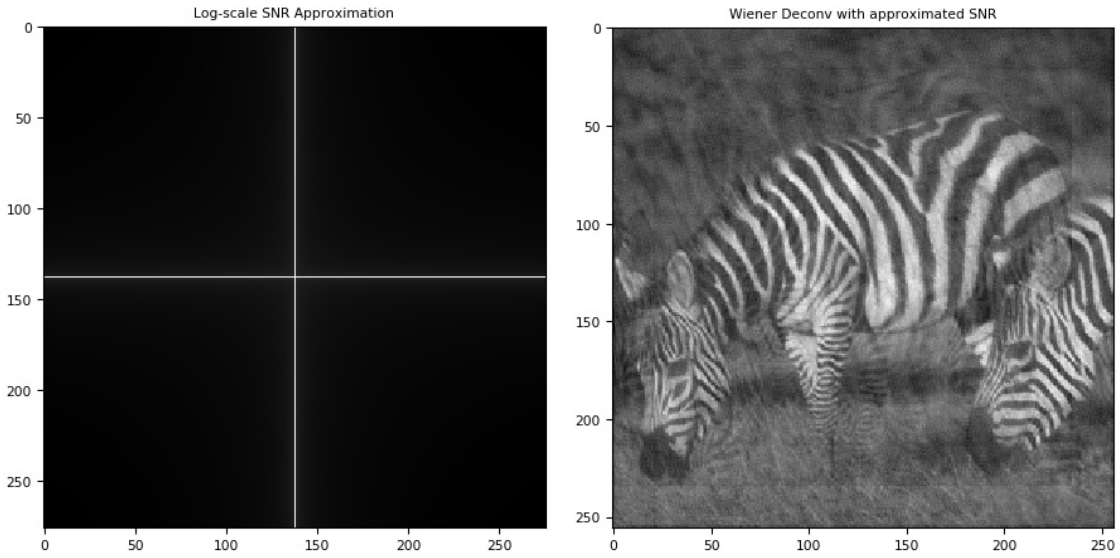


Figure 10: Image of log-scale SNR approximation and deconvolution result.

The performance evaluations are:

$$PSNR = 22.7547, SSIM = 0.7484.$$

(4) The approximation of SNR function is:

$$SNR(\omega_1, \omega_2) = \omega_1^{-0.6} * \omega_2^{-0.6}.$$

The parameter -0.6 comes from the same simulation as in Figure 9.
The Log-scale diagram of the SNR function and the recovered image are shown in Figure 11.

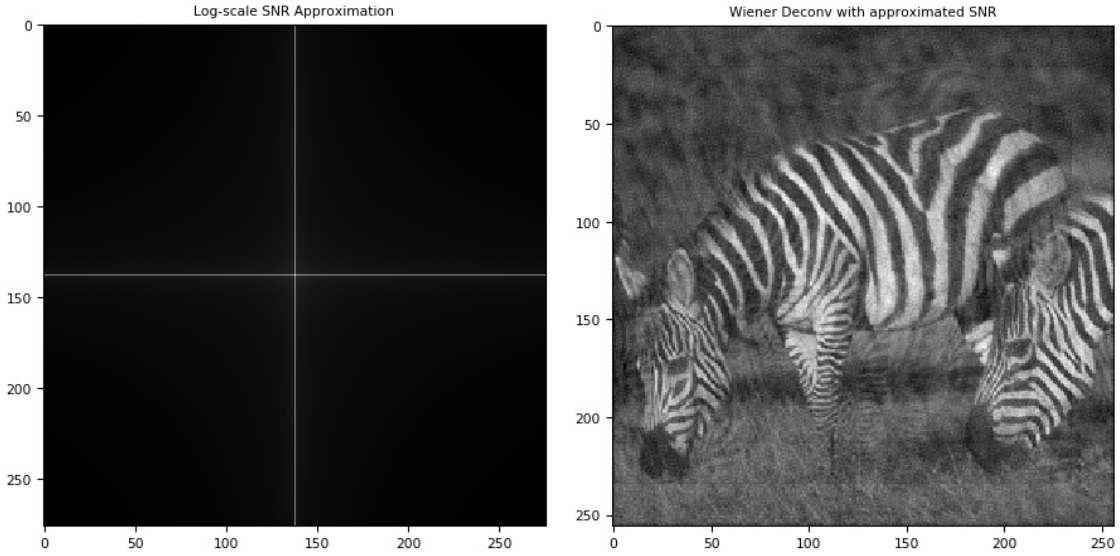


Figure 11: Image of log-scale SNR approximation and deconvolution result.

The performance evaluations are:

$$PSNR = 23.9514, SSIM = 0.7409.$$

(5) Summary and analysis:

Deconv Function	PSNR	SSIM
Baseline	16.6020	0.4104
Naive deconv	11.2365	0.0233
Wiener exact	24.7730	0.8450
Wiener: $1/(\omega_1^2 + \omega_2^2)$	22.7635	0.7459
Wiener: $\omega_1^2 + \omega_2^2$	26.5565	**0.8084**
Wiener: $\max\{1/\omega_1, 1/\omega_2\}$	24.4619	0.7193
Wiener: $\omega_1^{-0.6} + \omega_2^{-0.6}$	22.7547	0.7484
Wiener: $\omega_1^{-0.6} * \omega_2^{-0.6}$	23.9514	0.7409

a) The performance of different SNR approximation functions for Wiener Deconvolution is constrained in the range of performance results from "baseline" and "Wiener exact", so the SSIM will be no more than 0.8450 in this experiment, and should be no less than 0.4104.

b) It is strange to find the function $\omega_1^2 + \omega_2^2$ and $1/(\omega_1^2 + \omega_2^2)$ are both potentially applicable approximations, even though their trends of increasing and decreasing versus frequency are opposite. A possible reason is that $\omega_1^2 + \omega_2^2$ functions as a low-pass filter and $1/(\omega_1^2 + \omega_2^2)$

function as a high-pass filter, both can be used to extract and recover the image, and the performance estimation methods PSNR and SSIM cannot tell the difference whether the similarity with the original image comes from high frequency or low frequency. Therefore, it is possible that the function $\omega_1^2 + \omega_2^2$ returns a better solution.

c) The reason why the estimation function $\max\{1/\omega_1 + 1/\omega_2\}$ is experimented is that: the first two experiments are circular symmetric functions with regard to ω_1, ω_2, and Figure 4 show some square symmetry so this estimation might reveal some features of the original image. However, the results turn out that this approximation function of SNR have the least SSIM, which also occurs to experiments on other different images. Therefore, it might not be a good approximation.

d) The last two approximation functions in the table are inspired from the simulation as seen in Figure 9, where $\text{Log}\{\text{SNR}(\omega_1, 0)\} \propto \omega_1^{-0.6}$ and $\text{Log}\{\text{SNR}(0, \omega_2)\} \propto \omega_2^{-0.6}$. This simulation function is more elaborate than linear fitting as in the estimation that $\text{SNR}(\omega) \propto 1/\omega^2$, so hopefully the Wiener Deconvolution from such SNR approximation can be more accurate. One of these two functions is tested on addition and the other is using multiplication, the purpose of which is to look for an expression that can properly decouple the two frequencies. The experiment result is fairly ok, but is not as stable or extendable as the first two functions when tested on different images.

References

[1] Alireza Nasiri Avanaki. Exact global histogram specification optimized for structural similarity. *Optical Review*, 16(6):613–621, Nov 2009.

[2] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369. IEEE, 2010.

[3] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[4] Wikipedia contributors. Structural similarity — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Structural_similarity&oldid=940605506`, 2020. [Online; accessed 17-February-2020].