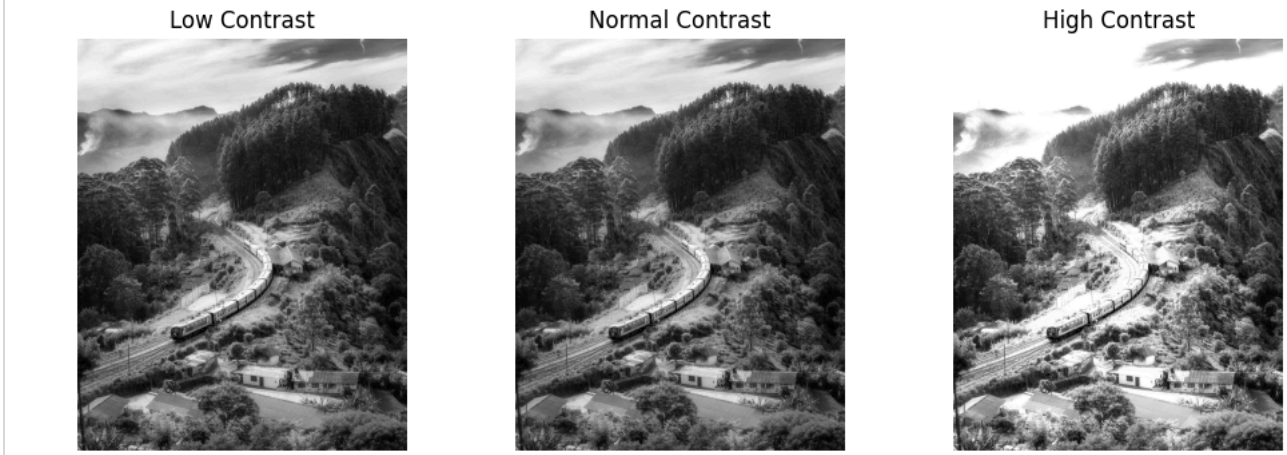
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load the image in grayscale
img = cv2.imread('/content/ref.jpg', cv2.IMREAD_GRAYSCALE)

if img is None:
    raise FileNotFoundError("Please provide a valid image path!")

plt.figure(figsize=(4,4))
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')
plt.show()
```


```
def adjust_contrast(image, level):
    """Adjust image contrast: low, normal, or high."""
    if level == 'low':
        return cv2.convertScaleAbs(image, alpha=0.5, beta=0)
    elif level == 'high':
        return cv2.convertScaleAbs(image, alpha=1.5, beta=0)
    return image

# Generate contrast versions
low_contrast = adjust_contrast(img, 'low')
normal_contrast = adjust_contrast(img, 'normal')
high_contrast = adjust_contrast(img, 'high')

# Display all contrasts
plt.figure(figsize=(12,4))
for i, (title, im) in enumerate(zip(
    ['Low Contrast', 'Normal Contrast', 'High Contrast'],
    [low_contrast, normal_contrast, high_contrast]
)):
    plt.subplot(1,3,i+1)
    plt.imshow(im, cmap='gray')
    plt.title(title)
    plt.axis('off')
plt.show()
```




```
def dft_image(image):
    """Perform DFT and shift the spectrum."""
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    return fshift

def idft_image(fshift):
    """Perform inverse DFT."""
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    return np.abs(img_back)

def distance_matrix(shape):
    """Compute distance matrix from center."""
    rows, cols = shape
    crow, ccol = rows//2, cols//2
    u = np.arange(rows)
    v = np.arange(cols)
    U, V = np.meshgrid(u, v, indexing='ij')
    D = np.sqrt((U - crow)**2 + (V - ccol)**2)
    return D
```

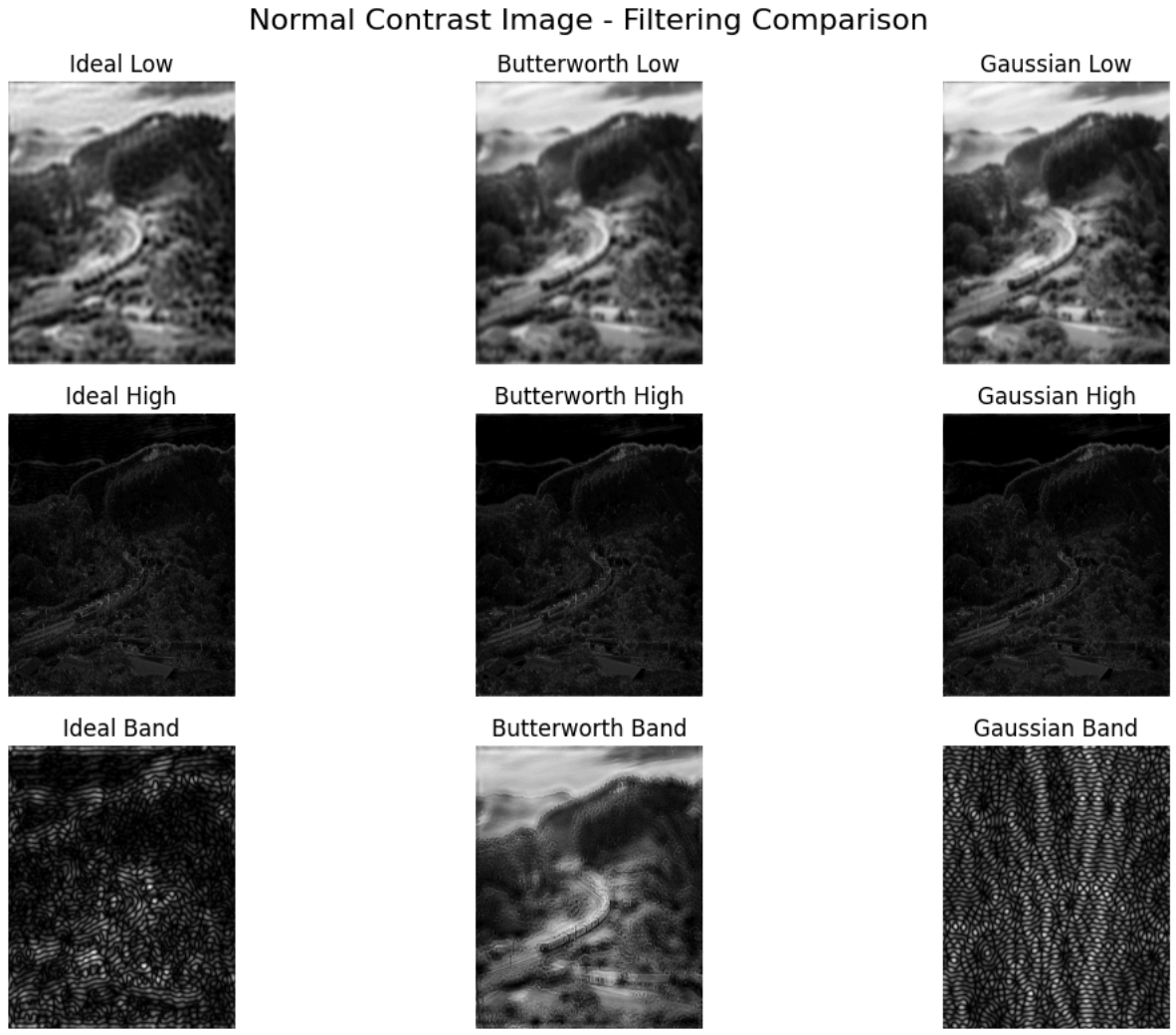
```
def ideal_filter(shape, D0, type='low', W=20):
    D = distance_matrix(shape)
    if type == 'low':
        H = (D <= D0).astype(float)
    elif type == 'high':
        H = (D > D0).astype(float)
    elif type == 'band':
        H = np.logical_and(D >= D0 - W/2, D <= D0 + W/2).astype(float)
    return H

def butterworth_filter(shape, D0, n=2, type='low', W=20):
    D = distance_matrix(shape)
    if type == 'low':
        H = 1 / (1 + (D / D0)**(2*n))
    elif type == 'high':
        H = 1 / (1 + (D0 / D)**(2*n))
    elif type == 'band':
        H = 1 / (1 + ((D*W)/(D**2 - D0**2))**(2*n))
    return H

def gaussian_filter(shape, D0, type='low', W=20):
    D = distance_matrix(shape)
    if type == 'low':
        H = np.exp(-(D**2) / (2 * (D0**2)))
    elif type == 'high':
        H = 1 - np.exp(-(D**2) / (2 * (D0**2)))
    elif type == 'band':
        H = np.exp(-((D**2 - D0**2)**2) / (2 * (W**2)))
    return H

def apply_filter(image, H):
    """Apply a filter mask in the frequency domain."""
    fshift = dft_image(image)
```

```
    filtered = fshift * H
    return idft_image(filtered)
```

```
D0 = 30   # Cutoff frequency
n = 2     # Butterworth order
filter_types = ['low', 'high', 'band']
```

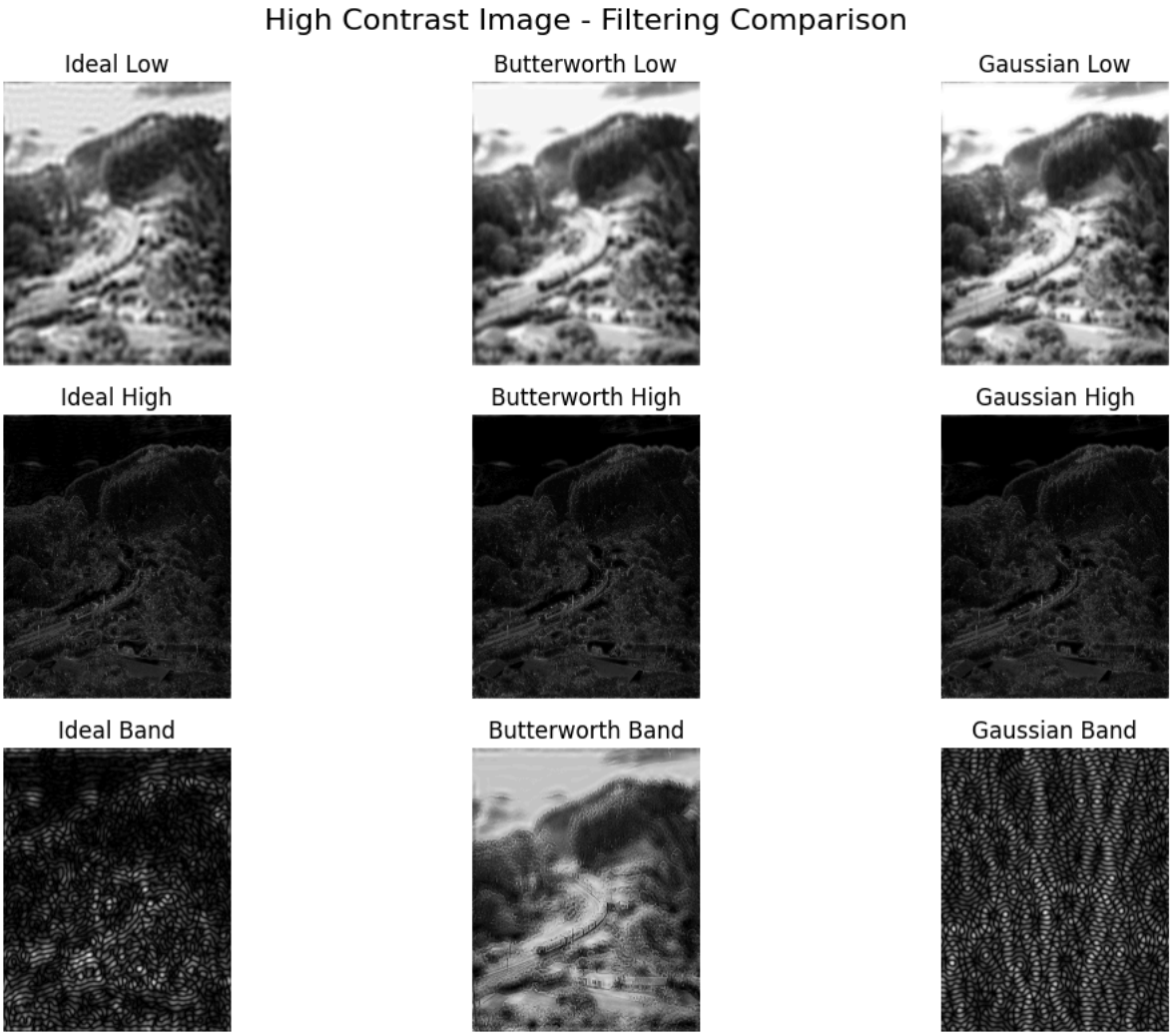
```
image = normal_contrast
plt.figure(figsize=(12, 8))
plt.suptitle("Normal Contrast Image - Filtering Comparison", fontsize=16)

for i, ftype in enumerate(filter_types):
    ideal = apply_filter(image, ideal_filter(image.shape, D0, ftype))
    butter = apply_filter(image, butterworth_filter(image.shape, D0, n=n, type=ftype))
    gauss = apply_filter(image, gaussian_filter(image.shape, D0, type=ftype))

    plt.subplot(3, 3, i*3 + 1)
    plt.imshow(ideal, cmap='gray')
    plt.title(f"Ideal {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 2)
    plt.imshow(butter, cmap='gray')
    plt.title(f"Butterworth {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 3)
    plt.imshow(gauss, cmap='gray')
    plt.title(f"Gaussian {ftype.capitalize()}")
    plt.axis('off')

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

```
/tmp/ipython-input-1052064311.py:16: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + (D0 / D)**(2*n))
/tmp/ipython-input-1052064311.py:18: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + ((D*W)/(D**2 - D0**2))**(2*n))
```




```
image = high_contrast
plt.figure(figsize=(12, 8))
plt.suptitle("High Contrast Image - Filtering Comparison", fontsize=16)

for i, ftype in enumerate(filter_types):
    ideal = apply_filter(image, ideal_filter(image.shape, D0, ftype))
    butter = apply_filter(image, butterworth_filter(image.shape, D0, n=n, type=ftype))
    gauss = apply_filter(image, gaussian_filter(image.shape, D0, type=ftype))

    plt.subplot(3, 3, i*3 + 1)
    plt.imshow(ideal, cmap='gray')
    plt.title(f"Ideal {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 2)
    plt.imshow(butter, cmap='gray')
    plt.title(f"Butterworth {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 3)
    plt.imshow(gauss, cmap='gray')
    plt.title(f"Gaussian {ftype.capitalize()}")
    plt.axis('off')

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

```
/tmp/ipython-input-1052064311.py:16: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + (D0 / D)**(2*n))
/tmp/ipython-input-1052064311.py:18: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + ((D*W)/(D**2 - D0**2))**(2*n))
```




```
image = low_contrast
plt.figure(figsize=(12, 8))
plt.suptitle("Low Contrast Image - Filtering Comparison", fontsize=16)

for i, ftype in enumerate(filter_types):
    ideal = apply_filter(image, ideal_filter(image.shape, D0, ftype))
    butter = apply_filter(image, butterworth_filter(image.shape, D0, n=n, type=ftype))
    gauss = apply_filter(image, gaussian_filter(image.shape, D0, type=ftype))

    plt.subplot(3, 3, i*3 + 1)
    plt.imshow(ideal, cmap='gray')
    plt.title(f"Ideal {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 2)
    plt.imshow(butter, cmap='gray')
    plt.title(f"Butterworth {ftype.capitalize()}")
    plt.axis('off')

    plt.subplot(3, 3, i*3 + 3)
    plt.imshow(gauss, cmap='gray')
    plt.title(f"Gaussian {ftype.capitalize()}")
    plt.axis('off')

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

```
/tmp/ipython-input-1052064311.py:16: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + (D0 / D)**(2*n))
/tmp/ipython-input-1052064311.py:18: RuntimeWarning: divide by zero encountered in divide
  H = 1 / (1 + ((D*W)/(D**2 - D0**2))**(2*n))
```