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

INSTRUCTOR: Prof. Achuta Kadambi
TA: Yunhao Ba

NAME: Aidan Cookson
UID: 605 284 665

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:

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

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 [1]. Remember to normalize the images before reporting the PSNR and SSIM scores.

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

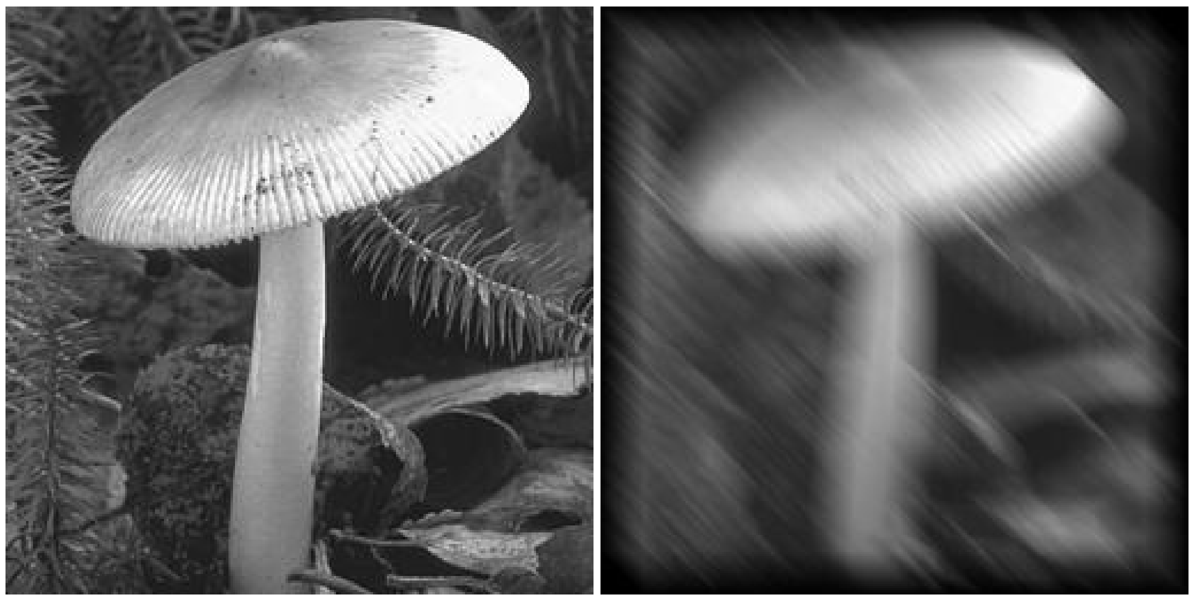
2 Image Preparation

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

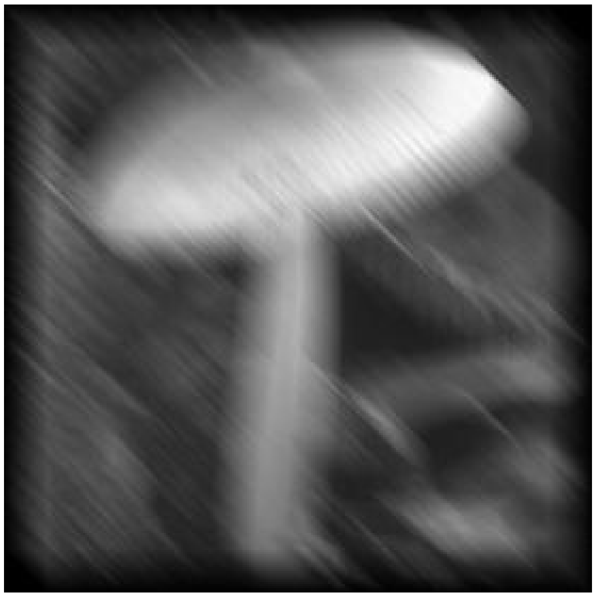


Figure 1: Images of $x(n_1,n_2)$, $y_{noiseless}(n_1,n_2)$ and $y_{noisy}(n_1,n_2)$.

Image Sizes:
$x(n_1,n_2)$: 256 x 256
$y_{noiseless}(n_1,n_2)$: 276 x 276
$y_{noisy}(n_1,n_2)$: 276 x 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.

PSNR and SSIM are both image similarity tests typically used in determining how much quality is lost due to an image process. While PSNR quantitatively compares image similarity using a scaled mean-squared-error, it does not do a good job in modeling what humans see as similar.
Instead of using absolute errors, SSIM uses a perception-based model that uses metrics such as luminance and contrast, as well as the structure of the image (the concept that pixels have stronger dependencies when they are spatially close together) to measure perceived change in structural information.
$PSNR(x, y_{noiseless}) = 68.2185$
$SSIM(x, y_{noiseless}) = .4322$
$PSNR(x, y_{noisy}) = 68.2148$
$SSIM(x, y_{noisy}) = .4310$

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.

```
% Prepare Ground Truth Image as 'x'
img_raw = imread('208001.jpg'); % Load Raw Image from Dataset
rect = [0 145 256 255];
img_cropped = imcrop(img_raw, rect); % Crop Image to 256 x 256 x 3
img_grey = rgb2gray(img_cropped); % Convert Image to GreyScale
x = im2double(img_grey);

% 2D Convolution on x using Normalized Identity Kernel 'h'
kernel_size = 21;
h = eye(21) ./ kernel_size; % Normalized Identity Matrix
y_noiseless = conv2(x, h); % Convolve input with kernel (x ** h), zero-padding edges (s

% Add Gaussian Noise to Noiseless y
var = (.01 * std2(x))^2; % Desired Gaussian Variance
y_noisy = imnoise(y_noiseless, 'gaussian', 0, var);

% Naive Deconvolution of Noiseless and Noisy Y
H = fft2(h, 276, 276) + 1e-12;
Y_noisy = fft2(y_noisy);
Y_noiseless = fft2(y_noiseless);

rect_crop = [0 0 256 256];
x_hat_noisy_naive = imcrop(ifft2(Y_noisy ./ H), rect_crop); % F-1(Y / H)
x_hat_noiseless_naive = imcrop(ifft2(Y_noiseless ./ H), rect_crop);
```

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

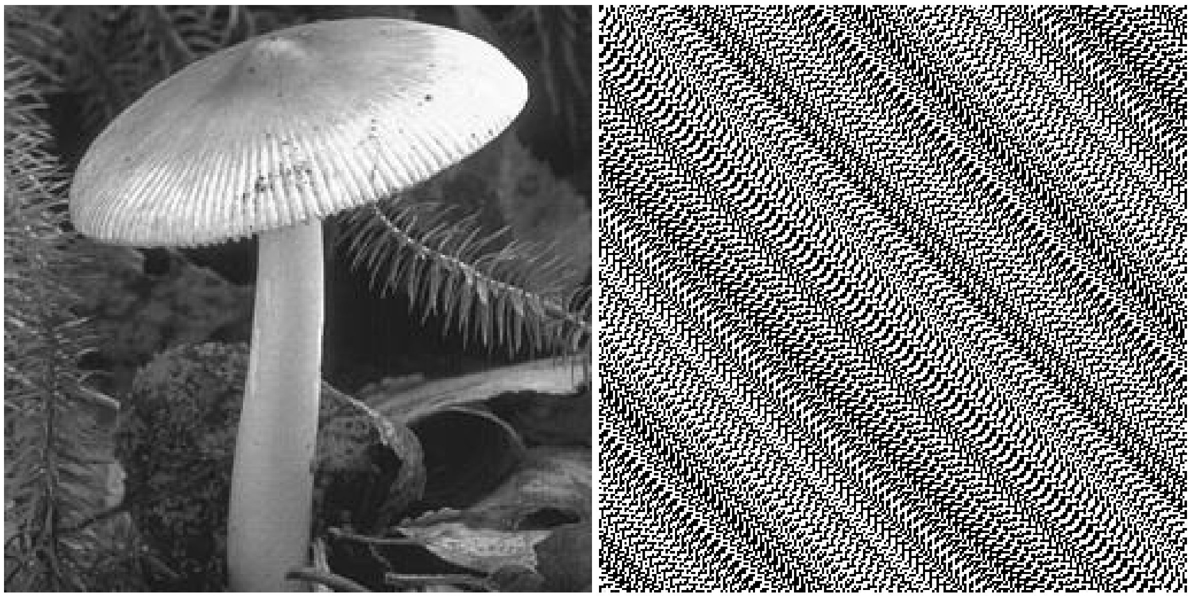


Figure 2: $\hat{x}_{noiseless}(n_1, n_2)$ and $\hat{x}_{noisy}(n_1, n_2)$

$PSNR(x,\ \hat{x}_{noiseless}) = 95.6986$
$SSIM(x,\ \hat{x}_{noiseless}) = .9945$
$PSNR(x,\ \hat{x}_{noisy}) = -118.1818$
$SSIM(x,\ \hat{x}_{noisy}) = 2.1212e-18$

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.

In the noiseless case we have : $y = h * x$
The result is $\hat{x} = F^{-1}(\frac{F(y)}{F(h)})$
In the noisy case we have: $y = h * x + \varepsilon$
The result is $\hat{x} = F^{-1}(\frac{F(y)}{F(h)} - \frac{F(\varepsilon)}{F(h)})$
The two results are very different because of the noise term. As you can see, The fourier transform of the noise divided by the fourier transform of the kernel is subtracted from the noiseless x hat. The fourier transform of the kernel at high frequencies is nearly zero. Since most of the noise is high frequencies, this term blows up, overpowering the rest of the signal.

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.

```
# Wiener Deconvolution of Noiseless and Noisy Y
X = fft2(x, 276, 276);
psd_x = X .* conj(X);
psd_h = H .* conj(H);
x_hat_noisy_wiener = ifft2((psd_h ./ (psd_h + 1 ./ psd_x)) .*
        (Y_noisy ./ H));
x_hat_noisy_wiener = imcrop(x_hat_noisy_wiener, rect_crop);
```

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.

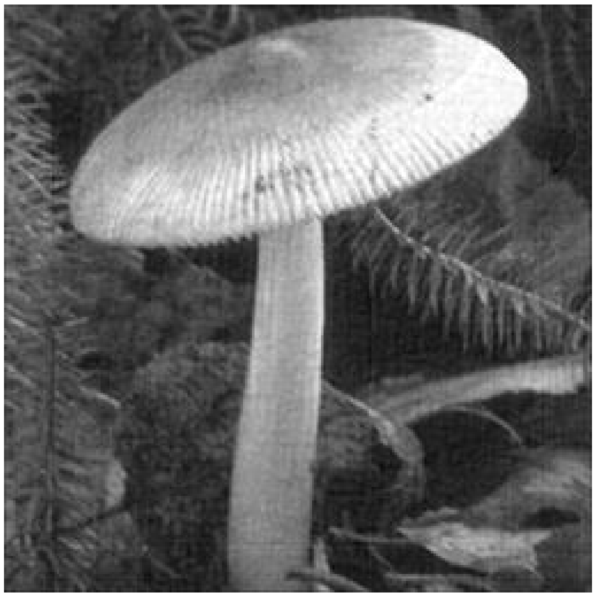
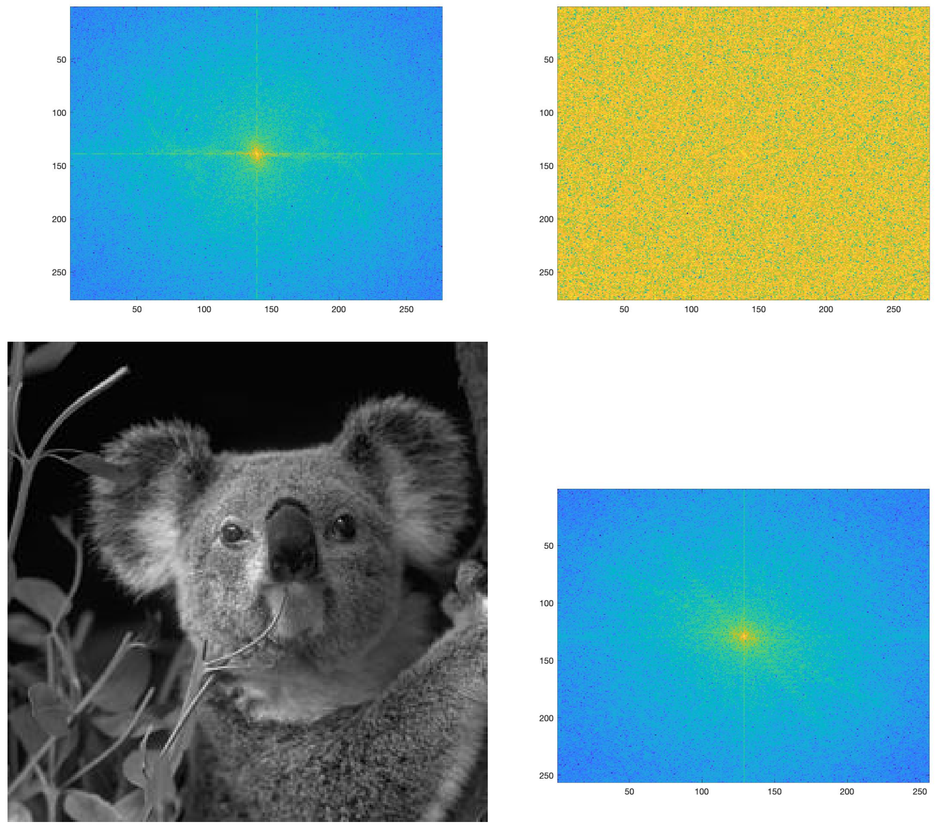


Figure 3: $\hat{x}_{wiener\_exact}(n_1,n_2)$

$SSIM(x,\ \hat{x}_{wiener\_exact}) = .8161$
$PSNR(x,\ \hat{x}_{wiener\_exact}) = 77.4076$

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.

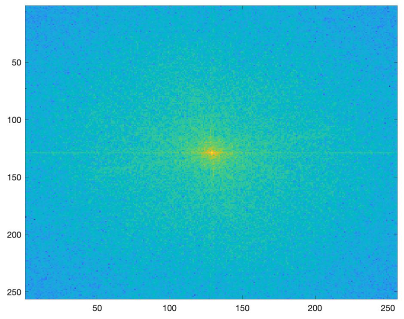


Figure 4: Log-Scale Spectral Density of x, added noise, and 2 additional natural images.

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?

Natural image noise appears to be constant with frequency (white noise). The power spectral density is also predictable in the frequency spectrum, resembling

$$1/x^2$$

This means the Singal-to-Noise ratio can be approximated as

$$\frac{Signal}{Noise} = \frac{\frac{1}{x^2}}{1} = \frac{1}{x^2}$$

4.5 Approximation Results (1.0 points)

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

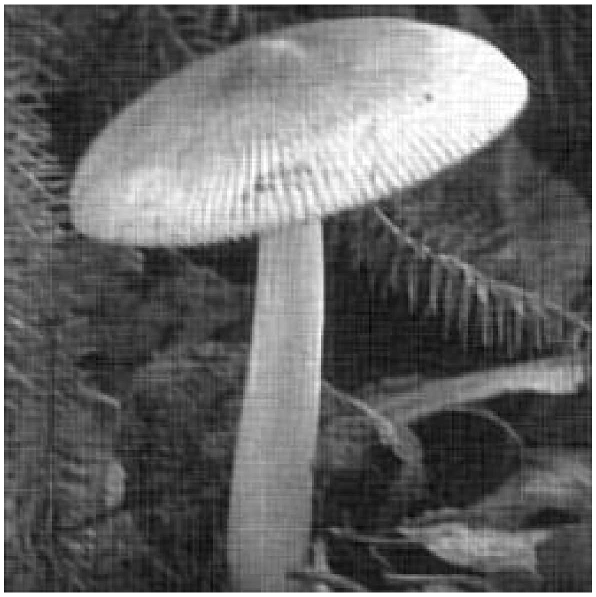


Figure 5: $\hat{x}_{wiener\_approx}(n_1, n_2)$

$SSIM(x,\, \hat{x}_{wiener\_est}) = .7511$
$PSNR(x,\, \hat{x}_{wiener\_est}) = 75.9852$

References

[1] 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.