



MSc in Digital Humanities
Image Processing/Analysis

Image Processing/Analysis

Anestis Koutsoudis, Chairi Kiourt, Christos Tsinos, George Pavlidis

Image Restoration

Christos Tsinos

e-mail: ctsinos@uoa.gr



Overview

- Introduction
- Types of Image Restoration
- Image Restoration and Image Enhancement
- Noise Reduction
- Deblurring
- Inpainting



Overview

- Examples

Introduction

- **Image restoration** is the process of recovering an original image from a degraded version by reducing noise, correcting distortions, and reconstructing missing information
- **Purpose:**
 - To enhance the visual quality of images for human interpretation
 - To improve image data for automated processing in computer vision applications

Introduction - Applications

- **Medical Imaging:** Enhancing MRI, CT, and X-ray scans to assist in diagnosis
- **Satellite Imagery:** Removing noise and distortions in remote sensing images
- **Historical Photo Restoration:** Recovering details in aged, damaged photographs
- **Surveillance:** Enhancing low-quality security camera footage for better recognition

Introduction - Goals

- Minimize the effect of noise and artifacts
- Restore images based on a known or estimated degradation model
- Improve image clarity and usability for further analysis

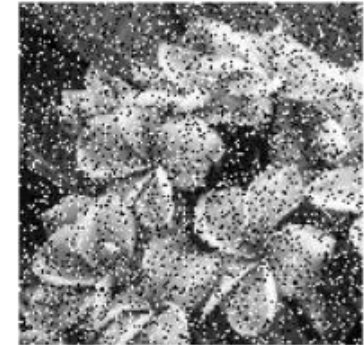
Types of Image Degradation

- **Noise:** Random variations in pixel intensity, often caused by sensor limitations or transmission errors
- **Types:** Gaussian noise (additive white noise), Salt & Pepper noise (sparse, high-contrast artifacts), Speckle noise, Poisson noise (quantum noise in low-light conditions)

Original image



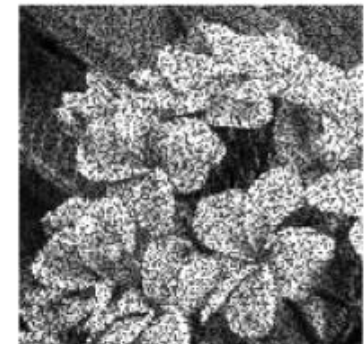
Salt and Pepper noise



Gaussian noise



Speckle noise



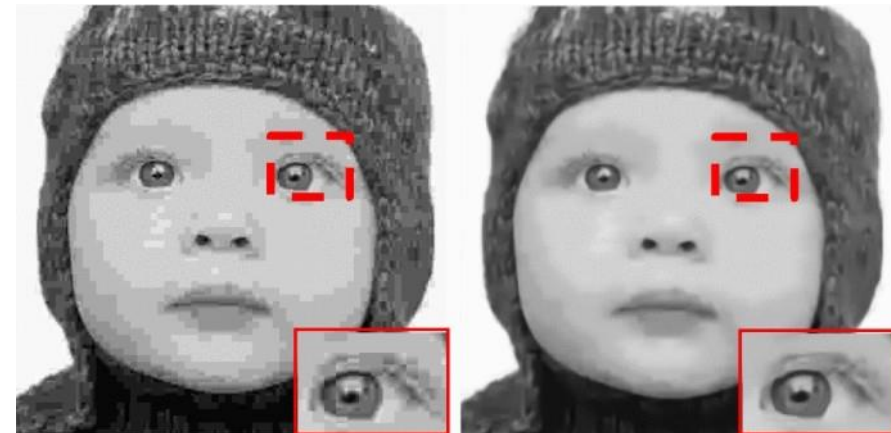
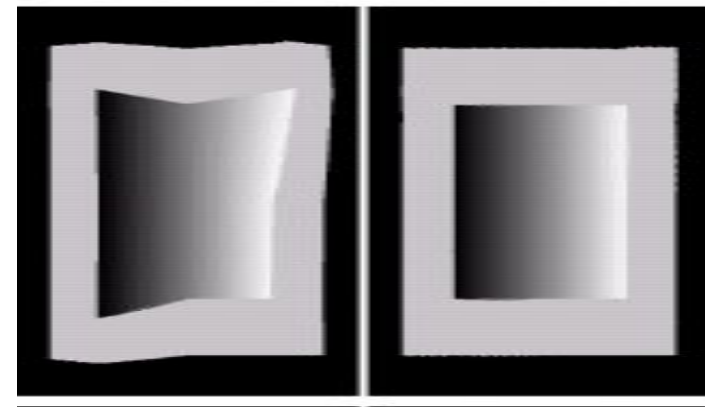
Types of Image Degradation

- **Blur:** Loss of sharpness due to motion, defocus, or optical imperfections.
- Motion blur: Caused by camera movement or object motion during exposure
- Defocus blur: Resulting from incorrect focal settings



Types of Image Degradation

- **Distortions:** Unwanted geometric alterations in the image structure
- **Geometric distortions:** Perspective or radial distortions from camera lenses
- **Compression artifacts:** Loss of quality due to aggressive compression algorithms (e.g., JPEG blocking and ringing effects)



Types of Image Degradation

- **Missing Data:** Portions of an image that are lost or occluded
- **Causes:** Physical damage (scratches, tears), occlusions, or sensor malfunctions

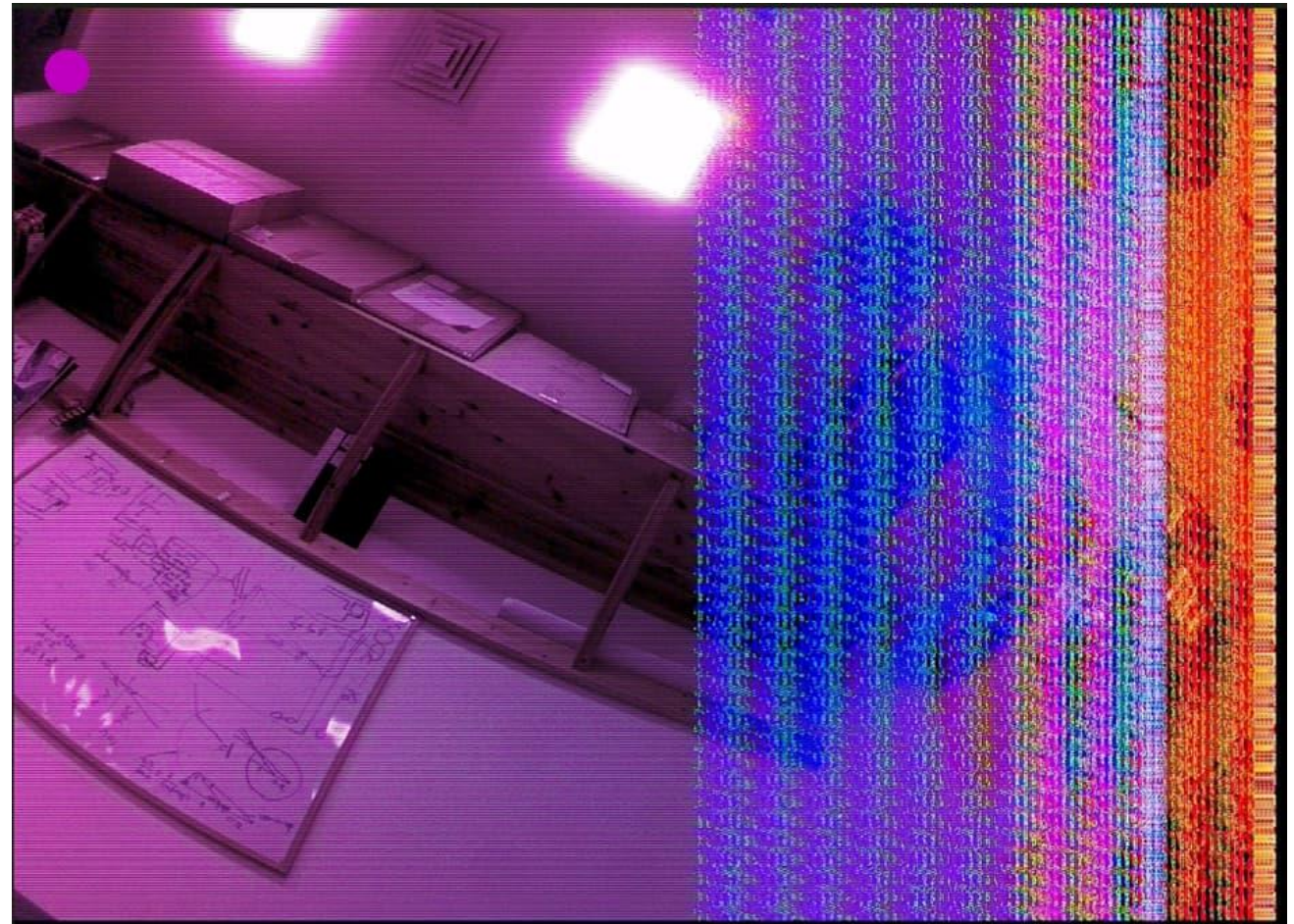


Image Restoration or Image Enhancement

- **Image Restoration:** Recovering original image using a known degradation model
- **Image Enhancement:** Improving image quality without a specific degradation model
- Example: Denoising vs. Contrast Adjustment

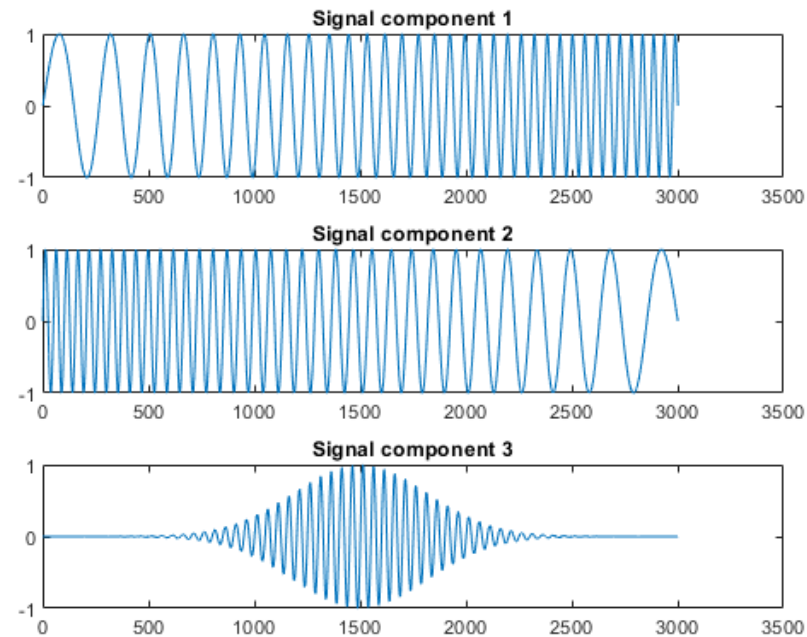
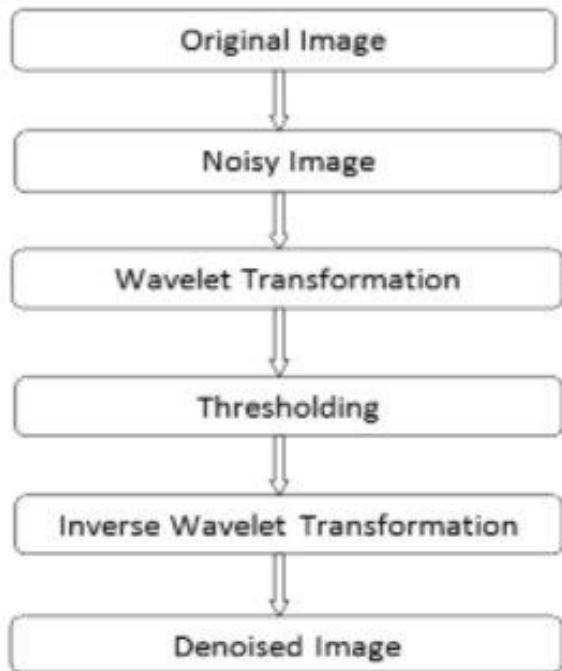
Noise Reduction

- **Spatial Domain Methods**
- **Mean Filtering:** Replaces each pixel value with the average of neighboring pixels, reducing random noise but potentially blurring edges
- **Median Filtering:** Uses the median value of neighboring pixels instead of the mean, making it effective in removing salt-and-pepper noise while preserving edges
- **Gaussian Filtering:** Applies a Gaussian-weighted average to smooth out noise, commonly used for softening images

Noise Reduction

- **Frequency Domain Methods:**
- **Fourier Transform-based Filtering:** Converts the image into the frequency domain, allowing selective removal of high-frequency noise while preserving low-frequency details
- **Wavelet-based Denoising:** Decomposes the image into multiple frequency components using wavelet transforms, allowing targeted noise reduction without significant loss of detail

Noise Reduction



Noise Reduction

- **Advanced Techniques**
- **Non-Local Means (NLM)** filtering is a powerful image denoising technique that leverages the redundancy of information within an image to reduce noise while preserving fine details
- Unlike traditional denoising methods that rely on local pixel neighbourhoods, NLM compares patches across the entire image to find similar structures, making it particularly effective for preserving textures and edges

Noise Reduction

- The similarity between patches is measured using a weighted Euclidean distance
- Patches that are more similar to the target patch contribute more to the denoised value
- The denoised value of a pixel is computed as a weighted average of all pixels in the image, where the weights depend on the similarity of their surrounding patches to the target patch
- The weights are typically computed using a Gaussian kernel applied to the patch distances

Noise Reduction

- **Parameters:**
- **Search Window:** The size of the region in which similar patches are searched
- **Patch Size:** The size of the patches used for comparison
- **Smoothing Parameter (h):** Controls the decay of the weights as a function of patch dissimilarity. A larger h results in smoother images but may lose details.

Noise Reduction

For a pixel i , the denoised value $\text{NLM}(i)$ is computed as:

$$\text{NLM}(i) = \frac{1}{Z(i)} \sum_{j \in \Omega} w(i, j) \cdot v(j)$$

where:

- $v(j)$ is the intensity of pixel j .
- $w(i, j)$ is the weight assigned to pixel j based on the similarity between patches centered at i and j .
- $Z(i)$ is a normalization factor: $Z(i) = \sum_{j \in \Omega} w(i, j)$.
- Ω is the search window.

Noise Reduction

The weights $w(i, j)$ are computed as:

$$w(i, j) = \exp \left(-\frac{\|P(i) - P(j)\|_2^2}{h^2} \right)$$

where:

- $P(i)$ and $P(j)$ are patches centered at pixels i and j , respectively.
- $\|P(i) - P(j)\|_2^2$ is the squared Euclidean distance between the patches.
- h is the smoothing parameter.

Noise Reduction

- **Edge Preservation:** NLM effectively preserves edges and fine details because it uses global information
- **Texture Preservation:** The method is particularly good at maintaining textures, which are often lost in local filtering methods
- **Adaptability:** The weights adapt to the image content, making NLM suitable for a wide range of noise levels and image types

Noise Reduction

- **Computational Complexity:** The algorithm is computationally expensive because it requires comparing patches across the entire image
- **Parameter Sensitivity:** The performance of NLM depends on the choice of parameters (patch size, search window, and h), which may require tuning for optimal results
- **Memory Usage:** Storing and comparing patches for the entire image can be memory-intensive

Noise Reduction



Original image



Noisy image



Denoised image

Noise Reduction

- **Total Variation (TV)** denoising is a popular technique for image denoising and restoration, particularly effective for preserving edges while reducing noise
- It is based on the principle of minimizing the total variation of the image, which is essentially the sum of the absolute gradients of the image intensity
- This approach favours piecewise-smooth images, making it ideal for images with sharp edges and flat regions.

Noise Reduction

- The total variation of an image I is defined as:

$$TV(I) = \sum_{i,j} |\nabla I(i, j)|$$

where $\nabla I(i, j)$ is the gradient of the image at pixel (i, j) .

- For a 2D image, this can be approximated as:

$$TV(I) = \sum_{i,j} \sqrt{(I_x(i, j))^2 + (I_y(i, j))^2}$$

where I_x and I_y are the horizontal and vertical derivatives, respectively.

Noise Reduction

- TV denoising aims to minimize an objective function that balances fidelity to the noisy image and smoothness:

$$\min_I \left(\frac{1}{2} \|I - I_{\text{noisy}}\|_2^2 + \lambda \cdot TV(I) \right)$$

where:

- I_{noisy} is the noisy image.
- $\|I - I_{\text{noisy}}\|_2^2$ ensures the denoised image is close to the noisy image.
- λ is a regularization parameter controlling the trade-off between noise reduction and edge preservation.

Noise Reduction

- **Edge Preservation:** Maintains sharp edges while reducing noise
- **Piecewise-Smooth Results:** Produces images with flat regions separated by sharp transitions
- **Effective for Low Noise Levels:** Works well for images with moderate to low noise

Noise Reduction

- **Staircasing Effect:** TV denoising can introduce artificial stair-like structures in smooth regions, especially for large λ
- **Loss of Texture:** Fine textures may be oversmoothed because TV favors piecewise-constant solutions
- **Computational Cost:** Solving the optimization problem can be computationally expensive for large images

Noise Reduction

Original



Noisy image



Denoised image



Noise Reduction

- Deep learning-based denoising has revolutionized image denoising by leveraging the power of neural networks to learn complex mappings from noisy images to clean images
- Unlike traditional methods, which rely on handcrafted priors or assumptions about the image and noise, deep learning models learn these priors directly from data, making them highly adaptive and effective for a wide range of noise types and levels

Noise Reduction

- **Data-Driven Approach:**
 - Deep learning models are trained on large datasets of noisy-clean image pairs
 - The model learns to predict the clean image from the noisy input
- **End-to-End Learning:** The entire denoising process is learned in a single framework, without the need for explicit noise modelling
- **Architectures:**
 - Convolutional Neural Networks (CNNs) are the backbone of most deep learning-based denoising methods
 - Advanced architectures like U-Nets, ResNets, and Transformers are commonly used

Noise Reduction

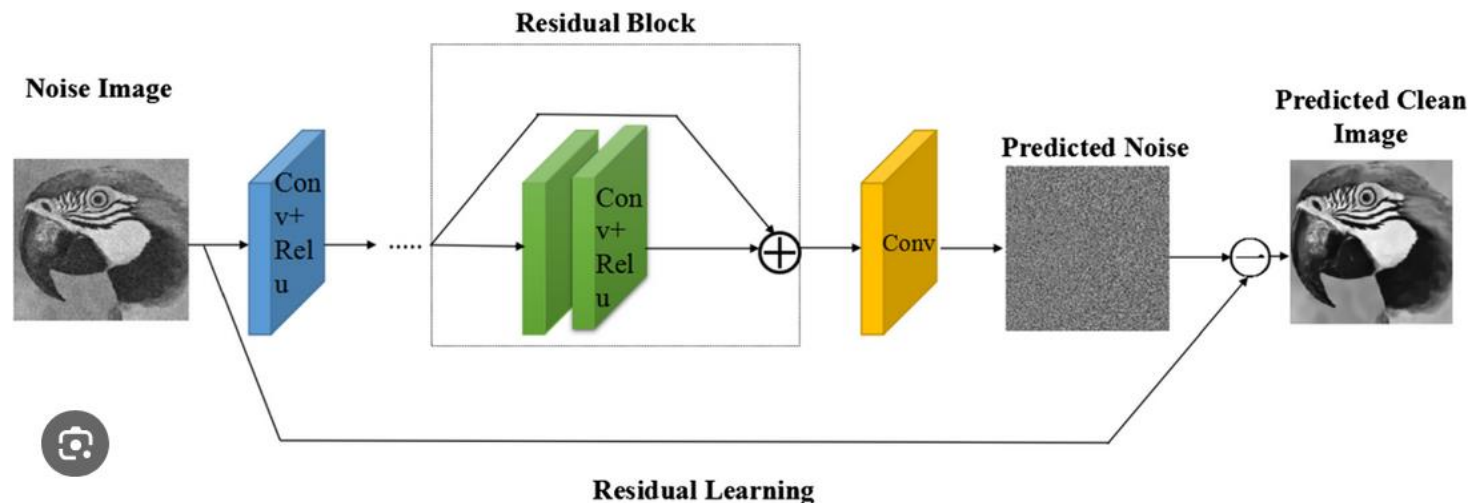
- **Loss Functions:** Common loss functions include Mean Squared Error (MSE), Mean Absolute Error (MAE), and perceptual losses (e.g., based on VGG features)
- **Types of Noise:** Deep learning models can handle various types of noise, including Gaussian, Poisson, salt-and-pepper, and real-world noise
- Training deep learning models requires large datasets of noisy-clean image pairs, which can be difficult to obtain

Noise Reduction

- Training and inference can be computationally expensive, especially for high-resolution images
- Models trained on specific noise types or datasets may not generalize well to unseen data
- Deep learning models are often considered "black boxes," making it difficult to understand their decision-making process

Noise Reduction

- Popular Architectures
- **DnCNN (Denoising Convolutional Neural Network)**
 - A simple yet effective CNN architecture for Gaussian denoising
 - Uses residual learning to predict the noise rather than the clean image

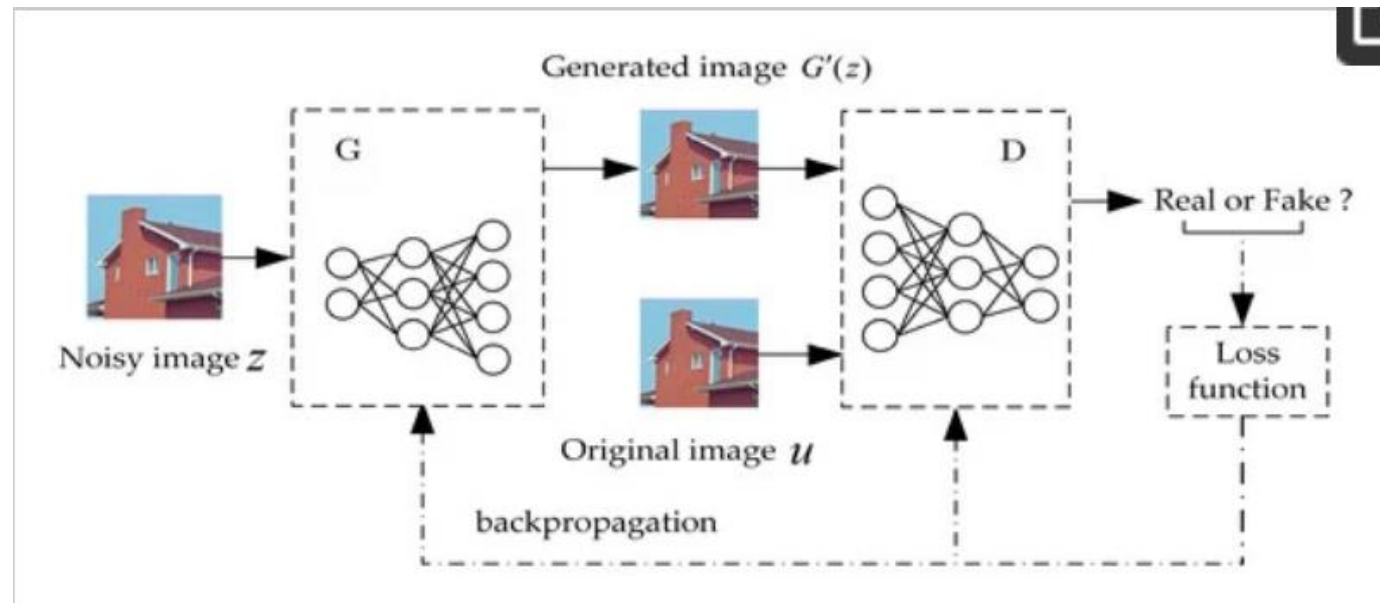


Noise Reduction

- **U-Net**
 - A popular encoder-decoder architecture with skip connections, widely used for image-to-image tasks
 - Effective for preserving fine details
- **ResNet (Residual Networks)**
 - Uses residual blocks to facilitate training of very deep networks
 - Commonly used in denoising tasks

Noise Reduction

- **GANs (Generative Adversarial Networks):**
 - Uses a generator to denoise images and a discriminator to distinguish between denoised and clean images
 - Effective for generating visually pleasing results



Deblurring

- The **Wiener Filter** is a classical linear filtering technique used for image restoration, particularly for deblurring and denoising
- It is optimal in the sense of minimizing the mean squared error (MSE) between the original (clean) image and the restored image, assuming knowledge of the degradation process (e.g., blur kernel and noise statistics)
- It is widely used for correcting **linear motion blur** and other types of degradations

Deblurring

- The observed (blurred and noisy) image $g(x, y)$ is modeled as:

$$g(x, y) = h(x, y) * f(x, y) + n(x, y)$$

where:

- $f(x, y)$: Original (clean) image.
- $h(x, y)$: Point Spread Function (PSF) or blur kernel (e.g., motion blur kernel).
- $n(x, y)$: Additive noise (usually assumed to be Gaussian).
- $*$: Convolution operator.

The blur kernel $h(x, y)$ is known or can be estimated.

The noise is additive and follows a known statistical distribution (e.g., Gaussian).

Deblurring

- The Wiener Filter is applied in the frequency domain using the Fourier Transform.
- The restored image $\hat{F}(u, v)$ is computed as:

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + K} \right] G(u, v)$$

where:

- $H(u, v)$: Fourier Transform of the blur kernel $h(x, y)$.
- $G(u, v)$: Fourier Transform of the observed image $g(x, y)$.
- $H^*(u, v)$: Complex conjugate of $H(u, v)$.
- K : Noise-to-signal power ratio (often estimated or tuned).

Deblurring

- **Optimal MSE Performance:** Minimizes the mean squared error between the original and restored images
- **Efficient Computation:** Computationally efficient when applied in the frequency domain using the Fast Fourier Transform (FFT)
- **Effective for Linear Blur:** Works well for linear degradations like motion blur

Deblurring

- **Requires Knowledge of Blur Kernel:** The blur kernel $h(x,y)$ must be known or accurately estimated
- **Sensitive to Noise:** Performance degrades in the presence of high noise levels
- **Assumes Stationarity:** Assumes the blur and noise are spatially invariant, which may not hold in real-world scenarios

Deblurring

- Steps for applying Wiener Filtering

1. **Estimate the Blur Kernel:**

- The motion blur kernel can be estimated from the image or provided as prior knowledge.
- For example, a horizontal motion blur kernel might look like:

$$h(x, y) = \begin{cases} \frac{1}{L} & \text{if } x = 0, |y| \leq \frac{L-1}{2} \\ 0 & \text{otherwise} \end{cases}$$

where L is the length of the motion blur.

2. **Compute the Fourier Transforms:**

- Compute the Fourier Transform of the observed image $G(u, v)$ and the blur kernel $H(u, v)$.

3. **Apply the Wiener Filter:**

- Use the Wiener Filter formula to compute the restored image in the frequency domain.

4. **Inverse Fourier Transform:**

- Convert the restored image back to the spatial domain.

Blurred Image



Restored Blurred Image



Deblurring

- **Blind Deconvolution** is a powerful image restoration technique used to recover a sharp image from a blurred and noisy observation when the blur kernel (Point Spread Function, or PSF) is unknown
- Unlike non-blind deconvolution methods (e.g., Wiener Filter), which require prior knowledge of the blur kernel, blind deconvolution estimates both the sharp image and the blur kernel simultaneously
- This makes it highly useful in real-world scenarios where the blur kernel is difficult to measure or unknown

Deblurring

- The observed (blurred and noisy) image $g(x, y)$ is modeled as:

$$g(x, y) = h(x, y) * f(x, y) + n(x, y)$$

where:

- $f(x, y)$: Original (sharp) image.
- $h(x, y)$: Unknown blur kernel (PSF).
- $n(x, y)$: Additive noise.
- $*$: Convolution operator.

Deblurring

- Estimate both the sharp image $f(x,y)$ and the blur kernel $h(x,y)$ from the observed image $g(x,y)$
- The problem is ill-posed because there are infinitely many combinations of $f(x,y)$ and $h(x,y)$ that can produce the same $g(x,y)$
- Regularization or prior knowledge about the image and kernel is required to constrain the solution

Deblurring

- **Maximum A Posteriori (MAP) Estimation:**
 - Formulates the problem as a probabilistic model and maximizes the posterior probability of the image and kernel given the observed data
 - Uses priors for both the image and kernel to regularize the solution
- **Variational Methods:** Minimize an energy functional that includes data fidelity terms and regularization terms for the image and kernel
- **Iterative Methods:** Alternate between estimating the image and the kernel in an iterative manner (**Richardson-Lucy** algorithm for blind deconvolution)

Deblurring

- **Deep Learning-Based Methods:**
 - Use neural networks to learn the mapping from blurred images to sharp images and blur kernels
 - Can handle complex and unknown blur kernels
- **Regularization and Priors:** To make the problem well-posed, blind deconvolution relies on regularization and prior knowledge
- **Image Priors:**
 - Sparsity of gradients (e.g., Total Variation, ℓ_1 -norm of gradients).
 - Natural image statistics (e.g., heavy-tailed distributions of gradients).

Deblurring

- **Kernel Priors:**
 - Sparsity of the blur kernel (e.g., ℓ_1 -norm of the kernel).
 - Non-negativity and smoothness constraints
- Solving the joint optimization problem is computationally expensive
- Performance degrades in the presence of high noise levels
- The optimization problem is non-convex, leading to potential convergence to local minima

Deblurring

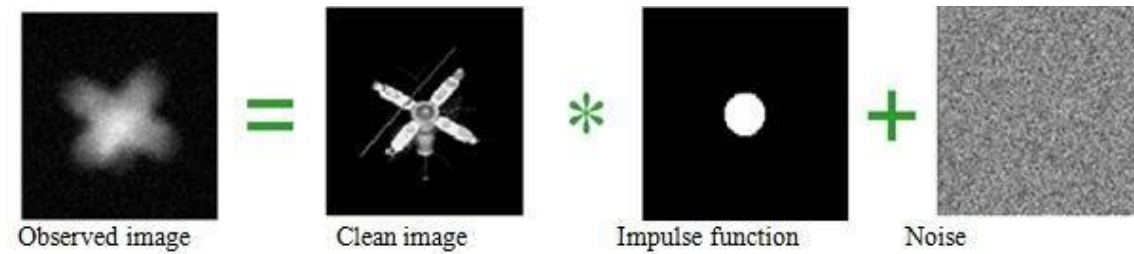


Fig. 2. General deconvolution procedure.

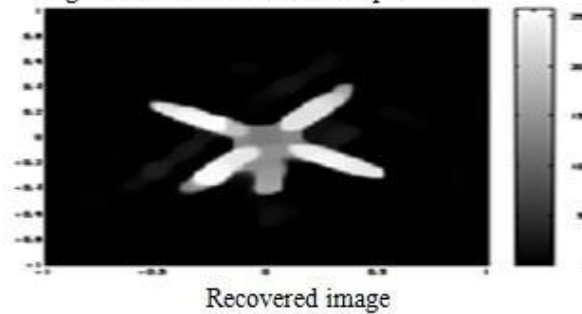
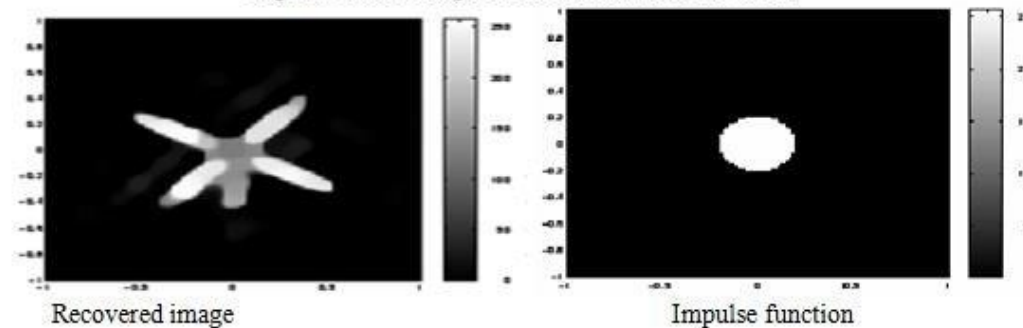


Fig. 3. General output in non-blind deconvolution



Inpainting

- **Texture Synthesis-Based Inpainting** is a technique used to fill in missing or damaged regions of an image by synthesizing new texture that is consistent with the surrounding areas
- Unlike simpler inpainting methods that rely on smooth interpolation (e.g., diffusion-based methods), texture synthesis-based methods aim to preserve the structural and textural details of the image, making them particularly effective for restoring complex textures and patterns

Inpainting

- **Inpainting Problem:** Given an image with a missing or damaged region (the "hole"), the goal is to fill the hole with content that is visually plausible and consistent with the surrounding areas
- Texture synthesis involves generating new texture patterns that match the statistical and perceptual properties of a given sample texture.
- In inpainting, the surrounding region of the hole is used as the sample texture
- The image is divided into small patches (e.g., 5x5 or 7x7 pixels)

Inpainting

- Missing regions are filled by copying and pasting patches from the known regions, ensuring consistency in texture and structure
- The synthesized texture must seamlessly blend with the surrounding regions
- Structural continuity (e.g., edges and lines) must be preserved

Inpainting

- **Advantages of Texture Synthesis-Based Inpainting**
- **Preservation of Texture:** Effectively restores complex textures and patterns
- **Structural Consistency:** Maintains edges and structures in the image
- **Flexibility:** Can handle a wide variety of textures and image types

Inpainting

- **Limitations of Texture Synthesis-Based Inpainting**
- **Computational Complexity:** Patch-based methods can be computationally expensive, especially for large images or holes
- **Dependence on Surrounding Texture:** Requires sufficient high-quality texture in the surrounding regions to synthesize plausible content
- **Artifacts:** May introduce repetition or inconsistencies if the patch selection and blending are not handled carefully

Inpainting

- **Methods**
- **Efros and Leung's Algorithm**
 - A non-parametric method that synthesizes texture by growing the image one pixel at a time, using the surrounding texture as a reference.
 - Relies on a nearest-neighbour search to find the best-matching pixel or patch
- **PatchMatch**
 - An efficient algorithm for finding approximate nearest-neighbour matches between patches
 - Widely used in inpainting and image editing due to its speed and effectiveness

Inpainting

- **Criminisi's Algorithm**

- A patch-based method that prioritizes filling patches with strong structural information (e.g., edges) first
- Combines texture synthesis with structure propagation

- **Deep Learning-Based Methods**

- Use neural networks to learn the mapping from damaged images to restored images
- Examples include **Context Encoders** and **Generative Adversarial Networks (GANs)**

Examples: Image Denoising

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.restoration import denoise_tv_chambolle, denoise_nl_means, estimate_sigma

def add_gaussian_noise(image, mean=0, sigma=25):
    noisy_image = image + np.random.normal(mean, sigma, image.shape)
    return np.clip(noisy_image, 0, 255).astype(np.uint8)

def compute_snr(original, denoised):
    signal_power = np.mean(original ** 2)
    noise_power = np.mean((original - denoised) ** 2)
    return 10 * np.log10(signal_power / noise_power)
```

Examples: Image Denoising

```
# Load grayscale image
```

```
image = cv2.imread('lenna.jpg', cv2.IMREAD_GRAYSCALE)
```

```
noisy_image = add_gaussian_noise(image)
```

```
# Apply Mean Filter
```

```
denoised_mean = cv2.blur(noisy_image, (10, 10))
```

```
# Apply Non-Local Means Filtering
```

```
sigma_est = np.mean(estimate_sigma(noisy_image))
```

```
denoised_nlm = denoise_nl_means(noisy_image.astype(np.float64) / 255.0, h=1.15 * sigma_est,  
fast_mode=True, patch_size=10, patch_distance=2, channel_axis=None)
```

```
denoised_nlm = (denoised_nlm * 255).astype(np.uint8)
```

```
# Apply Total Variation Denoising
```

```
denoised_tv = denoise_tv_chambolle(noisy_image.astype(np.float64) / 255.0, weight=0.1,  
channel_axis=None)
```

```
denoised_tv = (denoised_tv * 255).astype(np.uint8)
```

Examples: Image Denoising

Compute SNR

```
snr_noisy = compute_snr(image, noisy_image)
snr_mean = compute_snr(image, denoised_mean)
snr_nlm = compute_snr(image, denoised_nlm)
snr_tv = compute_snr(image, denoised_tv)
```

Display images

```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
axes = axes.ravel()
```

```
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')
```

Examples: Image Denoising

```
axes[1].imshow(noisy_image, cmap='gray')
axes[1].set_title(f'Noisy Image (SNR: {snr_noisy:.2f} dB)')
axes[2].imshow(denoised_mean, cmap='gray')
axes[2].set_title(f'Mean Filter (SNR: {snr_mean:.2f} dB)')
axes[3].imshow(denoised_nlm, cmap='gray')
axes[3].set_title(f'Non-Local Means (SNR: {snr_nlm:.2f} dB)')
axes[4].imshow(denoised_tv, cmap='gray')
axes[4].set_title(f'Total Variation (SNR: {snr_tv:.2f} dB)')

for ax in axes:
    ax.axis('off')

plt.tight_layout()
plt.show()
```

Examples: Image Denoising

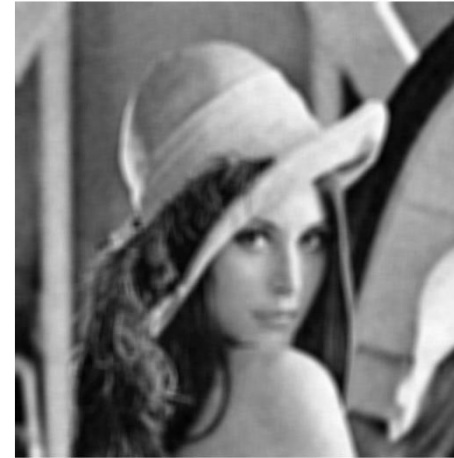
Original Image



Noisy Image (SNR: 0.55 dB)



Mean Filter (SNR: 2.76 dB)



Non-Local Means (SNR: 3.05 dB)



Total Variation (SNR: 3.49 dB)



Examples: Image Deblurring

```
import cv2
import numpy as np
from skimage import restoration, img_as_float
from skimage.metrics import peak_signal_noise_ratio as psnr
import matplotlib.pyplot as plt

# Load an image and convert to grayscale
image = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)

# Check if the image is loaded correctly
if image is None:
    raise ValueError("Image not loaded. Check the file path.")

# Convert the image to float and normalize to [0, 1]
image = img_as_float(image)
```

Examples: Image Deblurring

```
# Simulate motion blur
def motion_blur_kernel(size, angle):
    kernel = np.zeros((size, size))
    center = (size - 1) / 2
    cv2.line(kernel, (int(center - size/2 * np.sin(angle)), int(center + size/2 * np.cos(angle))),
              (int(center + size/2 * np.sin(angle)), int(center - size/2 * np.cos(angle))), 1, thickness=1)
    kernel /= np.sum(kernel)
    return kernel
```

```
kernel = motion_blur_kernel(15, np.pi/4) # Motion blur kernel (size=15, angle=45 degrees)
blurred_image = cv2.filter2D(image, -1, kernel)
```

```
# Add Gaussian noise to the blurred image
def add_noise(image, noise_level=0.01):
    noise = noise_level * np.random.randn(*image.shape)
    noisy_image = image + noise
    noisy_image = np.clip(noisy_image, 0, 1) # Clip values to [0, 1] to avoid overflow
    return noisy_image
```

Examples: Image Deblurring

```
noisy_blurred_image = add_noise(blurred_image)
```

```
# Wiener Filter Deblurring
```

```
def wiener_filter(blurred, kernel, K=0.01):
```

```
    kernel /= np.sum(kernel)
```

```
    H = np.fft.fft2(kernel, s=blurred.shape)
```

```
    G = np.fft.fft2(blurred)
```

```
    F_hat = (np.conj(H) / (np.abs(H)**2 + K)) * G
```

```
    return np.abs(np.fft.ifft2(F_hat))
```

```
wiener_restored = wiener_filter(noisy_blurred_image, kernel, K=0.01)
```

```
# Blind Deconvolution (using Richardson-Lucy algorithm)
```

```
def blind_deblur(blurred, num_iter=50):
```

```
    psf = motion_blur_kernel(15, np.pi/4) # Use the same kernel as the blur
```

```
    deblurred = restoration.richardson_lucy(blurred, psf, num_iter=num_iter)
```

```
    return deblurred
```

Examples: Image Deblurring

```
blind_restored = blind_deblur(noisy_blurred_image)
plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
```

```
plt.subplot(2, 3, 2)
plt.title(f'Blurred Image')
plt.imshow(blurred_image, cmap='gray')
```

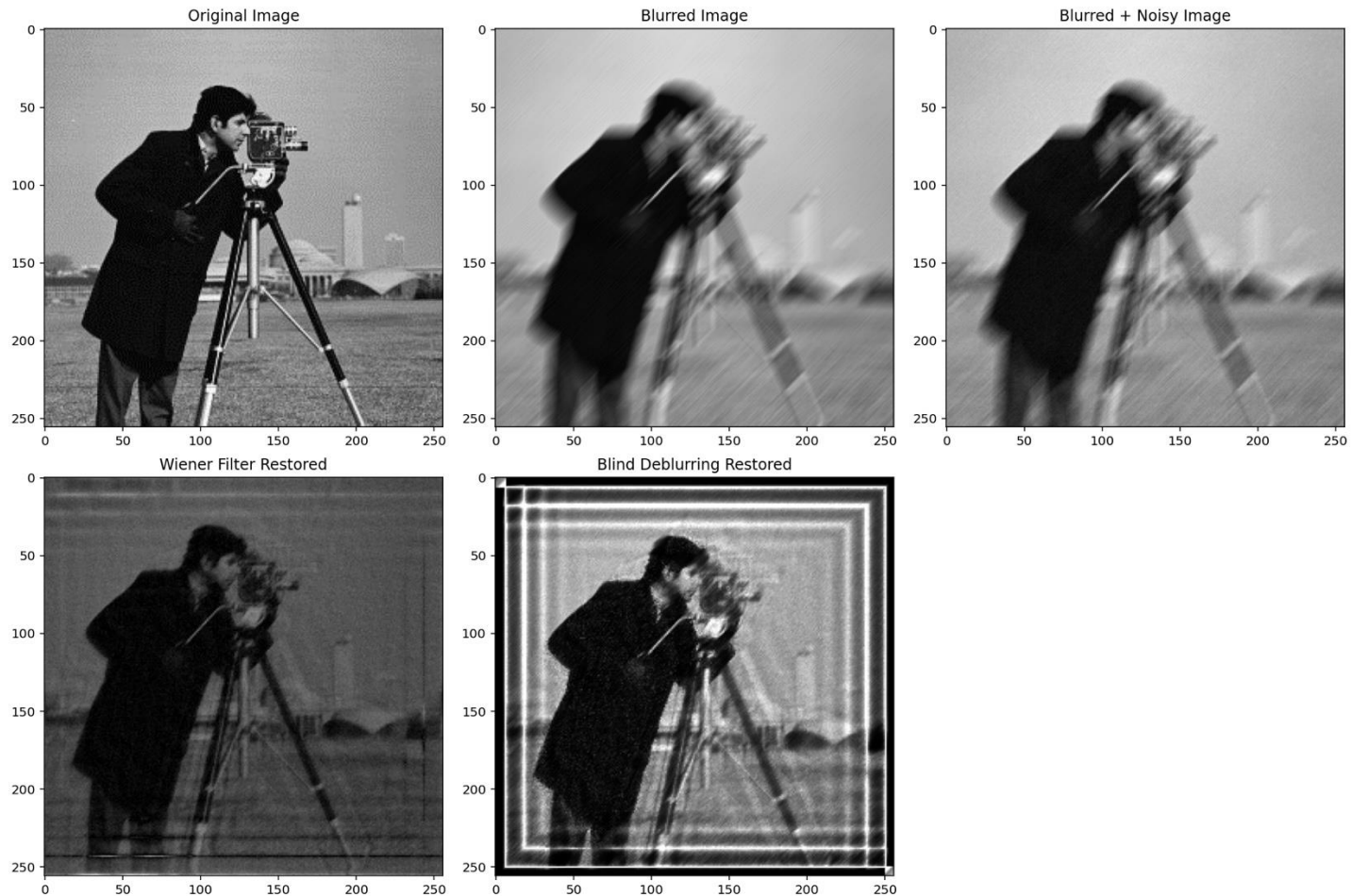
```
plt.subplot(2, 3, 3)
plt.title(f'Blurred + Noisy Image')
plt.imshow(noisy_blurred_image, cmap='gray')
```

```
plt.subplot(2, 3, 4)
plt.title(f'Wiener Filter Restored')
plt.imshow(wiener_restored, cmap='gray')
```

```
plt.subplot(2, 3, 5)
plt.title(f'Blind Deblurring Restored')
plt.imshow(blind_restored, cmap='gray')
```

```
plt.tight_layout()
plt.show()
```

Examples: Image Deblurring



Examples: Image Inpainting

```
import cv2
import numpy as np
from skimage.metrics import peak_signal_noise_ratio as psnr
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('lenna.jpg')

# Create a mask for the missing region (e.g., a rectangle)
mask = np.zeros(image.shape[:2], dtype=np.uint8)
cv2.rectangle(mask, (100, 100), (200, 200), 255, -1) # Define a rectangular hole

# Apply the mask to the image to create a damaged image
damaged_image = image.copy()
damaged_image[mask == 255] = 0 # Set the missing region to black
```

Examples: Image Inpainting

Inpainting using OpenCV's Telea Algorithm

```
telea_inpaint = cv2.inpaint(damaged_image, mask, inpaintRadius=3,  
flags=cv2.INPAINT_TELEA)
```

Inpainting using OpenCV's Navier-Stokes Algorithm

```
ns_inpaint = cv2.inpaint(damaged_image, mask, inpaintRadius=3,  
flags=cv2.INPAINT_NS)
```

Compute PSNR for each result

```
psnr_telea = psnr(image, telea_inpaint)
```

```
psnr_ns = psnr(image, ns_inpaint)
```

Display results

```
plt.figure(figsize=(15, 10))
```

```
plt.subplot(2, 3, 1)
```

```
plt.title('Original Image')
```

```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Examples: Image Inpainting

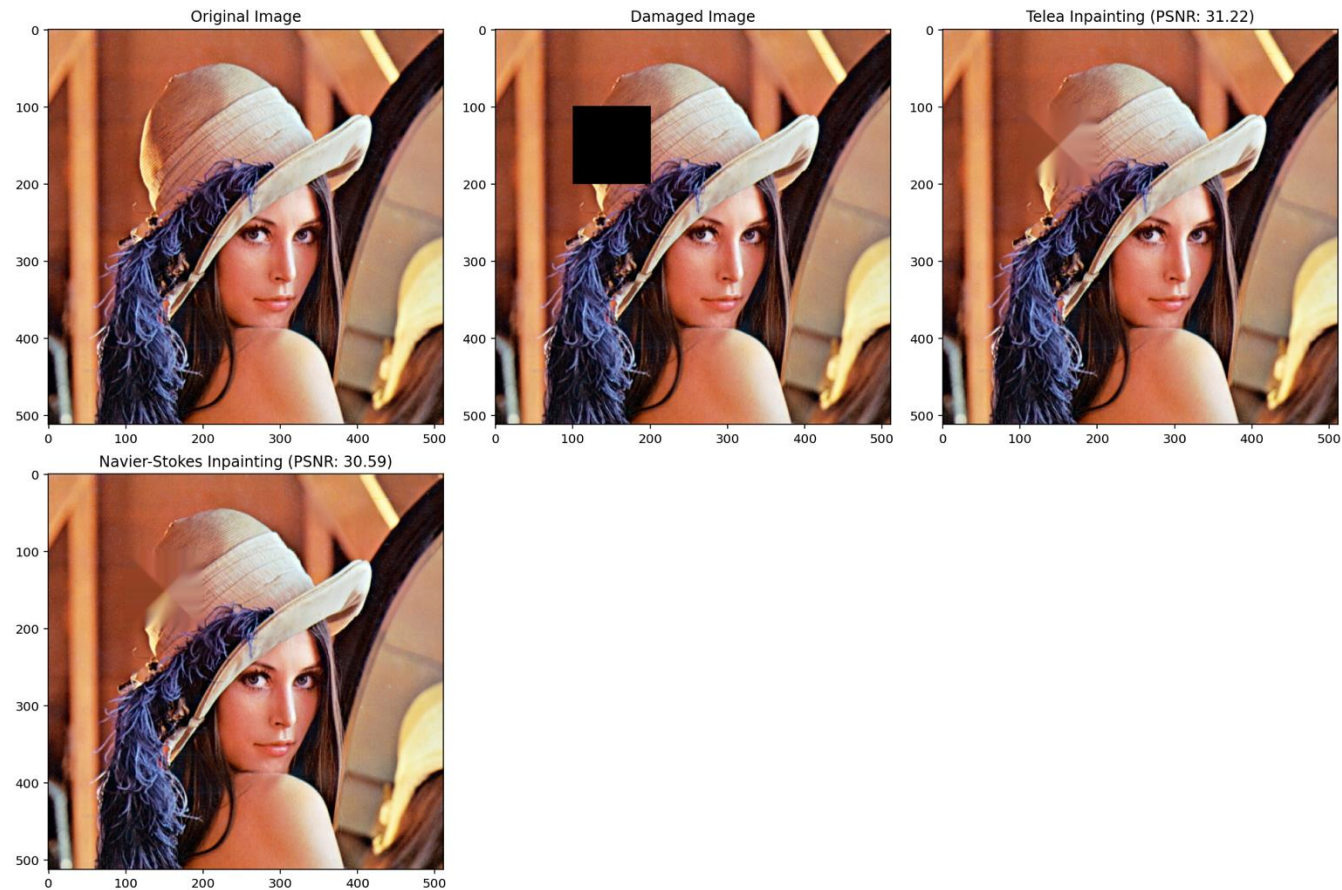
```
plt.subplot(2, 3, 2)
plt.title('Damaged Image')
plt.imshow(cv2.cvtColor(damaged_image, cv2.COLOR_BGR2RGB))
```

```
plt.subplot(2, 3, 3)
plt.title(f'Telea Inpainting (PSNR: {psnr_telea:.2f})')
plt.imshow(cv2.cvtColor(telea_inpainted, cv2.COLOR_BGR2RGB))
```

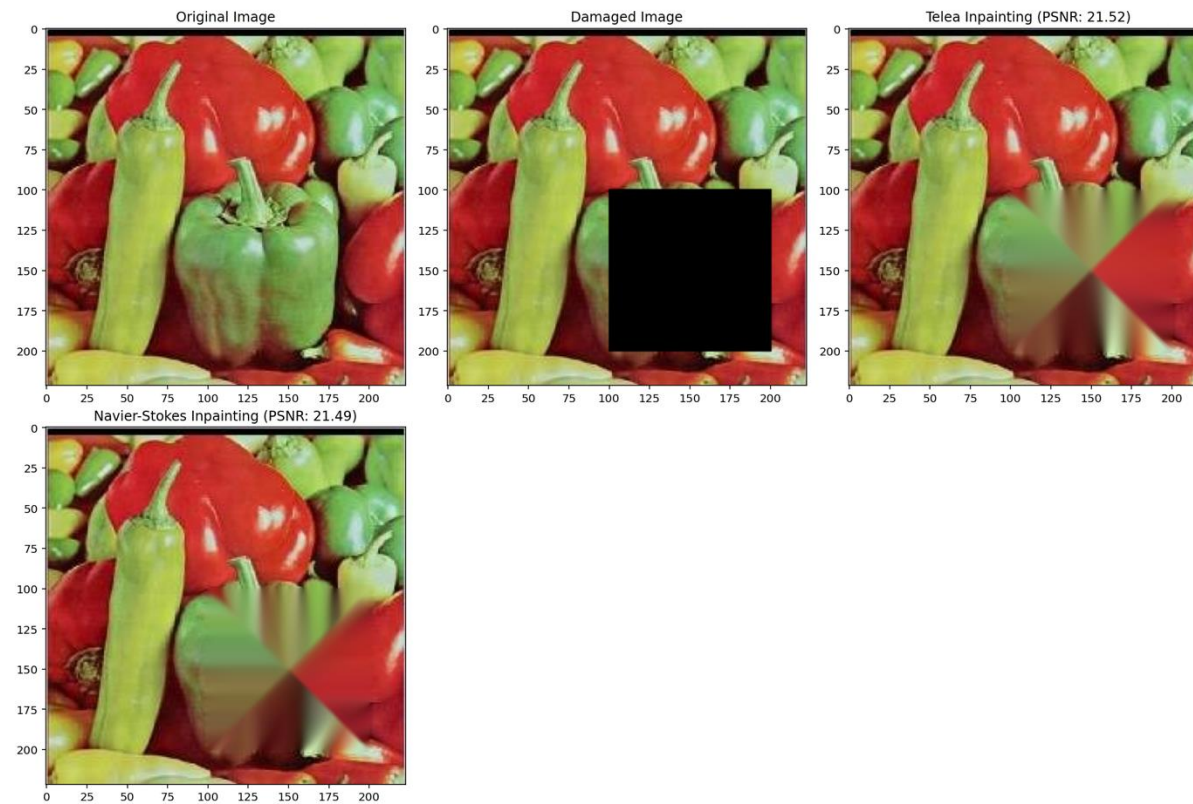
```
plt.subplot(2, 3, 4)
plt.title(f'Navier-Stokes Inpainting (PSNR: {psnr_ns:.2f})')
plt.imshow(cv2.cvtColor(ns_inpainted, cv2.COLOR_BGR2RGB))
```

```
plt.tight_layout()
plt.show()
```

Examples: Image Inpainting



Examples: Image Inpainting





Thank you for your attention!