IN4640 Machine Vision

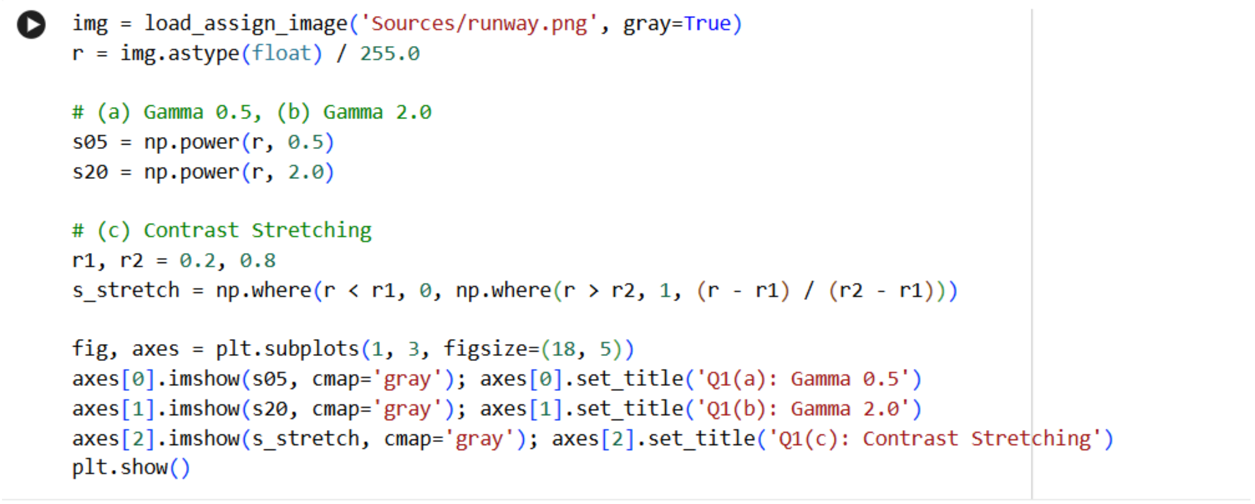
Assignment 1

Intensity Transformations and Neighborhood Filtering

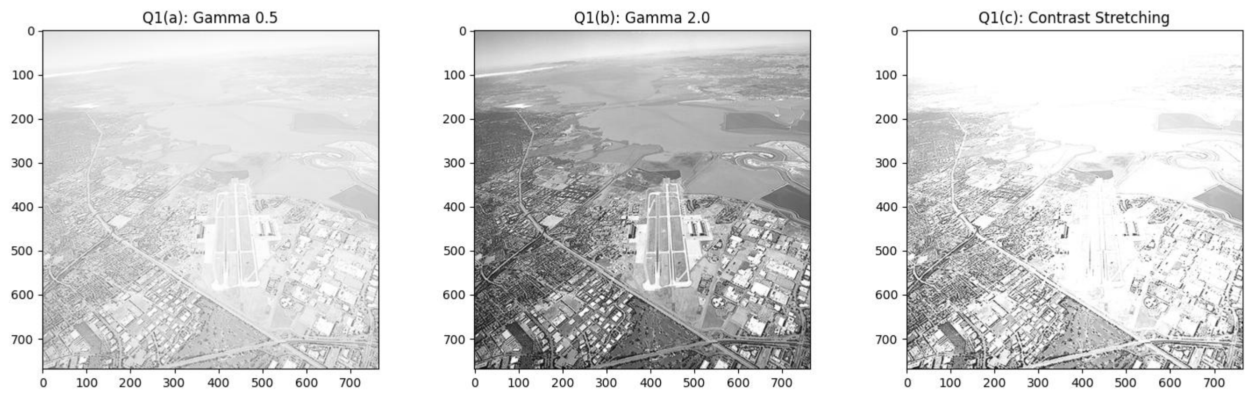
215519C

N D G S Dissasekara

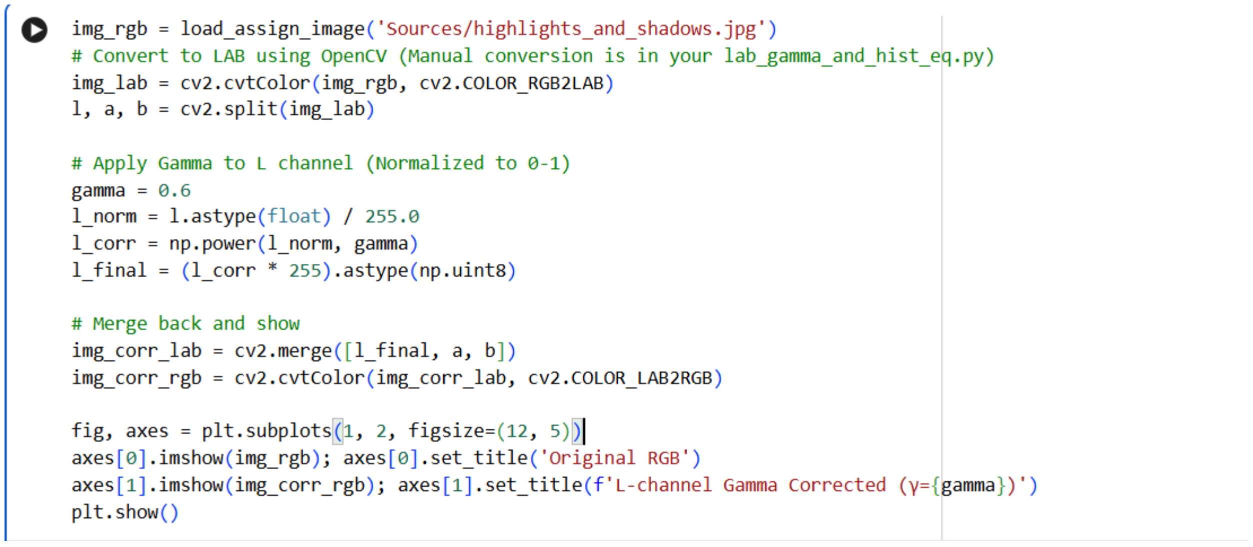
1. Apply the following intensity transform to the image



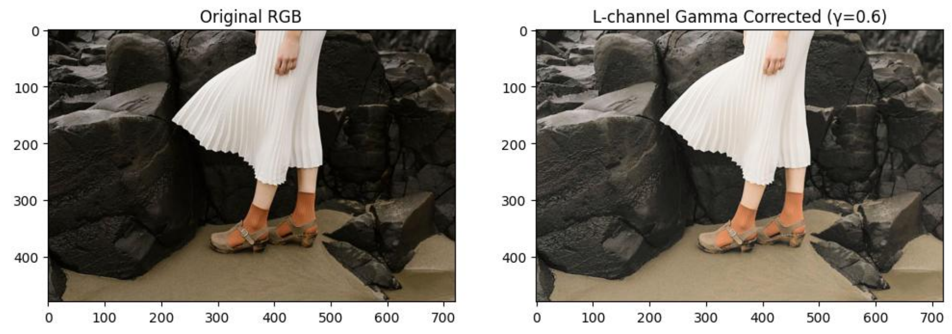
```python
img = load_assign_image('Sources/runway.png', gray=True)
r = img.astype(float) / 255.0

# (a) Gamma 0.5, (b) Gamma 2.0
s05 = np.power(r, 0.5)
s20 = np.power(r, 2.0)

# (c) Contrast Stretching
r1, r2 = 0.2, 0.8
s_stretch = np.where(r < r1, 0, np.where(r > r2, 1, (r - r1) / (r2 - r1)))

fig, axes = plt.subplots(1, 3, figsize=(18, 5))
axes[0].imshow(s05, cmap='gray'); axes[0].set_title('Q1(a): Gamma 0.5')
axes[1].imshow(s20, cmap='gray'); axes[1].set_title('Q1(b): Gamma 2.0')
axes[2].imshow(s_stretch, cmap='gray'); axes[2].set_title('Q1(c): Contrast Stretching')
plt.show()
```



2. Apply gamma correction

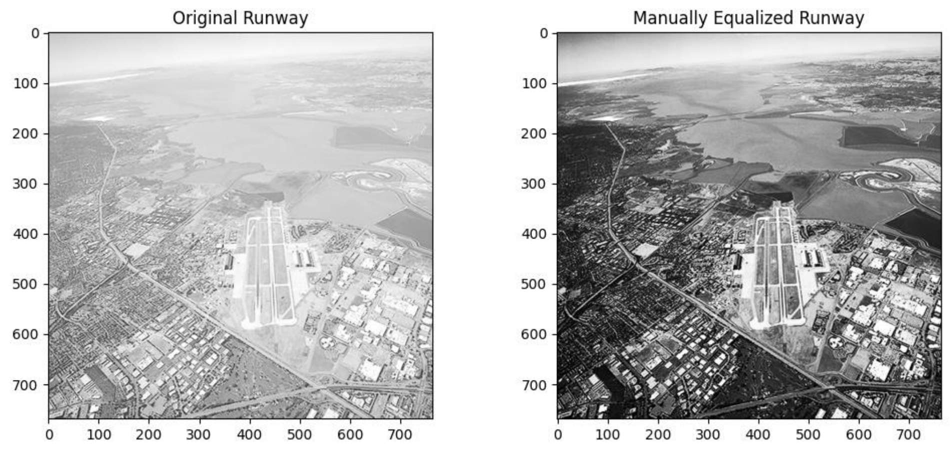


```python
img_rgb = load_assign_image('Sources/highlights_and_shadows.jpg')
# Convert to LAB using OpenCV (Manual conversion is in your lab_gamma_and_hist_eq.py)
img_lab = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2LAB)
l, a, b = cv2.split(img_lab)

# Apply Gamma to L channel (Normalized to 0-1)
gamma = 0.6
l_norm = l.astype(float) / 255.0
l_corr = np.power(l_norm, gamma)
l_final = (l_corr * 255).astype(np.uint8)

# Merge back and show
img_corr_lab = cv2.merge([l_final, a, b])
img_corr_rgb = cv2.cvtColor(img_corr_lab, cv2.COLOR_LAB2RGB)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].imshow(img_rgb); axes[0].set_title('Original RGB')
axes[1].imshow(img_corr_rgb); axes[1].set_title(f'L-channel Gamma Corrected (γ={gamma})')
plt.show()
```



3. Write your own function to equalize the histogram of an image

```
img_gray = load_assign_image('Sources/runway.png', gray=True)

# Manual Histogram Equalization logic
hist, _ = np.histogram(img_gray.flatten(), 256, [0, 256])
cdf = hist.cumsum()
cdf_m = np.ma.masked_equal(cdf, 0)
cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
cdf_final = np.ma.filled(cdf_m, 0).astype(np.uint8)
img_eq = cdf_final[img_gray]

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1); plt.imshow(img_gray, cmap='gray'); plt.title('Original Runway')
plt.subplot(1, 2, 2); plt.imshow(img_eq, cmap='gray'); plt.title('Manually Equalized Runway')
plt.show()
```

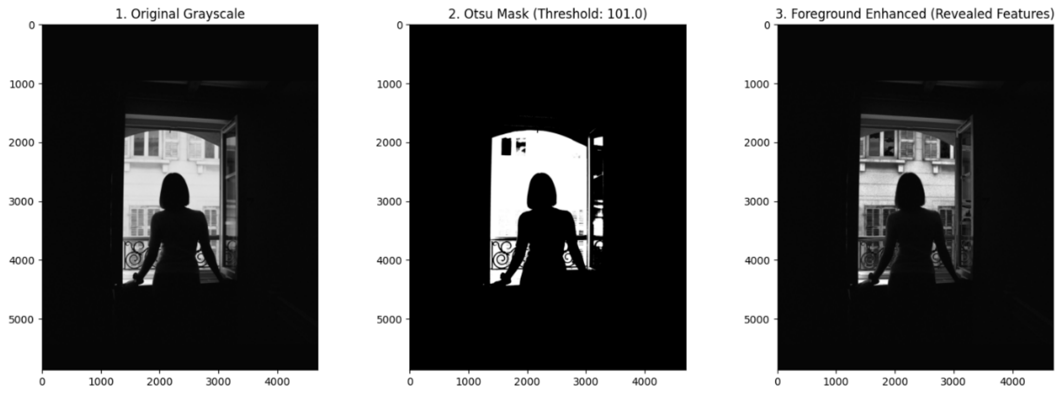


4. Otsu thresholding

```
img_q4 = load_assign_image('Sources/looking_out.jpg', gray=True)

# (a) Otsu Thresholding
val, mask = cv2.threshold(img_q4, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
print(f"Otsu Threshold Value: {val}")

# (b) Equalize only the foreground (where mask is white)
res = img_q4.copy()
m_pixels = img_q4[mask == 255]
h_m, _ = np.histogram(m_pixels, 256, [0, 256])
c_m = h_m.cumsum()
c_norm = ((c_m - c_m.min()) * 255 / (m_pixels.size - c_m.min())).astype(np.uint8)
res[mask == 255] = c_norm[img_q4[mask == 255]]

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1); plt.imshow(mask, cmap='gray'); plt.title("Otsu Binary Mask")
plt.subplot(1, 2, 2); plt.imshow(res, cmap='gray'); plt.title("Masked Region Equalized")
plt.show()
```

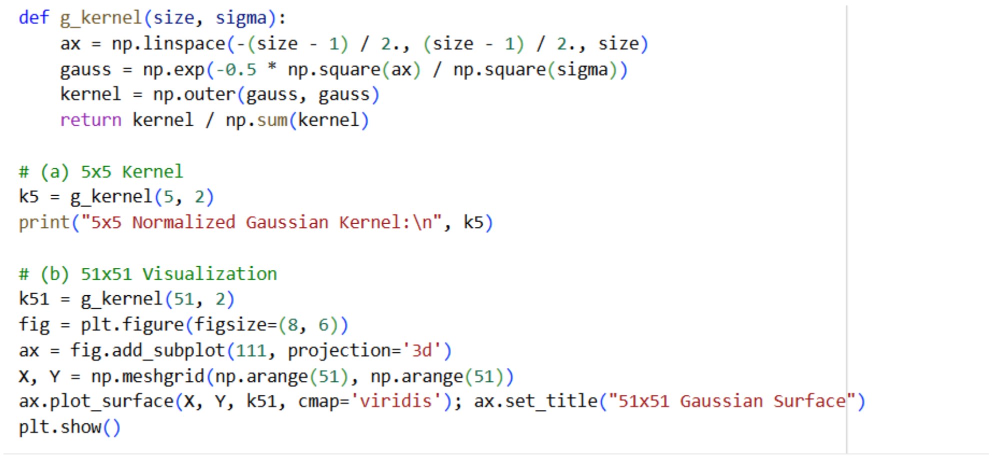
a. Otsu Threshold Value = 101.0

b. Features

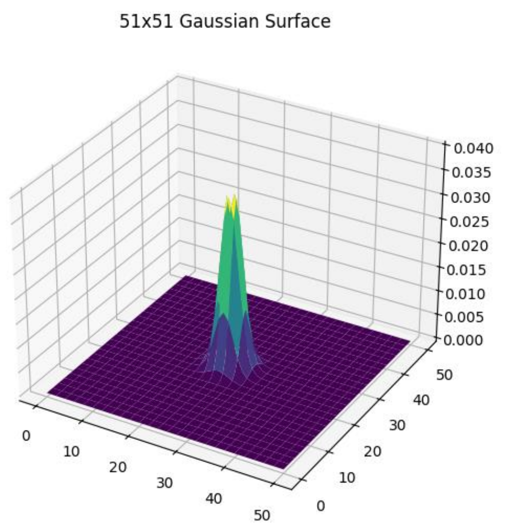

1. Original Grayscale
2. Otsu Mask (Threshold: 101.0)
3. Foreground Enhanced (Revealed Features)
0
1000
2000
3000
4000
5000

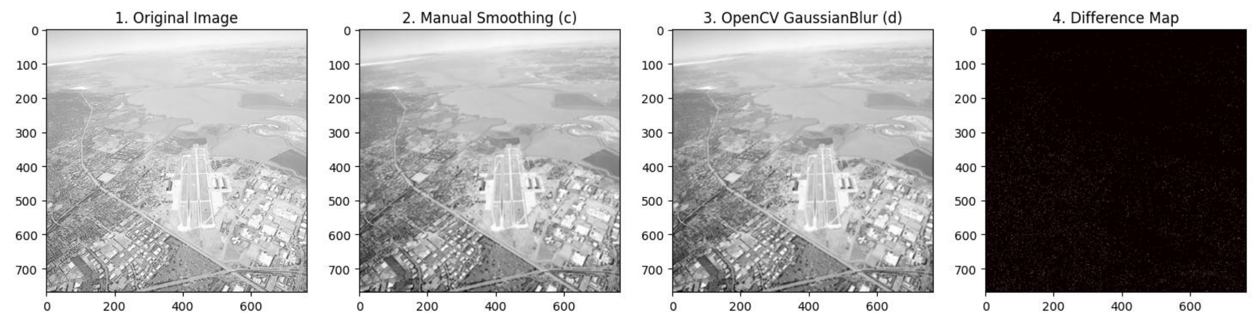
- Clothing Details -- Previously invisible fabric textures and folds (e.g., knit vs. smooth material) become visible.
- Hair Volume -- Highlights and individual strands appear, giving the hair realistic 3D depth.
- Room Interior -- Dark areas reveal wall textures, window frames, and surface details.
- Metalwork Details -- Fine features of railings or grills—joints, screws, and wear—become clear.
- Body Posture -- Shoulder curves, arm shape, and stance are now distinguishable.
- Floor & Furniture -- Hidden floor textures and unnoticed furniture elements emerge.

5. Gaussian filtering



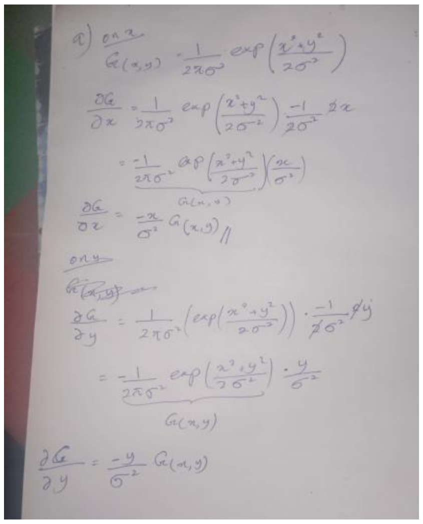
```
def g_kernel(size, sigma):
    ax = np.linspace(-(size - 1) / 2., (size - 1) / 2., size)
    gauss = np.exp(-0.5 * np.square(ax) / np.square(sigma))
    kernel = np.outer(gauss, gauss)
    return kernel / np.sum(kernel)

# (a) 5x5 Kernel
k5 = g_kernel(5, 2)
print("5x5 Normalized Gaussian Kernel:\n", k5)

# (b) 51x51 Visualization
k51 = g_kernel(51, 2)
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
X, Y = np.meshgrid(np.arange(51), np.arange(51))
ax.plot_surface(X, Y, k51, cmap='viridis'); ax.set_title("51x51 Gaussian Surface")
plt.show()
```





6. Derivative of Gaussian

a.



a) on x

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{x^2+y^2}{2\sigma^2}\right)$$

$$\frac{\partial G}{\partial x} = \frac{1}{2\pi\sigma^2} \exp\left(\frac{x^2+y^2}{2\sigma^2}\right) \frac{-1}{\cancel{2}\sigma^2} \cancel{2}x$$

$$= \underbrace{\frac{-1}{2\pi\sigma^2} \exp\left(\frac{x^2+y^2}{2\sigma^2}\right)}_{G(x,y)} \left(\frac{x}{\sigma^2}\right)$$

$$\frac{\partial G}{\partial x} = \frac{-x}{\sigma^2} G(x,y) //$$

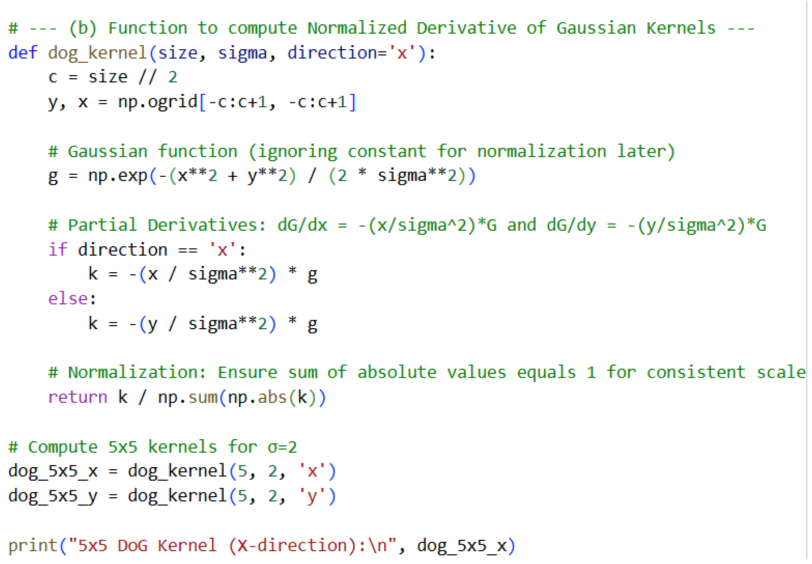
on y

~~$G(x,y)$~~

$$\frac{\partial G}{\partial y} = \frac{1}{2\pi\sigma^2}\left(\exp\left(\frac{x^2+y^2}{2\sigma^2}\right)\right) \cdot \frac{-1}{\cancel{2}\sigma^2} \cancel{2}y$$

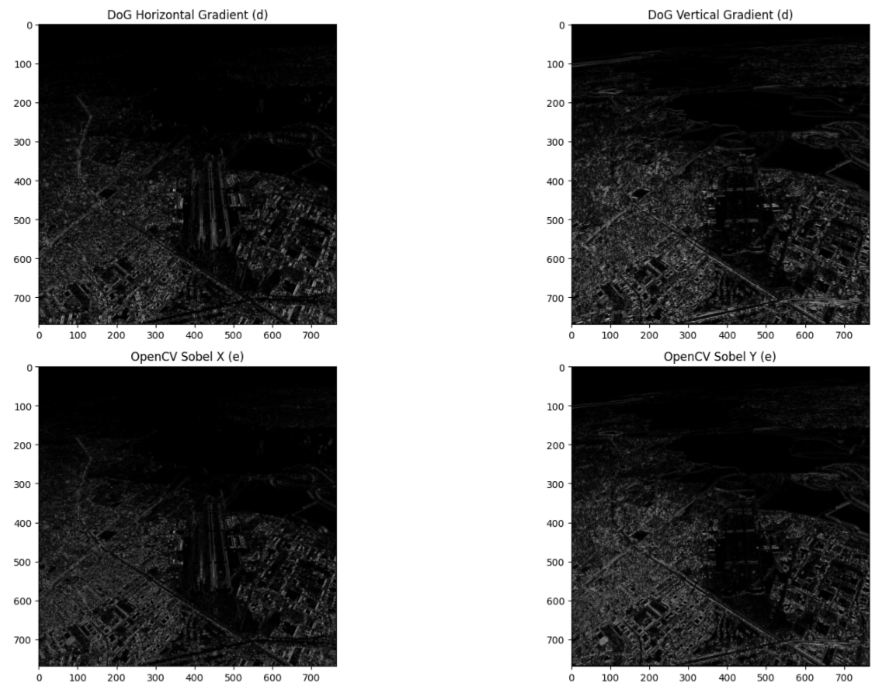
$$= \underbrace{\frac{-1}{2\pi\sigma^2} \exp\left(\frac{x^2+y^2}{2\sigma^2}\right)}_{G(x,y)} \cdot \frac{y}{\sigma^2}$$

$$\frac{\partial G}{\partial y} = \frac{-y}{\sigma^2} G(x,y)$$



```python
# --- (b) Function to compute Normalized Derivative of Gaussian Kernels ---
def dog_kernel(size, sigma, direction='x'):
    c = size // 2
    y, x = np.ogrid[-c:c+1, -c:c+1]

    # Gaussian function (ignoring constant for normalization later)
    g = np.exp(-(x**2 + y**2) / (2 * sigma**2))

    # Partial Derivatives: dG/dx = -(x/sigma^2)*G and dG/dy = -(y/sigma^2)*G
    if direction == 'x':
        k = -(x / sigma**2) * g
    else:
        k = -(y / sigma**2) * g

    # Normalization: Ensure sum of absolute values equals 1 for consistent scale
    return k / np.sum(np.abs(k))

# Compute 5x5 kernels for σ=2
dog_5x5_x = dog_kernel(5, 2, 'x')
dog_5x5_y = dog_kernel(5, 2, 'y')

print("5x5 DoG Kernel (X-direction):\n", dog_5x5_x)
```

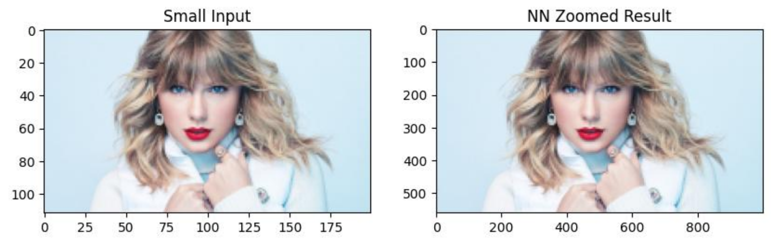
b.



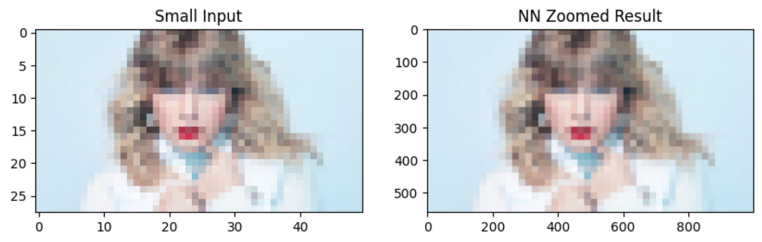
7. Zoom

a. nearest neighbor

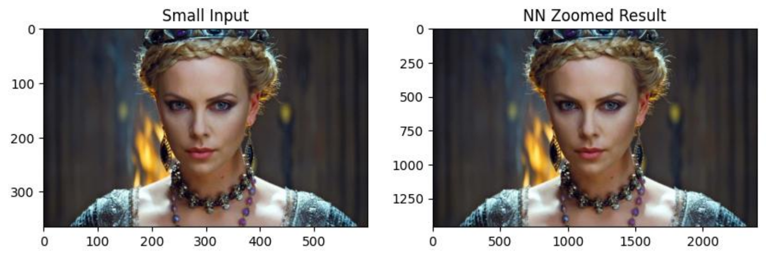
Scale: 5.00x | SSD (NN): 238.03



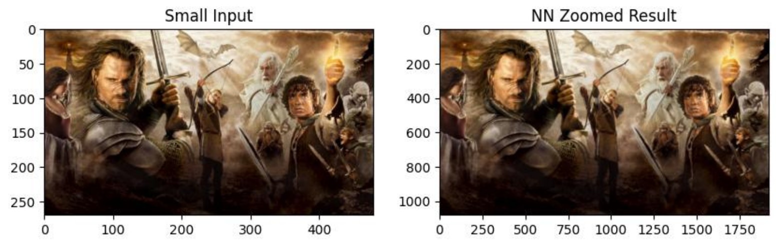
Scale: 20.00x | SSD (NN): 485.89



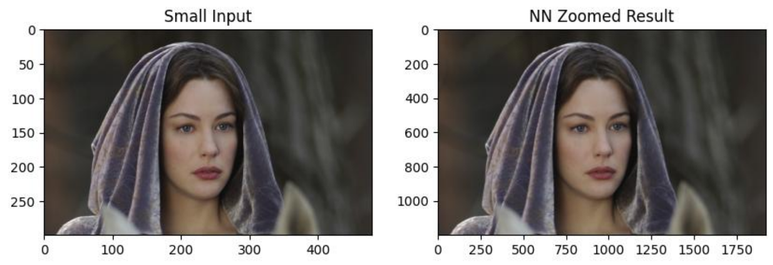
Scale: 4.00x | SSD (NN): 73.17



Scale: 4.00x | SSD (NN): 136.27

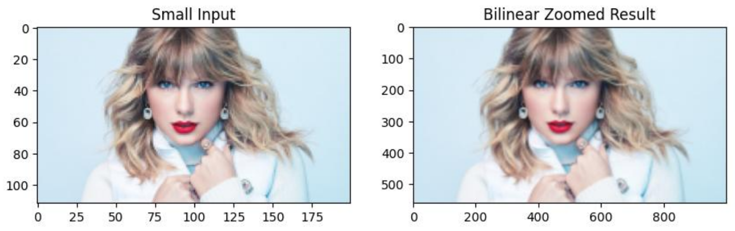


Scale: 4.00x | SSD (NN): 26.45

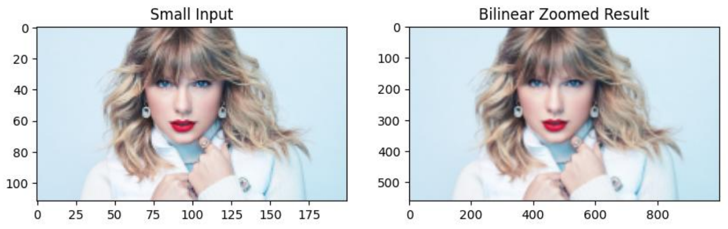


b. bilinear interpolation.

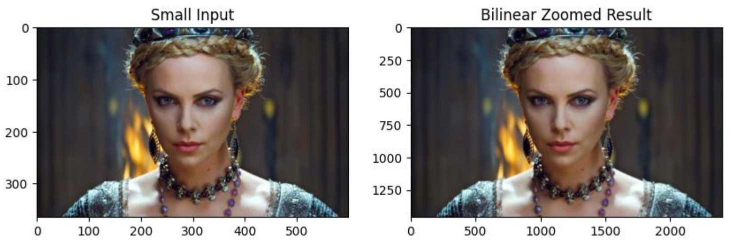
Scale: 5.00x | SSD (Bilinear): 285.14



Scale: 20.00x | SSD (Bilinear): 654.60

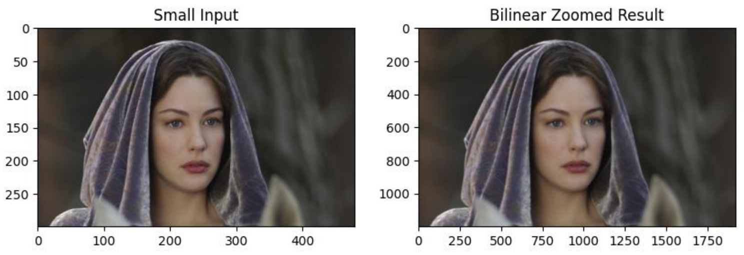


Scale: 4.00x | SSD (Bilinear): 138.78

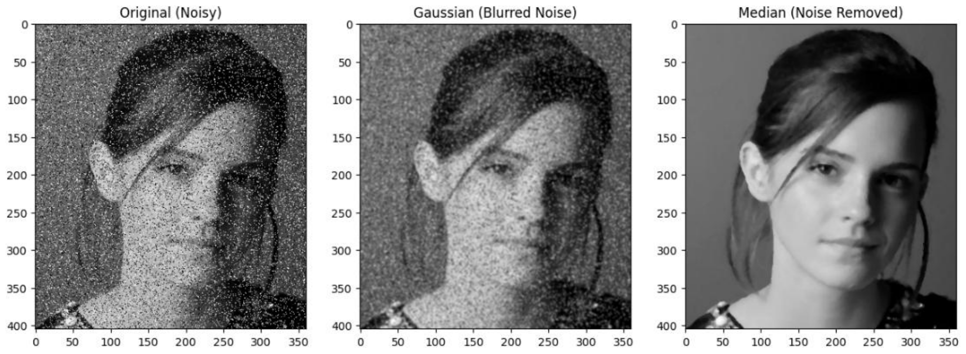


Scale: 4.00x | SSD (Bilinear): 200.25

Scale: 4.00x | SSD (Bilinear): 48.96

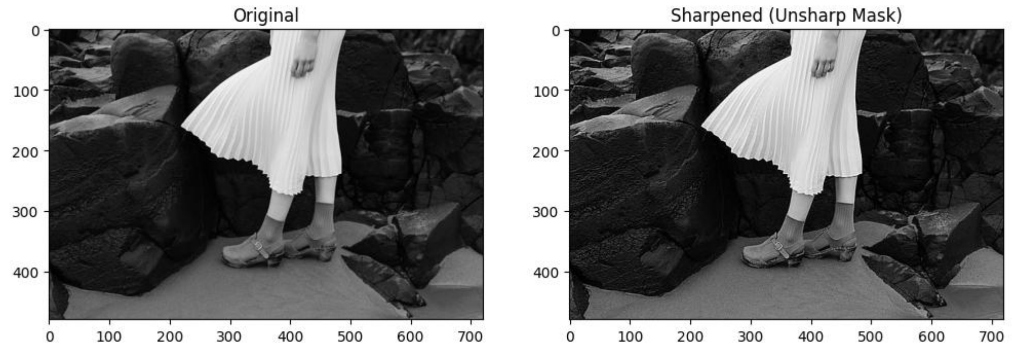


8. Salt and paper noise

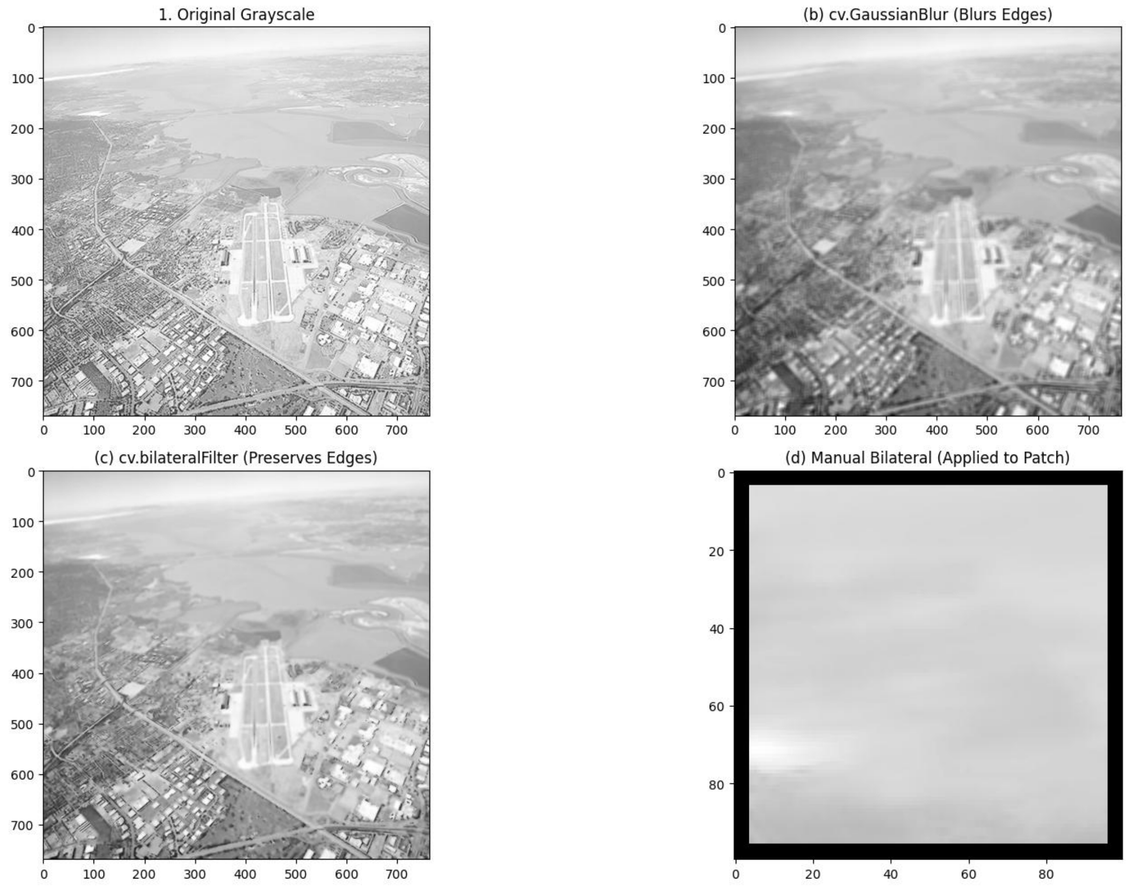


9. Sharpening

```
img_q9 = load_assign_image('Sources/runway.png', gray=True)

# Unsharp Masking: Sharpened = Original + strength * (Original - Blurred)
blurred = cv2.GaussianBlur(img_q9, (5, 5), 1.0)
detail = img_q9.astype(float) - blurred.astype(float)
sharpened = np.clip(img_q9.astype(float) + 1.5 * detail, 0, 255).astype(np.uint8)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1); plt.imshow(img_q9, cmap='gray'); plt.title("Original")
plt.subplot(1, 2, 2); plt.imshow(sharpened, cmap='gray'); plt.title("Sharpened (Unsharp Mask)")
plt.show()
```



10. Bilateral filtering


1. Original Grayscale
(b) cv.GaussianBlur (Blurs Edges)
(c) cv.bilateralFilter (Preserves Edges)
(d) Manual Bilateral (Applied to Patch)