1 Image deblur

Il problema di deblur consiste nella ricostruzione di un immagine a partire da un dato acquisito mediante il seguente modello:

$$y = Ax + \eta \tag{1}$$

dove :

1. y rappresenta l'immagine corrotta,
2. x rappresenta l'immagine originale che vogliamo ricostruire
3. A rappresenta l'operatore che applica il blur Gaussiano
4. $\eta \sim \mathcal{N}(0, \sigma^2)$ rappresenta una realizzazione di rumore additivo con distribuzione Gaussiana di media $\mu = 0$ e deviazione standard σ

Exercise 1.1. Problema test

- Caricare l'immagine camera dal modulo skimage.data rinormalizzandola nel range $[0, 1]$.

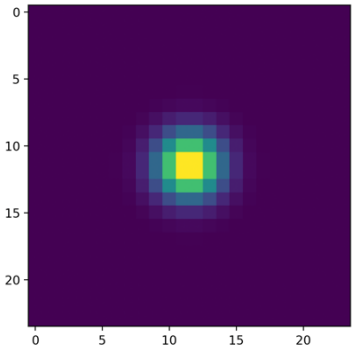
  Per evitare immagini banali si preferisce usare un immagine campione attraverso "imread()"; il codice per usare skimage.data verrà comunque fornito ma sarà commentato

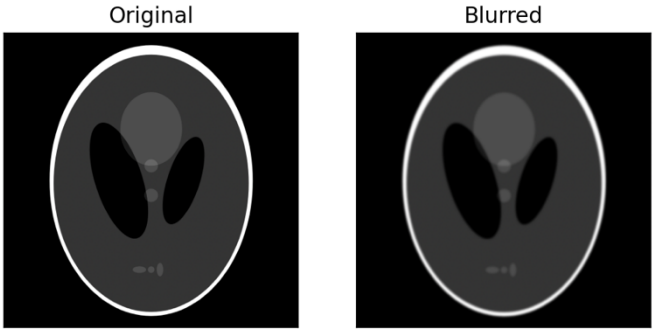
```
X = imread("phantom.png").astype(np.float64)/255.0
# X = data.camera().astype(np.float64)/255
```

- Applicare un blur di tipo gaussiano con deviazione standard 3 il cui kernel ha dimensioni 24 × 24 utilizzando la funzione. Utilizzare prima cv2 (open-cv) e poi la trasformata di Fourier.

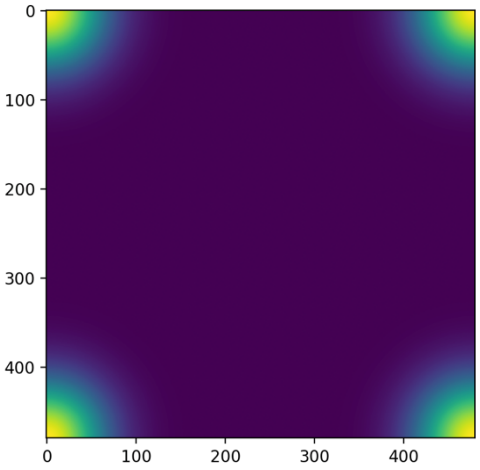
```
# Genera il filtro di blur
k = gaussian_kernel(24,3)
plt.imshow(k)
plt.show()

# Blur with openCV
X_blurred_OpenCV = cv.filter2D(X, -1, k)
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122).imshow(X_blurred_OpenCV, cmap='gray', vmin=0, vmax=1)

plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

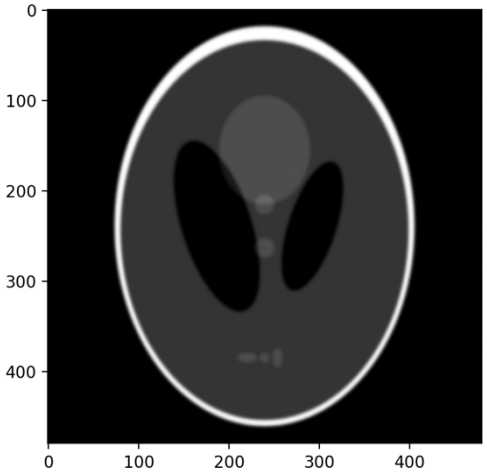
Output





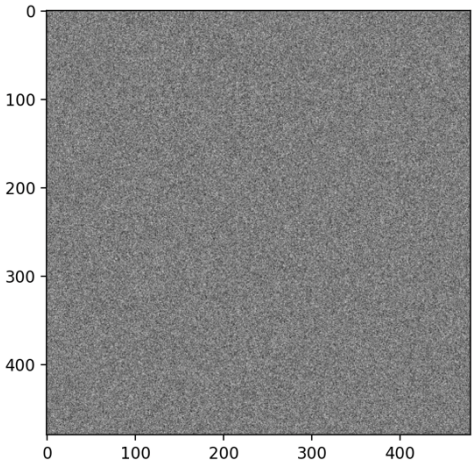
```
# Blur with FFT
K = psf_fft(k, 24, X.shape)
plt.imshow(np.abs(K))
plt.show()

X_blurred = A(X, K)

plt.imshow(X_blurred_FFT,
cmap='gray', vmin=0, vmax=1)

plt.show()
```

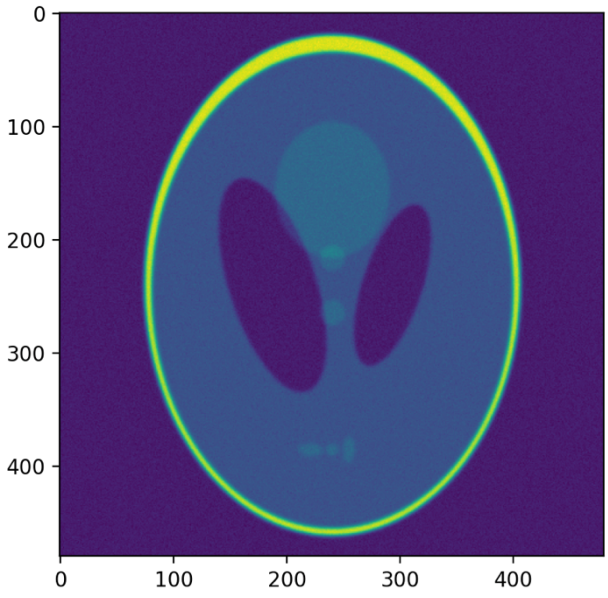


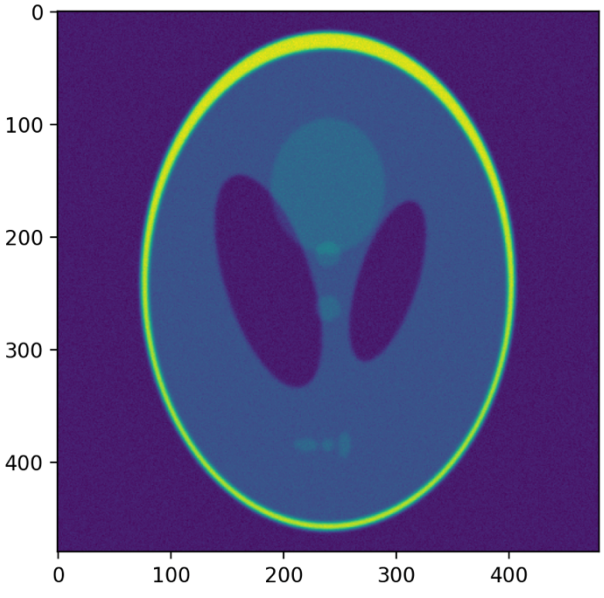


• Aggiungere rumore di tipo gaussiano, con σ = 0.02, usando la funzione np.random.normal( ).

```
# Genera il rumore
sigma = 0.02
np.random.seed(42)
noise = np.random.normal(size=X.shape) * sigma

#mostro il rumore
plt.imshow(noise, cmap='gray')
plt.show()

# Aggiungi blur (OpenCV) e rumore
y_OpenCV = X_blurred_OpenCV + noise
plt.imshow(y_OpenCV)
plt.show()

# Aggiungi blur (FFT) e rumore
y_FFT = X_blurred_FFT + noise
plt.imshow(y_FFT)
plt.show()
```

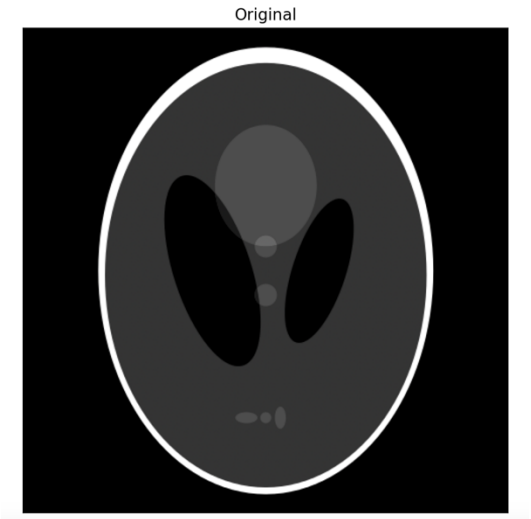


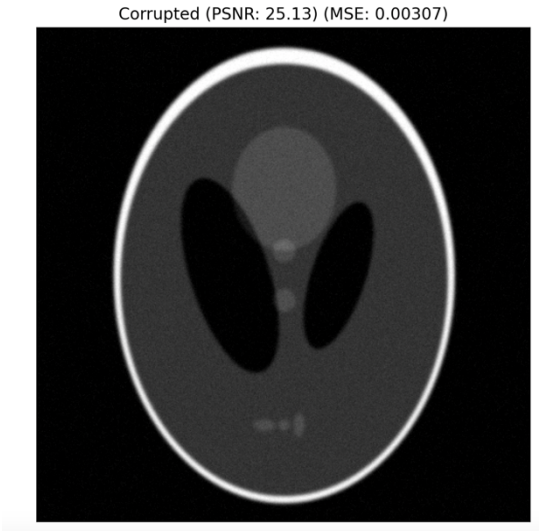




• Calcolare le metriche Peak Signal Noise Ratio (PSNR) e Mean Squared Error (MSE) tra l'immagine degradata e l'immagine esatta usando le funzioni peak signal noise ratio e mean squared error disponibili nel modulo skimage.metrics.

```
y = y_OpenCV
PSNR = metrics.peak_signal_noise_ratio(X, y)
MSE = metrics.mean_squared_error(X, y)
ATy = AT(y_OpenCV, K)


# Visualizziamo i risultati
plt.figure(figsize=(30, 10))
plt.subplot(121).imshow(X, cmap='gray',
vmin=0, vmax=1)

plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122).imshow(y_OpenCV,
cmap='gray', vmin=0, vmax=1)

plt.title(f'Corrupted (PSNR: {PSNR:.2f}) (MSE: {MSE:.2f})')
plt.xticks([]), plt.yticks([])
plt.show()
```





Exercise 1.2. Soluzione naive Una possibile ricostruzione dell'immagine originale x partendo dall'immagine corrotta y è la soluzione naive data dal minimo del seguente problema di ottimizzazione:

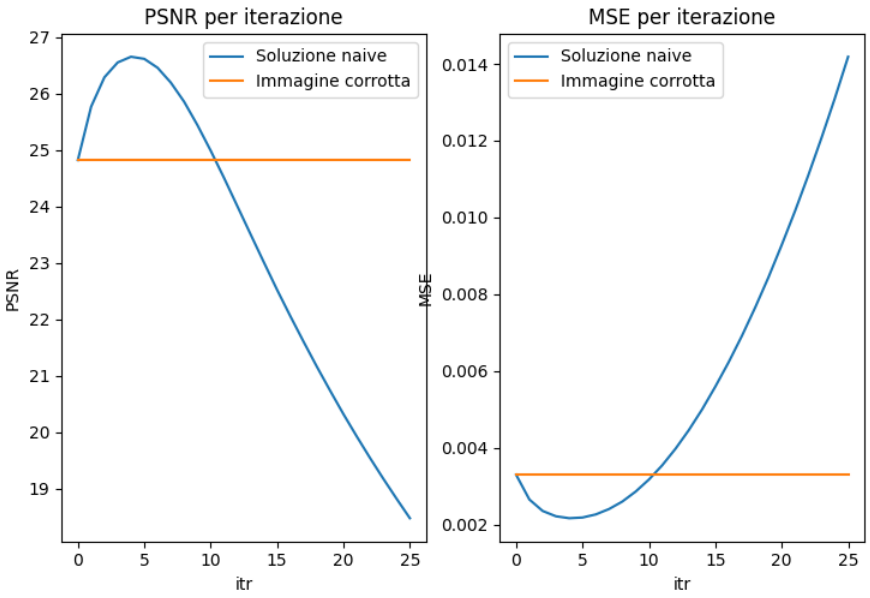
$$x^* = \operatorname{argmin}_x \frac{1}{2}\|Ax - y\|_2^2 \tag{2}$$

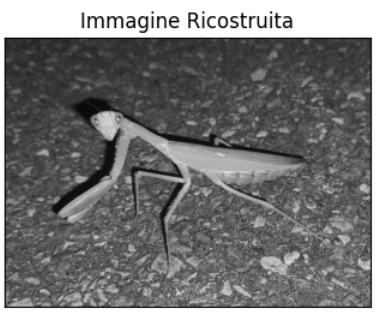
- Utilizzando il metodo del gradiente coniugato implementato dalla funzione `minimize` della libreria `scipy`, calcolare la soluzione naive.
- Analizza l'andamento del PSNR e dell'MSE al variare del numero di iterazioni

```
X = io.imread("test_fotografico.jpg").astype(np.float64) / 255.0
m, n = X.shape

# Genera il filtro di blur
k = gaussian_kernel(24, 3)
plt.imshow(k)
plt.show()

# Blur with openCV
X_blurred = cv.filter2D(X, -1, k) #gexact con convoluzione
plt.subplot(121).imshow(X_blurred, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([]) #rimozione assi x e y
plt.subplot(122).imshow(X_blurred, cmap='gray', vmin=0, vmax=1)
plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()

# Blur with FFT
K = psf_fft(k, 24, X.shape)
plt.imshow(np.abs(K))
plt.show()

X_blurred = A(X, K)
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122).imshow(X_blurred, cmap='gray', vmin=0, vmax=1)
plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()

# Genera il rumore
sigma = 0.02
np.random.seed(42)
noise = np.random.normal(size=X.shape) * sigma

# Aggiungi blur e rumore
y = X_blurred + noise

PSNR = metrics.peak_signal_noise_ratio(X, y)
MSE = metrics.mean_squared_error(X, y)
ATy = AT(y, K)
```

```
# Visualizziamo i risultati
plt.figure(figsize=(30, 10))
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122).imshow(y, cmap='gray', vmin=0, vmax=1)
plt.title(f'Corrupted (PSNR: {PSNR:.2f} MSE {MSE:.5f})')
plt.xticks([]), plt.yticks([])
plt.show()

# Soluzione naive
from scipy.optimize import minimize

# Funzione da minimizzare
def f(x):
    x = x.reshape((m, n)) #passaggio vettore-matrice, #
    Ax = A(x, K) #A(x, K, sf)
    return 0.5 * np.sum(np.square(Ax - y))

# Gradiente della funzione da minimizzare
def df(x):
    x = x.reshape((m, n))
    ATAx = AT(A(x,K),K) #AT(A(x, K, sf), K, sf)
    d = ATAx - ATy
    return d.reshape(m * n) #passaggio matrice-vettore

# Minimizzazione della funzione
x0 = y.reshape(m*n)
max_iter = 25
res = minimize(f, x0, method='CG', jac=df, options={'maxiter':max_iter, 'return_all':True})

PSNR = np.zeros(max_iter + 1)
MSE = np.zeros(max_iter + 1)

for k, x_k in enumerate(res.allvecs):
    PSNR[k] = metrics.peak_signal_noise_ratio(X, x_k.reshape(X.shape))
    MSE[k] = metrics.mean_squared_error(X, x_k.reshape(X.shape))

# Risultato della minimizzazione
X_res = res.x.reshape((m, n))

# PSNR dell'immagine corrotta rispetto all'oginale
starting_PSNR = np.full(PSNR.shape[0], metrics.peak_signal_noise_ratio(X, y))
starting_MSE = np.full(PSNR.shape[0], metrics.mean_squared_error(X, y))
```

```
# Visualizziamo i risultati
ax2 = plt.subplot(1, 3, 1)
ax2.plot(PSNR, label="Soluzione naive")
ax2.plot(starting_PSNR, label="Immagine corrotta")
plt.legend()
plt.title('PSNR per iterazione')
plt.ylabel("PSNR")
plt.xlabel('itr')

ax2 = plt.subplot(1, 3, 2)
ax2.plot(MSE, label="Soluzione naive")
ax2.plot(starting_MSE, label="Immagine corrotta")
plt.legend()
plt.title('MSE per iterazione')
plt.ylabel("MSE")
plt.xlabel('itr')

plt.subplot(1, 3, 3).imshow(X_res, cmap='gray', vmin=0, vmax=1)
plt.title('Immagine Ricostruita')
plt.xticks([]), plt.yticks([])
plt.show()
```





Exercise 1.3. Soluzione regolarizzata Si consideri il seguente problema regolarizzato secondo Tikhonov

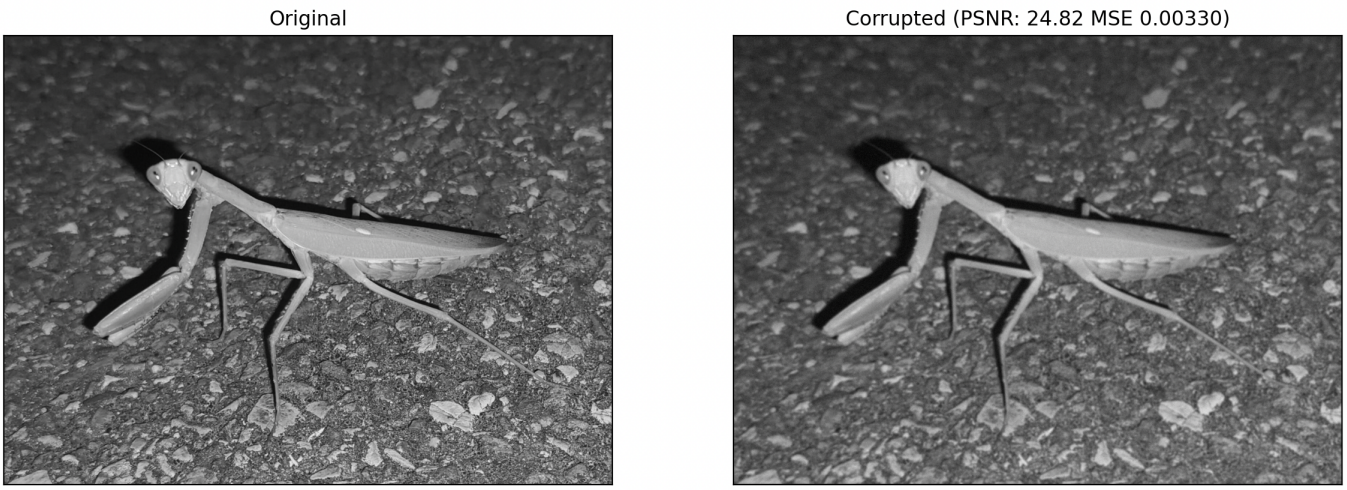
$$x^* = \operatorname*{argmin}_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_2^2 \tag{3}$$

- Analizzare l'andamento del PSNR e dell'MSE al variare del numero di iterazioni.
- Facendo variare il parametro di regolarizzazione λ, analizzare come questo influenza le prestazioni del metodo analizzando le immagini.
- Scegliere λ con il metodo di discrepanza.
- Scegliere λ attraverso test sperimentali come il valore che massimizza il valore del PSNR. Confrontare il valore ottenuto con quella della massima discrepanza.

```
def f(x, L):
    nsq = np.sum(np.square(x))
    x  = x.reshape((m, n))
    Ax = A(x, K)
    return 0.5 * np.sum(np.square(Ax - y)) + 0.5 * L * nsq

# Gradiente ∇f = AT·(Ax−y)+L·x
def df(x, L):
    Lx = L * x
    x = x.reshape(m, n)
    ATAx = AT(A(x,K),K)
    d = ATAx - ATy
    return d.reshape(m * n) + Lx

for i, L in enumerate(lambdas):
    # Esegui la minimizzazione con al massimo 50 iterazioni
    max_iter = 50
    res = minimize(f, x0, (L), method='CG', jac=df, options={'maxiter':max_iter})

    X_curr = res.x.reshape(X.shape)
    images.append(X_curr)

    PSNR = metrics.peak_signal_noise_ratio(X, X_curr)
    PSNRs.append(PSNR)
    MSE = metrics.mean_squared_error(X, X_curr)
    MSEs.append(MSE)

    diff = np.sum(np.square(y-A(X_curr, K)))
    tolerance = 1.1 * np.sum(np.square(noise))

    print(f'PSNR: {PSNR:.3f} (\u03BB = {L:.5f})')
    print(f'MSE: {MSE:.6f} (\u03BB = {L:.5f})')
    print(f'Discrepanza: {diff:.3f} (toll: {tolerance:.3f})\n')

plt.plot(lambdas,PSNRs)
plt.title('PSNR per $\lambda$')
plt.ylabel("PSNR")
plt.xlabel('$\lambda$')
plt.show()
```

```
plt.plot(lambdas,MSEs)
plt.title('MSE per $\lambda$')
plt.ylabel("MSE")
plt.xlabel('$\lambda$')
plt.show()

plt.figure(figsize=(30, 10))

plt.subplot(1, len(lambdas) + 2, 1).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title("Originale")
plt.xticks([]), plt.yticks([])
plt.subplot(1, len(lambdas) + 2, 2).imshow(y, cmap='gray', vmin=0, vmax=1)
plt.title("Corrotta")
plt.xticks([]), plt.yticks([])


for i, L in enumerate(lambdas):
  plt.subplot(1, len(lambdas) + 2, i + 3).imshow(images[i], cmap='gray', vmin=0, vmax=1)
  plt.title(f"Ricostruzione ($\lambda$ = {L:.2f})")
plt.show()
```

Output con immagine fotografica:



PSNR: 13.651 ($\lambda = 0.00010$)

MSE: 0.043141 ($\lambda = 0.00010$)

Discrepanza: 359.980 (toll: 532.435)

PSNR: 14.658 ($\lambda = 0.00020$)

MSE: 0.034211 ($\lambda = 0.00020$)

Discrepanza: 361.617 (toll: 532.435)

PSNR: 13.748 (λ = 1.00000)

MSE: 0.042190 (λ = 1.00000)

Discrepanza: 46967.738 (toll: 532.435)

--

PSNR: 27.027 (λ = 0.01500)

MSE: 0.001983 (λ = 0.01500)

Discrepanza: 460.290 (toll: 532.435)

PSNR: 27.186 (λ = 0.02000)

MSE: 0.001912 (λ = 0.02000)

Discrepanza: 498.793 (toll: 532.435)

PSNR: 27.201 (λ = 0.02250)

MSE: 0.001905 (λ = 0.02250)

Discrepanza: 520.875 (toll: 532.435)

PSNR: 27.190 (λ = 0.02500)

MSE: 0.001910 (λ = 0.02500)

Discrepanza: 544.863 (toll: 532.435)

PSNR: 27.160 (λ = 0.02750)

MSE: 0.001923 (λ = 0.02750)

Discrepanza: 570.752 (toll: 532.435)

PSNR: 27.115 (λ = 0.03000)

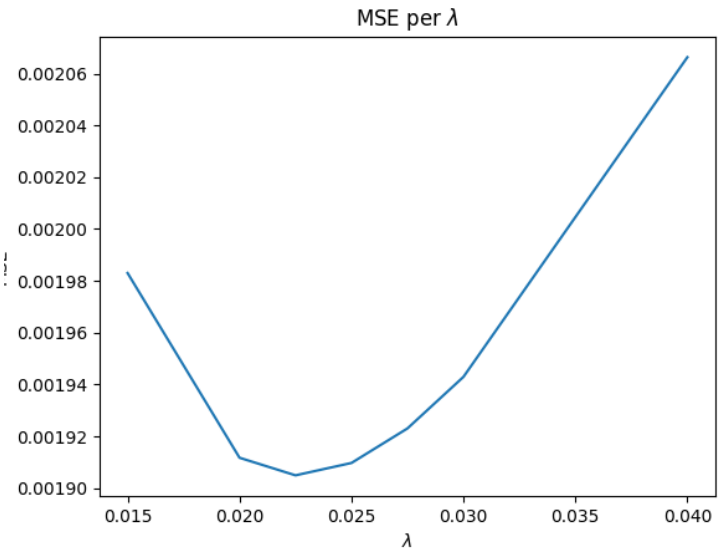
MSE: 0.001943 (λ = 0.03000)

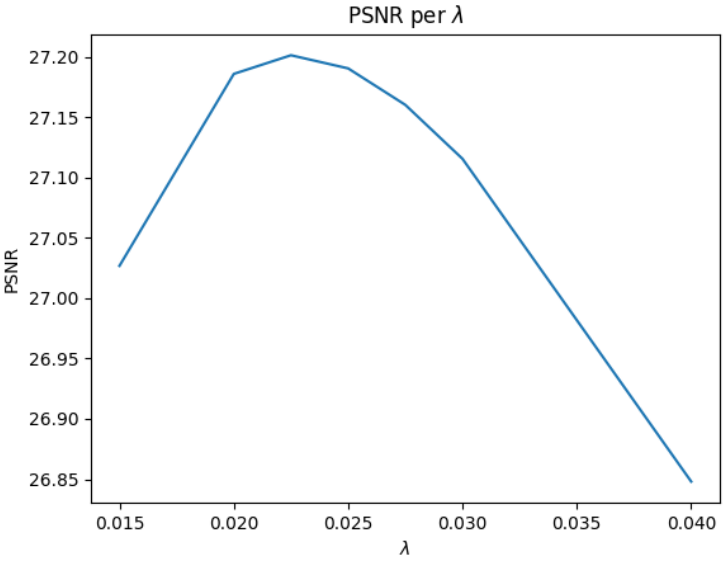
Discrepanza: 598.521 (toll: 532.435)

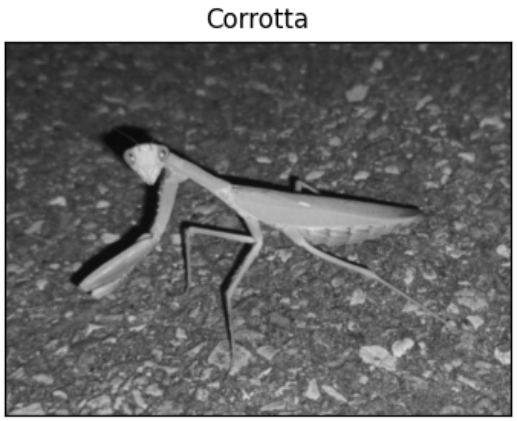
PSNR: 26.848 (λ = 0.04000)

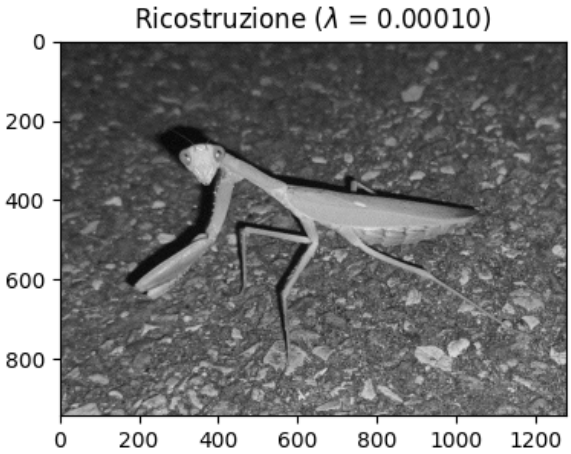
MSE: 0.002066 (λ = 0.04000)

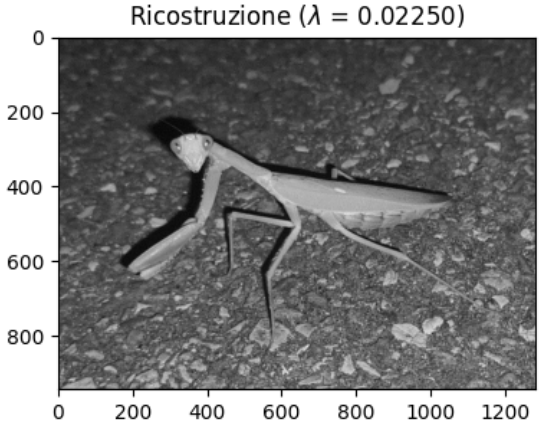
Discrepanza: 728.202 (toll: 532.435)

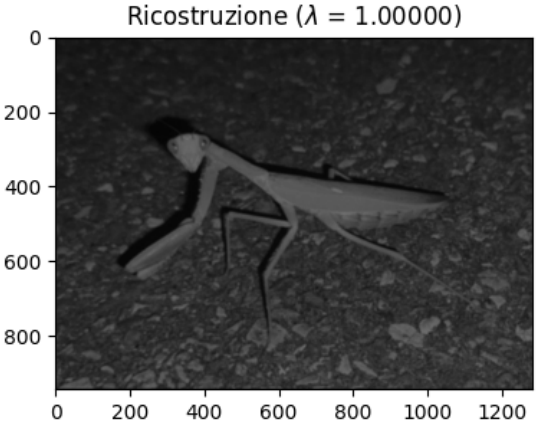




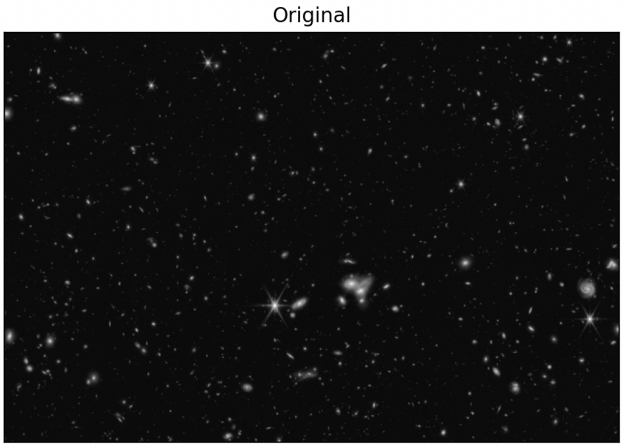


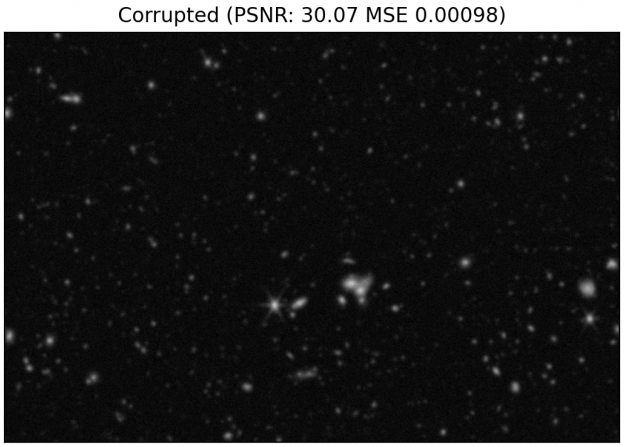






Test con immagine telescopio:

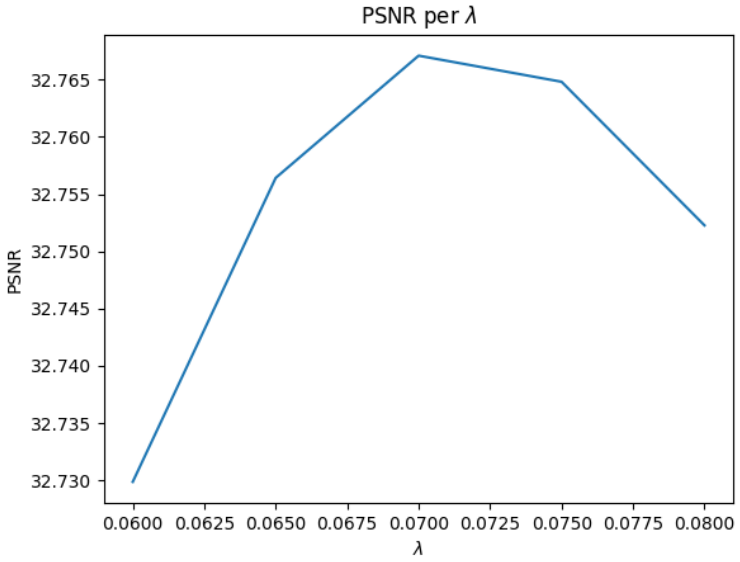




PSNR: 32.730 (λ = 0.06000)

MSE: 0.000533 (λ = 0.06000)

Discrepanza: 150.433 (toll: 173.084)



PSNR: 32.756 (λ = 0.06500)

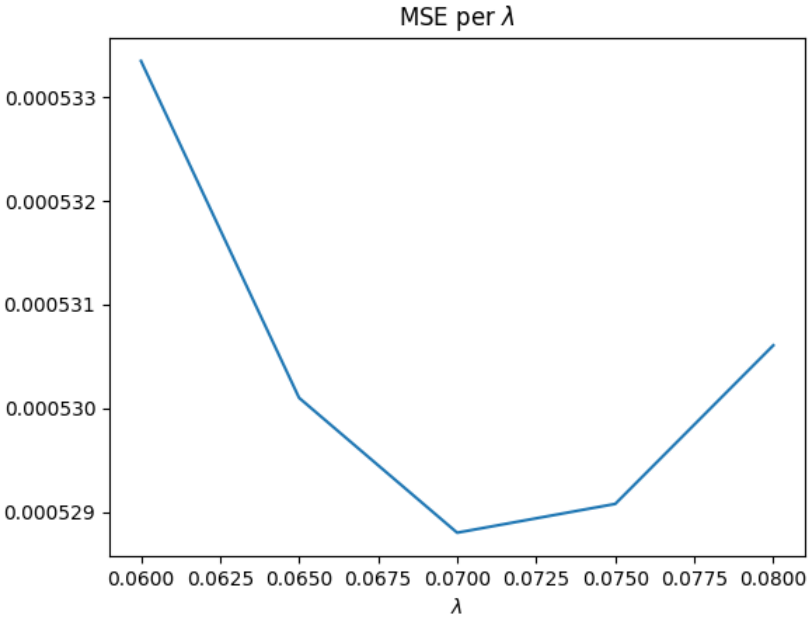
MSE: 0.000530 (λ = 0.06500)

Discrepanza: 152.263 (toll: 173.084)

PSNR: 32.767 (λ = 0.07000)

MSE: 0.000529 (λ = 0.07000)

Discrepanza: 154.145 (toll: 173.084)



PSNR: 32.765 (λ = 0.07500)

MSE: 0.000529 (λ = 0.07500)

Discrepanza: 156.080 (toll: 173.084)

PSNR: 32.752 (λ = 0.08000)

MSE: 0.000531 (λ = 0.08000)

Discrepanza: 158.059 (toll: 173.084)

PSNR: 13.494 (λ = 0.00010)

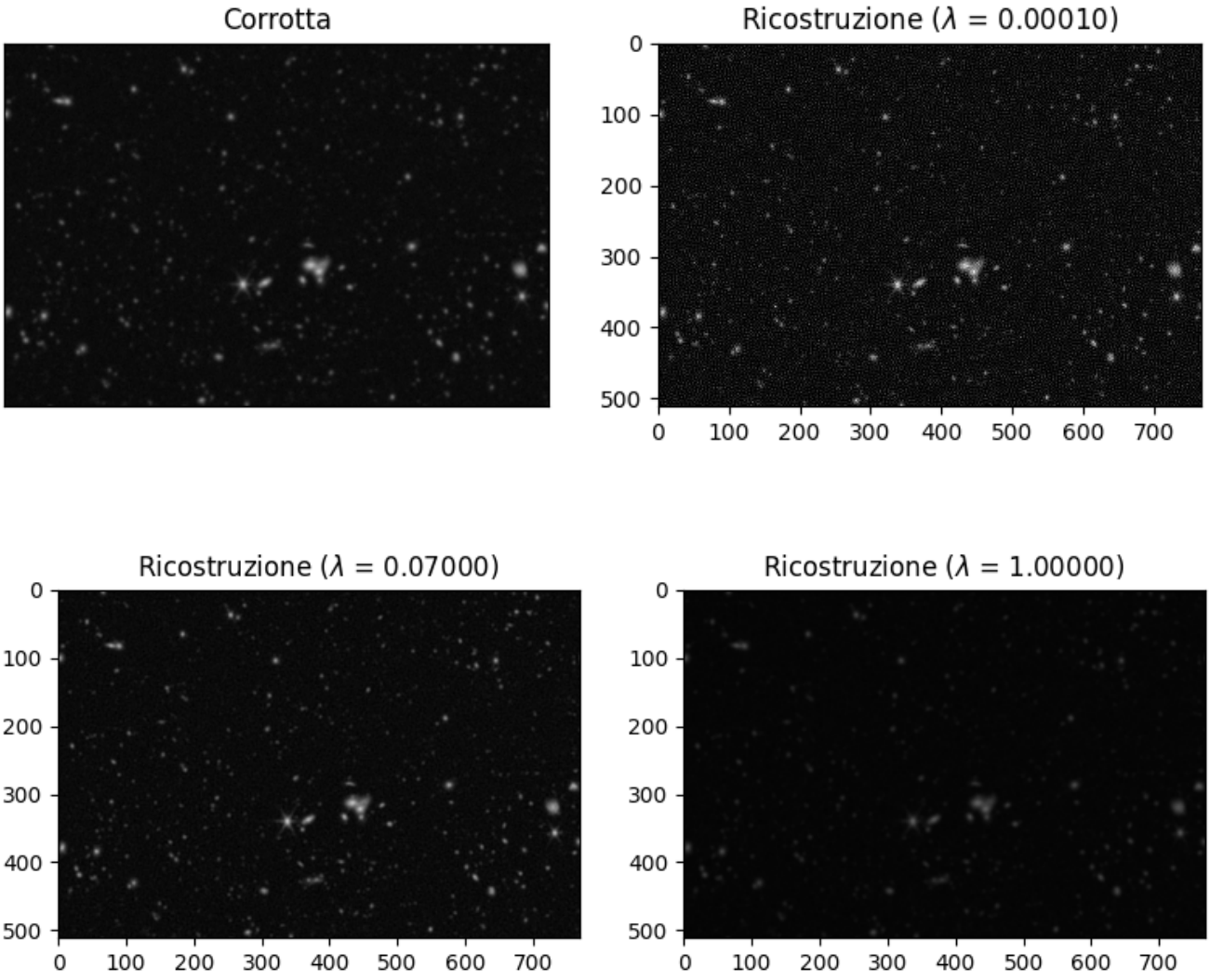
MSE: 0.044730 (λ = 0.00010)

Discrepanza: 116.768 (toll: 173.084)

PSNR: 26.622 (λ = 1.00000)

MSE: 0.002176 (λ = 1.00000)

Discrepanza: 705.046 (toll: 173.084)



Metodo della Varianza Totale:

Exercise 1.5 (Facoltativo TV)**.** Un'altra funzione adatta come termine di regolarizzazione è la Variazione Totale. Data x immagine di dimensioni $m \times n$ la variazione totale TV di x è definita come:

$$TV(x) = \sum_{i}^{n} \sum_{j}^{m} \sqrt{||\nabla x(i,j)||_2^2 + \epsilon^2} \tag{4}$$

Come nei casi precedenti il problema di minimo che si va a risolvere è il seguente:

$$x^* = \arg\min_x \frac{1}{2}||Ax - b||_2^2 + \lambda TV(x) \tag{5}$$

il cui gradiente ∇f è dato da

$$\nabla f(x) = (A^T Ax - A^T b) + \lambda \nabla TV(x) \tag{6}$$

- Utilizzando il metodo del gradiente e la funzione `minimize`, calcolare la soluzione del precendente problema di minimo regolarizzato con la funzione TV per differenti valori di λ, utilizzando le funzioni `totvar` e `grad_totvar`.

- Per calcolare il gradiente dell'immagine ∇u usiamo la funzione 'np.gradient' che approssima la derivata per ogni pixel calcolando la differenza tra pixel adiacenti. I risultati sono due immagini della stessa dimensione dell'immagine in input, una che rappresenta il valore della derivata orizzontale e l'altra della derivata verticale . Il gradiente dell'immagine nel punto (i, j) è quindi un vettore di due componenti, uno orizzontale contenuto e uno verticale.

- Per risolvere il problema di minimo è necessario anche calcolare il gradiente della variazione totale che è definito nel modo seguente

$$\nabla TV(u) = -div\left(\frac{\nabla u}{\sqrt{||\nabla u||_2^2 + \epsilon^2}}\right) \tag{7}$$

  dove la divergenza è definita come

$$div(F) = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} \tag{8}$$

  $div(F)$ è la divergenza del campo vettoriale F, nel nostro caso F ha due componenti dati dal gradiente dell'immagine ∇u scalato per il valore $\frac{1}{\sqrt{||\nabla u||_2^2+\epsilon^2}}$. Per calcolare la divergenza bisogna calcolare la derivata orizzontale $\frac{\partial F_x}{\partial x}$ della componente x di F e sommarla alla derivata verticale $\frac{\partial F_y}{\partial y}$ della componente y di F. Per specificare in quale direzione calcolare la derivata con la funzione 'np.gradient' utilizziamo il parametro 'axis $= 0$' per l'orizzontale e 'axis $= 1$' per la verticale.

```
def f(x, L):
    x = x.reshape((m, n))
    TV = totvar(x)
    Ax = A(x, K)
    return 0.5 * np.sum(np.square(Ax - y)) + L * TV

# Gradiente della funzione da minimizzare
def df(x, L):
    x = x.reshape((m, n))
    LDTV = L * grad_totvar(x)
    ATAx = AT(A(x,K),K)
    d = ATAx - y
    return d.reshape(m * n) + LDTV.reshape(m * n)
```

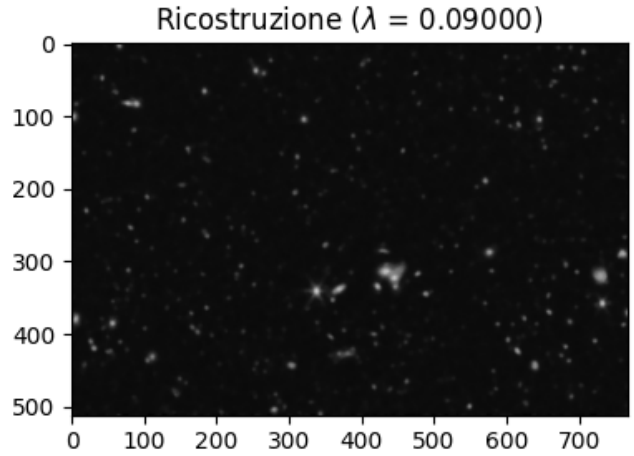
Output:

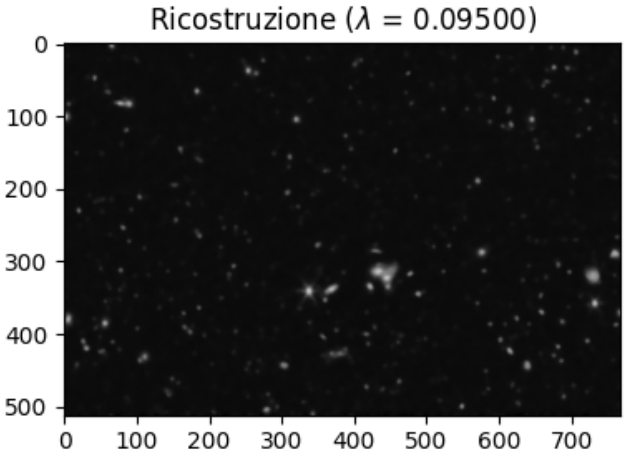
PSNR: 31.300 (λ = 0.08700)
MSE: 0.000741 (λ = 0.08700)
Discrepanza: 186.244 (toll: 173.084)

PSNR: 31.423 (λ = 0.09000)
MSE: 0.000721 (λ = 0.09000)
Discrepanza: 185.887 (toll: 173.084)

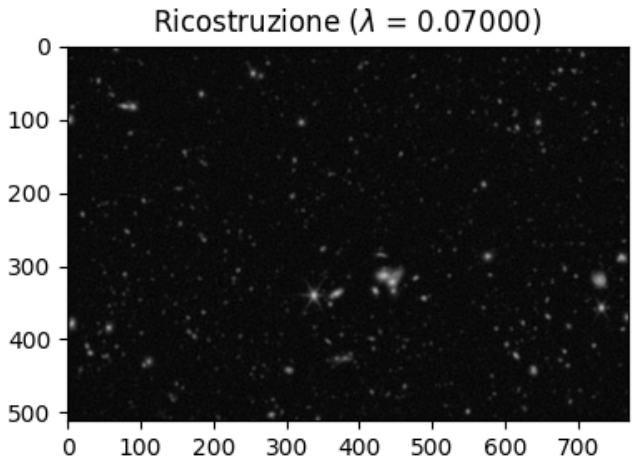
PSNR: 31.324 (λ = 0.09500)
MSE: 0.000737 (λ = 0.09500)
Discrepanza: 188.554 (toll: 173.084)

PSNR: 31.433 (λ = 0.08845)
MSE: 0.000719 (λ = 0.08845)
Discrepanza: 185.371 (toll: 173.084)





(Tikhonov)



Si nota che TV approssima meglio i bordi (Guardare i due punti bianchi in alto a sinistra) mentre Tikhonov è ottimo per rimuovere il rumore dell'immaggine.

Exercise 1.4.

- Ripetere i punti precedenti utilizzando anche l'operatore downsampling con i seguenti fattori di scaling $sf = 2, 4, 8, 16$.
- Testare i punti precedenti su due immagini in scala di grigio con caratteristiche differenti (per esempio, un'immagine tipo fotografico e una ottenuta con uno strumento differente, microscopio o altro).
- Degradare le nuove immagini applicando, mediante le funzioni `gaussian_kernel()`, `psf_fft()`, l'operatore di blur con parametri:
  - $\sigma = 0,5$ dimensione del kernel 7×7 e 9×9
  - $\sigma = 1,3$ dimensione del kernel 5×5
  - Aggiungendo rumore gaussiano con deviazione standard nell' intervallo $(0, 0, 05]$.

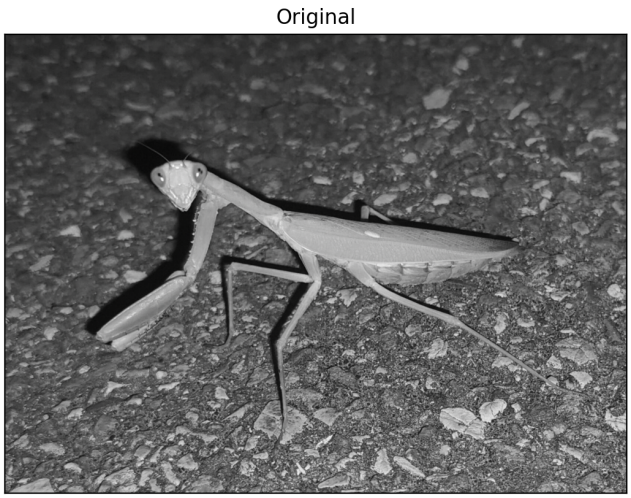
Verrà usata una immagine ottenuta tramite fotocamera e una ottenuta tramite telescopio;

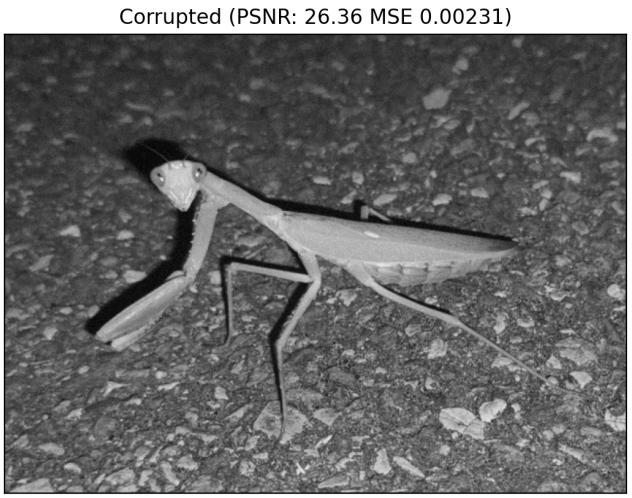
```
X = io.imread("./test_spazio.png").astype(np.float64) / 255.0
m, n = X.shape
sf = 16

# Genera il filtro di blur
k = gaussian_kernel(24, 3)
plt.imshow(k)
plt.show()

# Blur with openCV
X_blurred = cv.filter2D(X, -1, k)
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([]) #rimozione assi x e y
plt.subplot(122).imshow(X_blurred, cmap='gray', vmin=0, vmax=1)
plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()

# Blur with FFT
K = psf_fft(k, 24, X.shape)
plt.imshow(np.abs(K))
plt.show()

X_blurred = A(X, K, sf)
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([]) #rimozione assi x e y
plt.subplot(122).imshow(X_blurred, cmap='gray', vmin=0, vmax=1)
plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()

# Genera il rumore
sigma = 0.02
np.random.seed(42)
noise = np.random.normal(size=X_blurred.shape) * sigma
```

```
# Aggiungi blur e rumore
y = X_blurred + noise
ATy = AT(y, K, sf)
PSNR = metrics.peak_signal_noise_ratio(X, ATy)
MSE = metrics.mean_squared_error(X, ATy)

# Visualizziamo i risultati
plt.figure(figsize=(30, 10))
plt.subplot(121).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122).imshow(y, cmap='gray', vmin=0, vmax=1)
plt.title(f'Corrupted (PSNR: {PSNR:.2f} MSE {MSE:.5f})')
plt.xticks([]), plt.yticks([])
plt.show()

def f(x, L):  # Funzione da minimizzare
    nsq = np.sum(np.square(x))
    x  = x.reshape((m, n))
    Ax = A(x, K, sf)
    return 0.5 * np.sum(np.square(Ax - y)) + L*nsq

def df(x, L):  # Gradiente della funzione da minimizzare
    Lx = L * x
    x = x.reshape(m, n)
    ATAx = AT(A(x,K, sf),K,sf)
    d = ATAx - ATy
    return d.reshape(m * n) + Lx

x0 = ATy.reshape(m*n)
lambdas = np.linspace(0.001, 0.01, 5)
PSNRs = []
MSEs = []
images = []

for i, L in enumerate(lambdas):
    max_iter = 50
    res = minimize(f, x0, (L), method='CG', jac=df, options={'maxiter':max_iter})

    # Aggiungi la ricostruzione nella lista images
    X_curr = res.x.reshape(X.shape)
    images.append(X_curr)

    # Stampa il PSNR e MSE per il valore di lambda attuale
    PSNR = metrics.peak_signal_noise_ratio(X, X_curr)
    PSNRs.append(PSNR)

    MSE = metrics.mean_squared_error(X, X_curr)
    MSEs.append(MSE)
    print(f'PSNR: {PSNR:.3f} (\u03BB = {L:.5f})')
    print(f'MSE: {MSE:.6f} (\u03BB = {L:.5f})')

    # Calcolo della discrepanza tra il rumore simulato e il rumore osservato
    diff = np.sum(np.square(y-A(X_curr, K, sf)))
    # Calcolo della tolleranza
    tolerance = 1.1 * np.sum(np.square(noise))
    print(f'Discrepanza: {diff:.3f} ({tolerance:.3f})\n')
```

```
# Visualizziamo i risultati
plt.plot(lambdas,PSNRs)
plt.title('PSNR per $\lambda$')
plt.ylabel("PSNR")
plt.xlabel('$\lambda$')
plt.show()

plt.plot(lambdas,MSEs)
plt.title('MSE per $\lambda$')
plt.ylabel("MSE")
plt.xlabel('$\lambda$')
plt.show()

plt.figure(figsize=(30, 10))
plt.subplot(1, len(lambdas) + 2, 1).imshow(X, cmap='gray', vmin=0, vmax=1)
plt.title("Originale")
plt.xticks([]), plt.yticks([])
plt.subplot(1, len(lambdas) + 2, 2).imshow(y, cmap='gray', vmin=0, vmax=1)
plt.title("Corrotta")
plt.xticks([]), plt.yticks([])

for i, L in enumerate(lambdas):
  plt.subplot(1, len(lambdas) + 2, i + 3).imshow(images[i], cmap='gray', vmin=0, vmax=1)
  plt.title(f"Ricostruzione ($\lambda$ = {L:.5f})")
plt.show()
```

sf = 2, Test fotografico
Test con k = gaussian_kernel(7, 0.5):

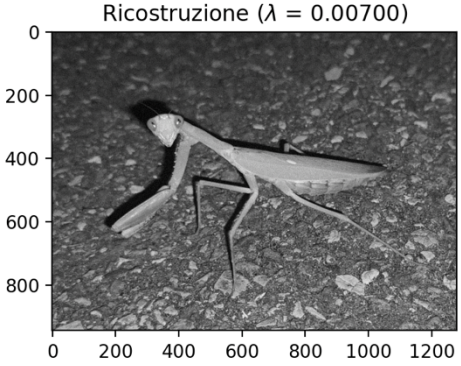


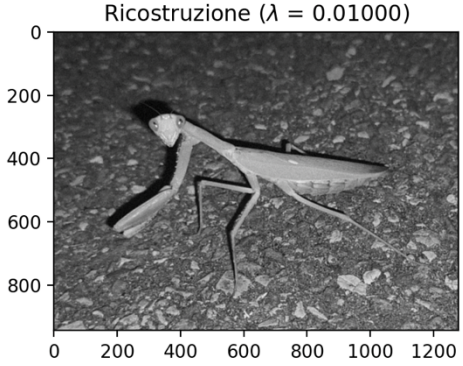


PSNR: 28.037 (λ = 0.00600)
MSE: 0.001572 (λ = 0.00600)
Discrepanza: 29.041 (132.875)

PSNR: 28.039 (λ = 0.00700)
MSE: 0.001571 (λ = 0.00700)
Discrepanza: 31.066 (132.875)

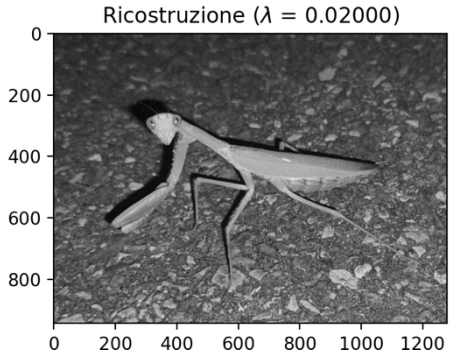
PSNR: 27.869 (λ = 0.01000)
MSE: 0.001633 (λ = 0.01000)
Discrepanza: 106.087 (132.875)

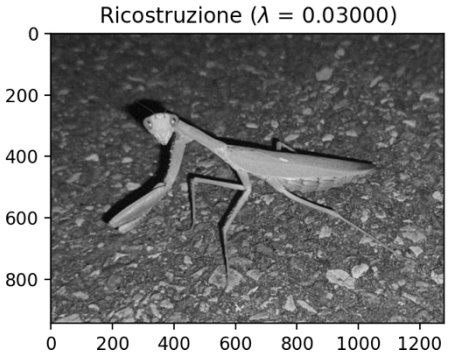




PSNR: 27.644 (λ = 0.02000)
MSE: 0.001720 (λ = 0.02000)
Discrepanza: 130.238 (132.875)

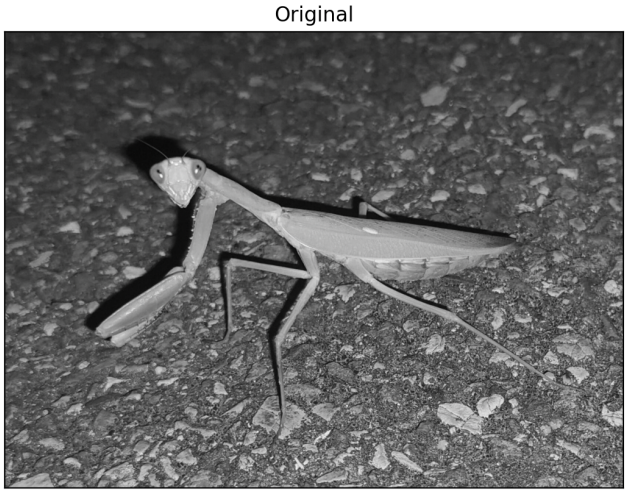
PSNR: 27.357 (λ = 0.03000)
MSE: 0.001838 (λ = 0.03000)
Discrepanza: 161.698 (132.875)

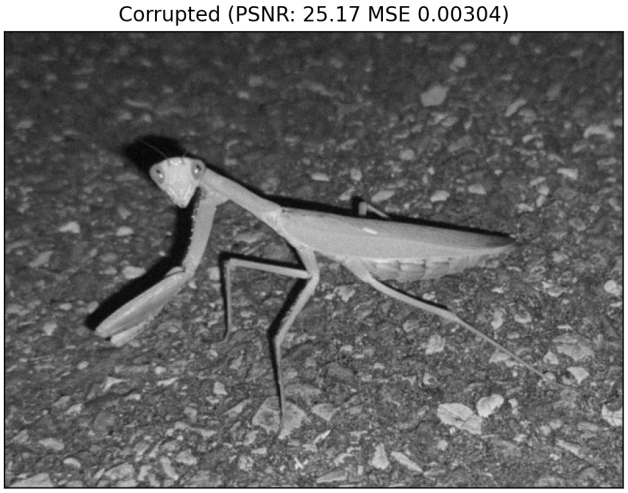




Test con k = *gaussian_kernel* (9, 0.5): Analogo al precedente

Test con k = *gaussian_kernel* (5, 1.3):



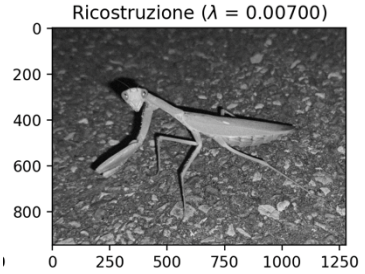


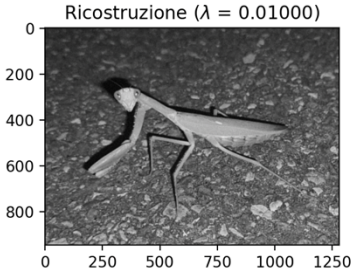
PSNR: 27.038 (λ = 0.00600)
MSE: 0.001978 (λ = 0.00600)
Discrepanza: 67.955 (132.875)

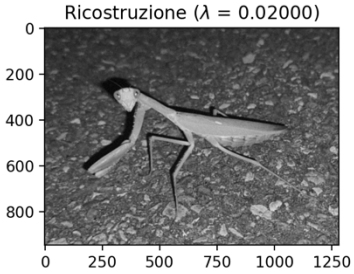
PSNR: 27.171 (λ = 0.00700)
MSE: 0.001918 (λ = 0.00700)
Discrepanza: 68.274 (132.875)

PSNR: 26.482 (λ = 0.01000)
MSE: 0.002248 (λ = 0.01000)
Discrepanza: 135.540 (132.875)

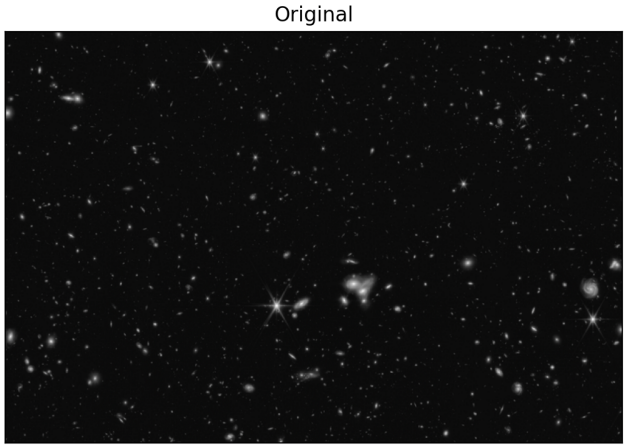
PSNR: 26.301 (λ = 0.02000)
MSE: 0.002344 (λ = 0.02000)
Discrepanza: 159.164 (132.875)

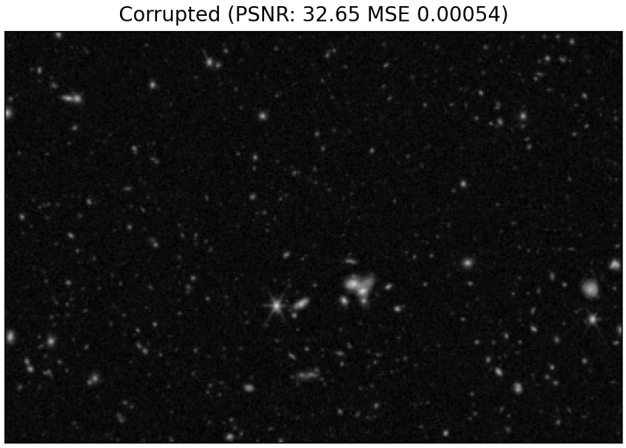




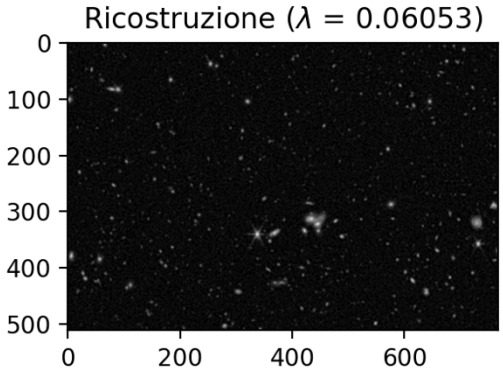


Sf=2, Test foto telescopio
Test con k = *gaussian_kernel* (7, 0.5):

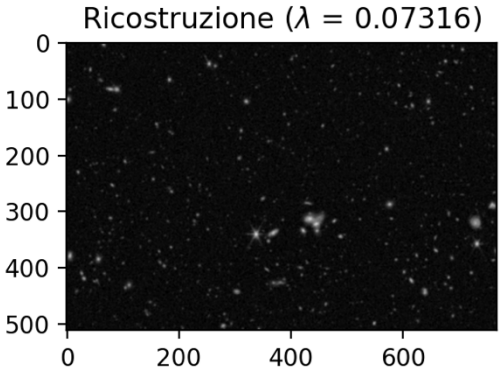




PSNR: 31.770 (λ = 0.06053)
MSE: 0.000665 (λ = 0.06053)
Discrepanza: 21.333 (43.345)



PSNR: 32.807 (λ = 0.07316)
MSE: 0.000524 (λ = 0.07316)
Discrepanza: 25.055 (43.345)

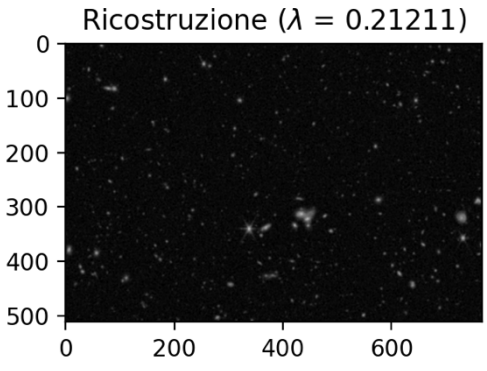


PSNR: 32.749 (λ = 0.08579)
MSE: 0.000531 (λ = 0.08579)
Discrepanza: 26.315 (43.345)

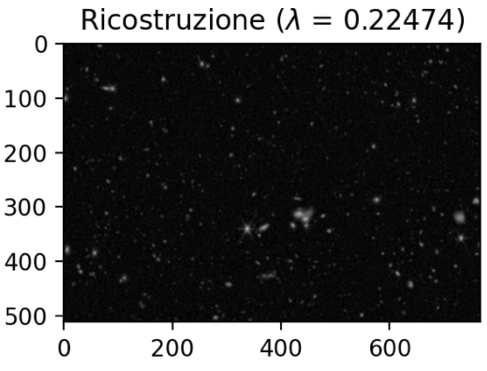
PSNR: 31.939 (λ = 0.13632)
MSE: 0.000640 (λ = 0.13632)
Discrepanza: 26.099 (43.345)

PSNR: 31.843 (λ = 0.14895)
MSE: 0.000654 (λ = 0.14895)
Discrepanza: 29.042 (43.345)

PSNR: 32.104 (λ = 0.16158)
MSE: 0.000616 (λ = 0.16158)
Discrepanza: 31.921 (43.345)

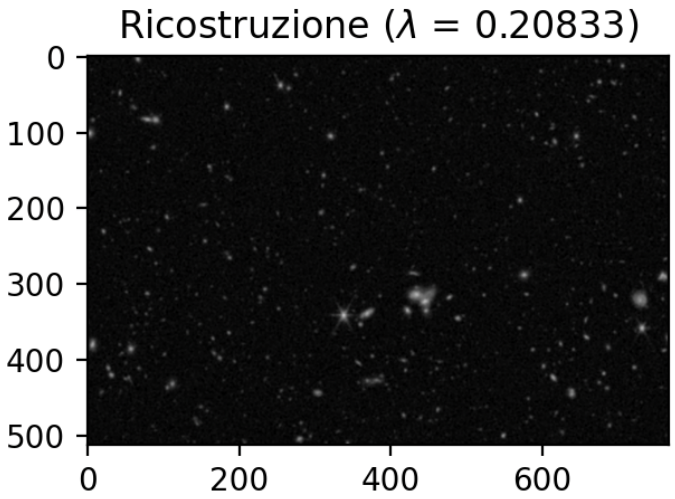


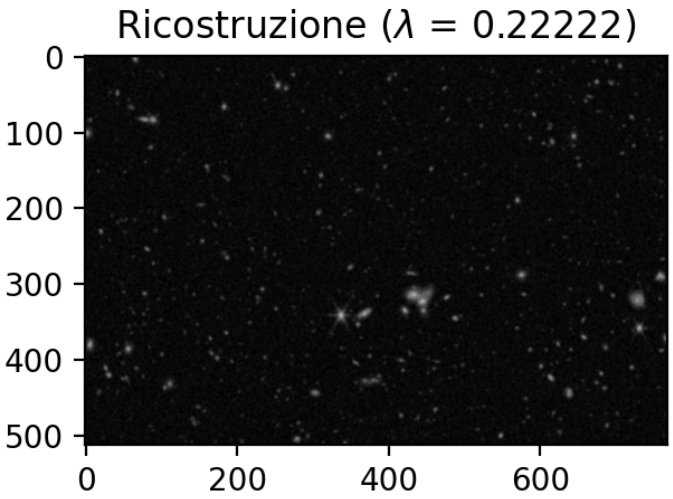
PSNR: 31.995 (λ = 0.17421)
MSE: 0.000632 (λ = 0.17421)
Discrepanza: 34.093 (43.345)



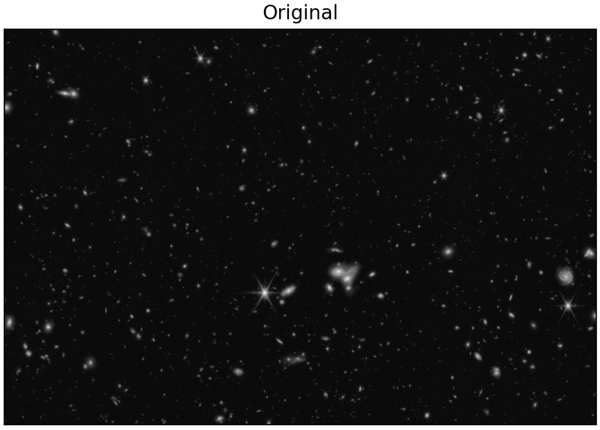
PSNR: 31.622 (λ = 0.21211)
MSE: 0.000688 (λ = 0.21211)
Discrepanza: 41.255 (43.345)

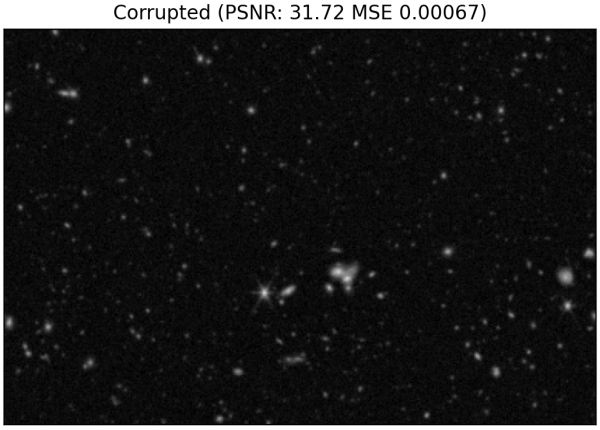
PSNR: 31.312 (λ = 0.22474)
MSE: 0.000739 (λ = 0.22474)
Discrepanza: 48.958 (43.345)





Test con k = *gaussian_kernel*(5, 1.3):





PSNR: 31.740 (λ = 0.13889)
MSE: 0.000670 (λ = 0.13889)
Discrepanza: 35.560 (43.345)

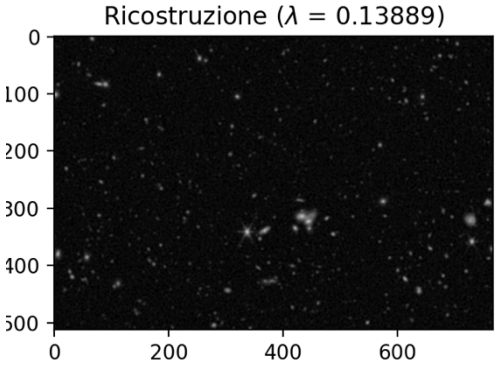
PSNR: 31.580 (λ = 0.15278)
MSE: 0.000695 (λ = 0.15278)
Discrepanza: 38.471 (43.345)

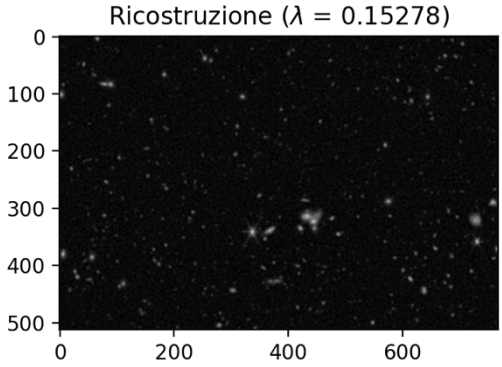
PSNR: 31.616 (λ = 0.16667)
MSE: 0.000689 (λ = 0.16667)
Discrepanza: 40.965 (43.345)

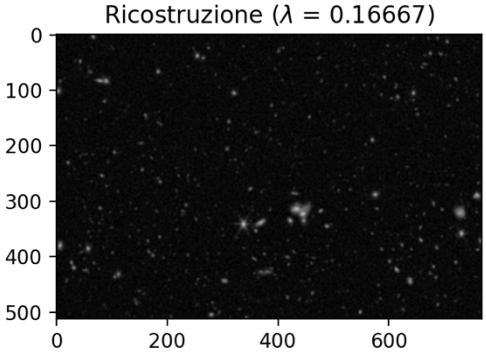
PSNR: 31.489 (λ = 0.18056)
MSE: 0.000710 (λ = 0.18056)
Discrepanza: 43.244 (43.345)

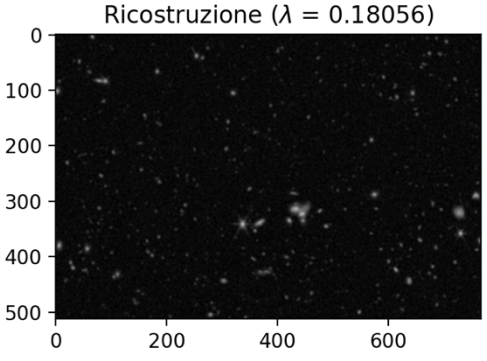
PSNR: 31.356 (λ = 0.19444)
MSE: 0.000732 (λ = 0.19444)
Discrepanza: 45.649 (43.345)

PSNR: 31.217 (λ = 0.20833)
MSE: 0.000756 (λ = 0.20833)
Discrepanza: 48.182 (43.345)

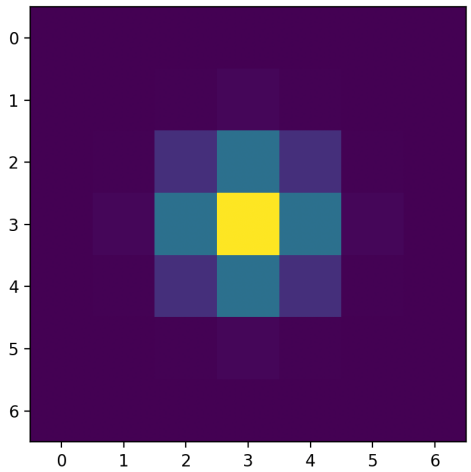




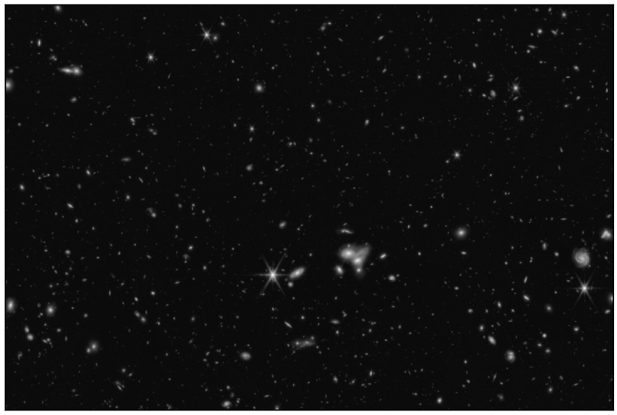




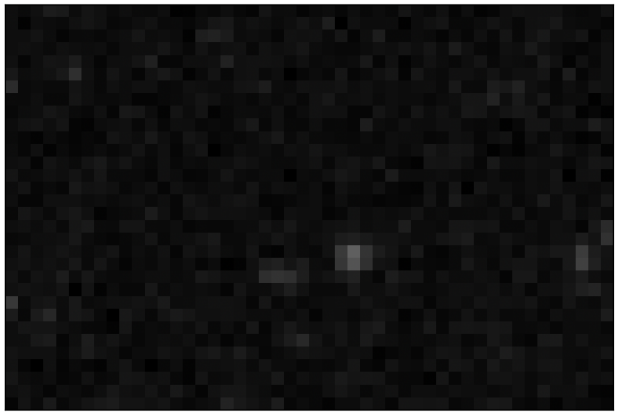
sf = 16, Test con foto presa tramite microscopio
Test con k = *gaussian_kernel* (7, 0.5):



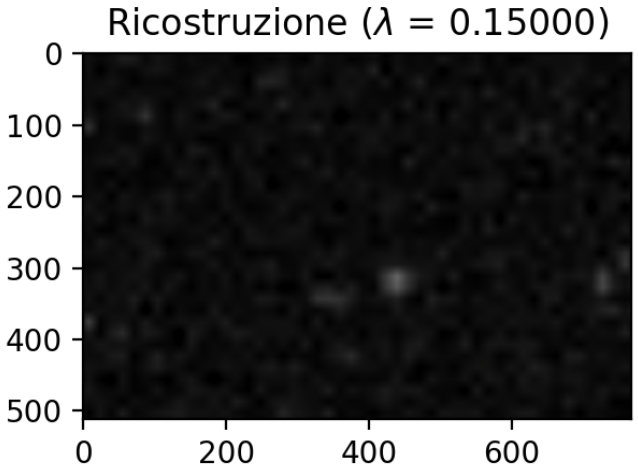
Original



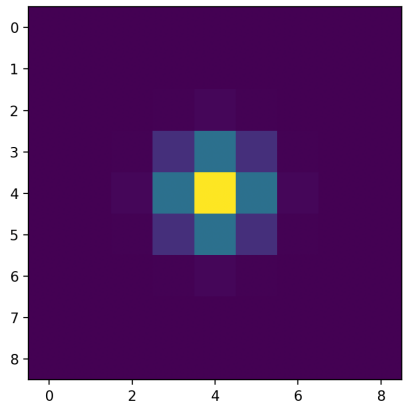
Corrupted (PSNR: 26.31 MSE 0.00234)

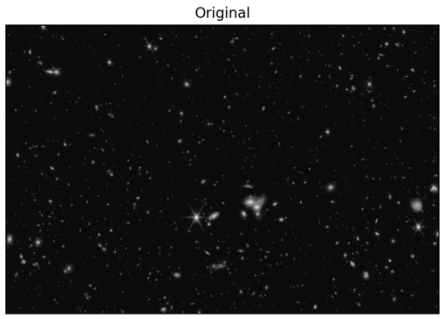


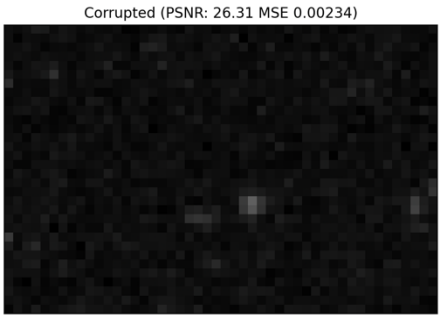
Ricostruzione migliore:
PSNR: 26.210 (λ = 0.15000)
MSE: 0.002393 (λ = 0.15000)
Discrepanza: 0.322 (0.664)



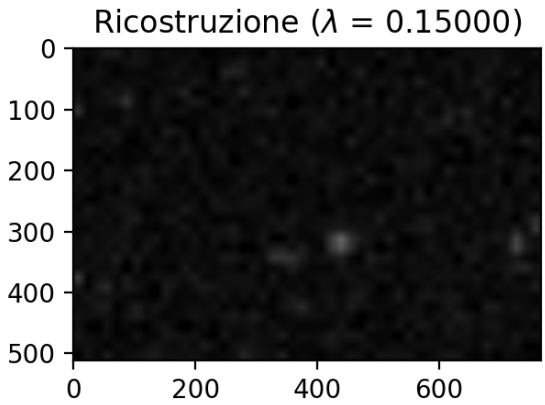
Test con k = *gaussian_kernel*(9, 0.5):



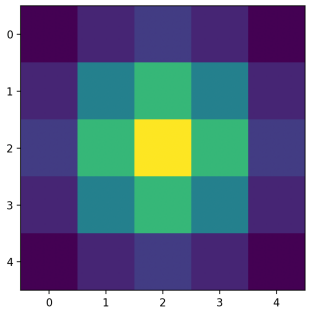


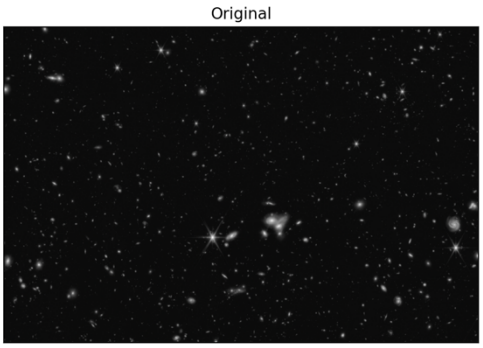


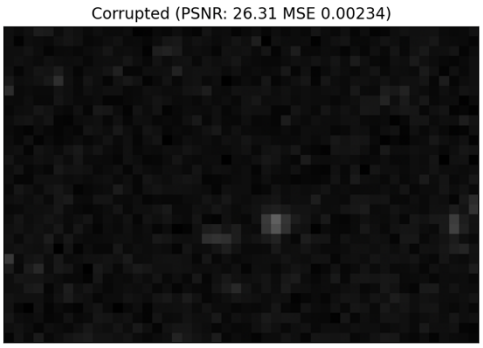
PSNR: 26.210 (λ = 0.15000)
MSE: 0.002393 (λ = 0.15000)
Discrepanza: 0.322 (0.664)



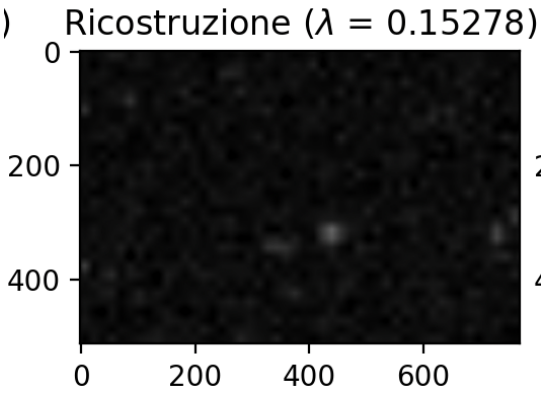
Test con k = *gaussian_kernel*(5, 1.3):



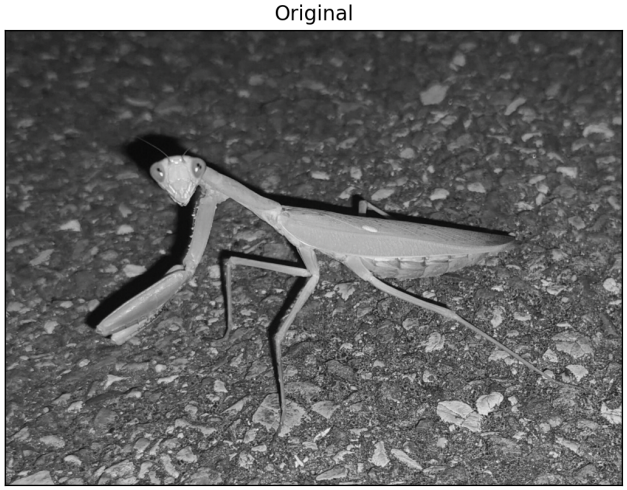


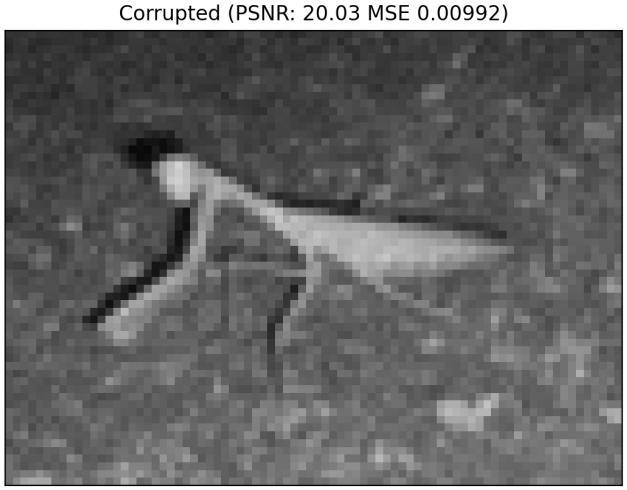


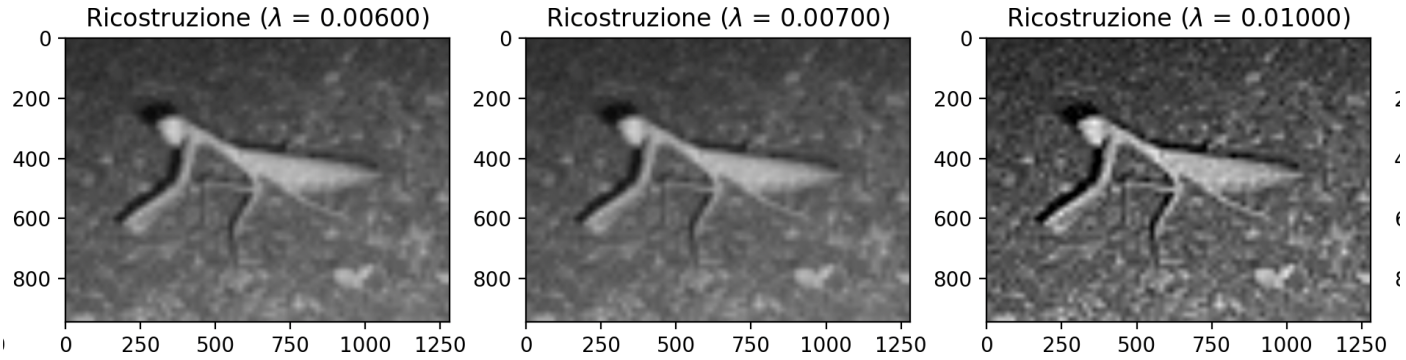
PSNR: 26.210 (λ = 0.15278)
MSE: 0.002393 (λ = 0.15278)
Discrepanza: 0.329 (0.664)



Test fotografico:
Test con k = *gaussian_kernel*(7, 0.5):



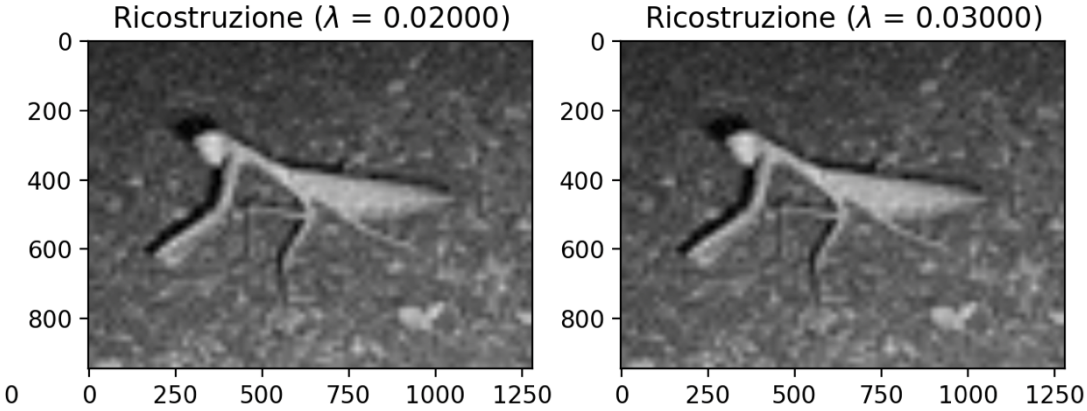




PSNR: 20.033 (λ = 0.00600)
MSE: 0.009924 (λ = 0.00600)
Discrepanza: 3.252 (2.055)

PSNR: 20.033 (λ = 0.00700)
MSE: 0.009924 (λ = 0.00700)
Discrepanza: 3.252 (2.055)

PSNR: 20.317 (λ = 0.01000)
MSE: 0.009295 (λ = 0.01000)
Discrepanza: 0.857 (2.055)



PSNR: 20.298 (λ = 0.02000)
MSE: 0.009336 (λ = 0.02000)
Discrepanza: 1.524 (2.055)

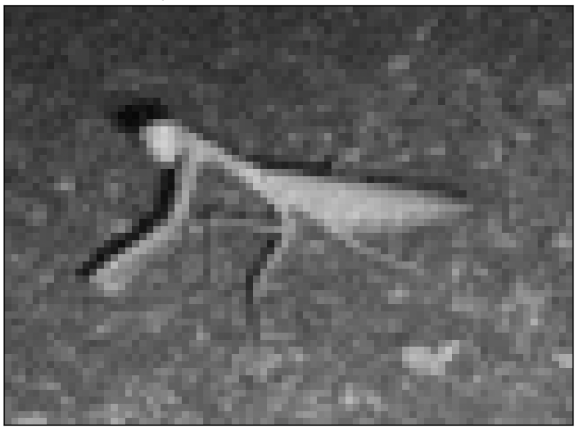
PSNR: 20.256 (λ = 0.03000)
MSE: 0.009427 (λ = 0.03000)
Discrepanza: 1.933 (2.055)

Test con k = *gaussian_kernel*(9, 0.5): analogo al precedente

Test con k = *gaussian_kernel*(5, 1.3):

Original

Corrupted (PSNR: 20.02 MSE 0.00996)



PSNR: 20.246 (λ = 0.03000)
MSE: 0.009449 (λ = 0.03000)
Discrepanza: 1.958 (2.055)