1. Decompose the image using the Fourier Transform: Transform the image from the spatial domain to the frequency domain.
2. Apply the following filtering techniques:

a. Ideal Low Pass Filter

b. Ideal High Pass Filter

c. Butterworth Low Pass Filter

d. Butterworth High Pass Filter

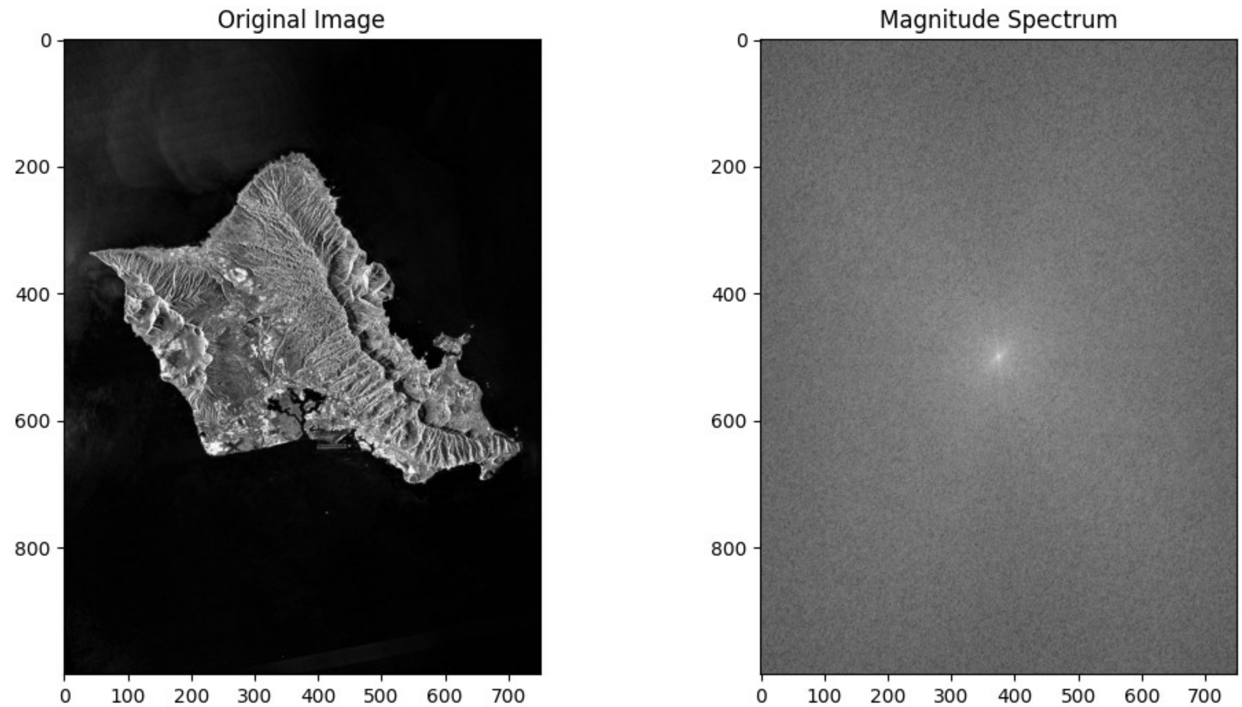
e. Gaussian Low Pass Filter

f. Gaussian High Pass Filter

1. Analyze with visualization proof, the impact of filter radius: Vary the cutoff radius for each filter type and observe changes in image quality and frequency representation.

Decompose the image using Fourier Transform

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

image = cv2.imread('/content/large_Radar Constellation - Hawaiian
Islands USA - Acquisition  Modes.jpg', 0)

# Perform Fourier Transform and shift zero frequency component to the
center
dft = np.fft.fft2(image)
dft_shift = np.fft.fftshift(dft)

# Get the magnitude spectrum for visualization
magnitude_spectrum = 20 * np.log(np.abs(dft_shift))

# Visualize the original image and its magnitude spectrum
plt.figure(figsize=(12,6))
plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Original
Image')
plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray'),
plt.title('Magnitude Spectrum')
plt.show()
```




```
Original Image:
This is the unprocessed satellite image in the spatial domain,
containing both
 low-frequency (broad areas) and high-frequency (fine details)
components.

Magnitude Spectrum:
The Fourier Transform of the image, displaying the frequency content.
The center of the spectrum represents low-frequency components,
 while the outer parts represent high frequencies. A bright center
spot indicates strong low-frequency content.
```

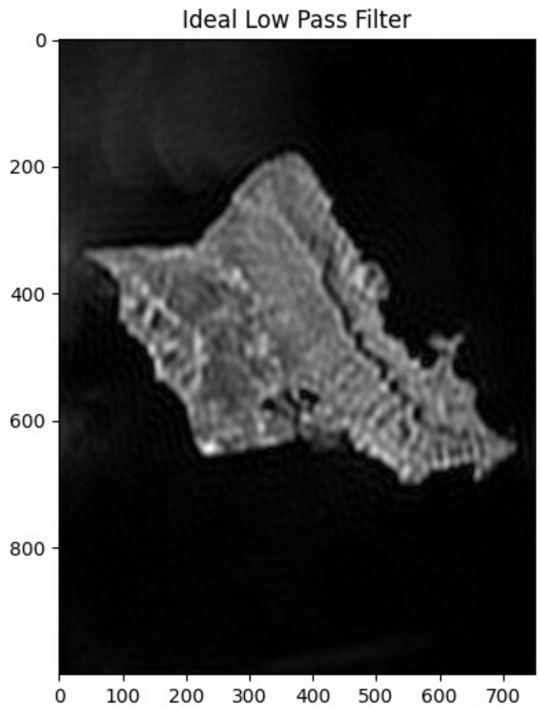
Define Filters

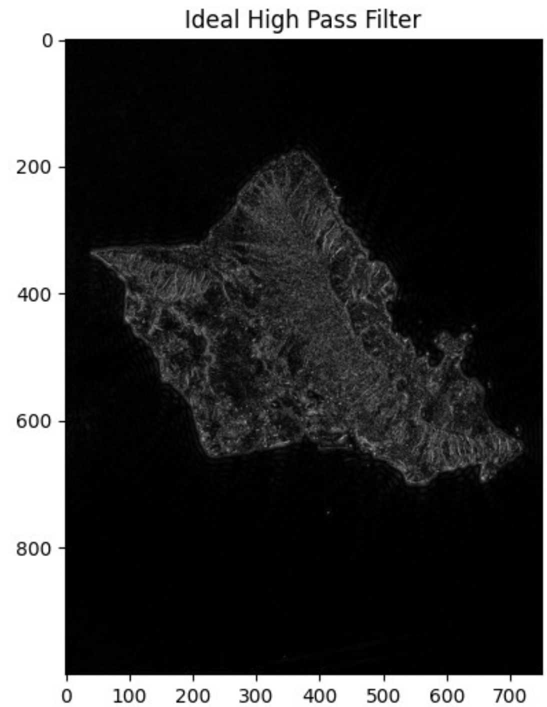
```python
def ideal_low_pass_filter(shape, radius):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), np.uint8)
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - crow)**2 + (j - ccol)**2) <= radius:
                mask[i, j] = 1
    return mask

def ideal_high_pass_filter(shape, radius):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
```

```python
    mask = np.ones((rows, cols), np.uint8)
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - crow)**2 + (j - ccol)**2) <= radius:
                mask[i, j] = 0
    return mask

def butterworth_low_pass_filter(shape, radius, order):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - crow)**2 + (j - ccol)**2)
            mask[i, j] = 1 / (1 + (distance / radius)**(2 * order))
    return mask

def butterworth_high_pass_filter(shape, radius, order):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    mask = np.ones((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - crow)**2 + (j - ccol)**2)
            mask[i, j] = 1 / (1 + (radius / distance)**(2 * order))
    return mask

def gaussian_low_pass_filter(shape, radius):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - crow)**2 + (j - ccol)**2)
            mask[i, j] = np.exp(-(distance**2) / (2 * (radius**2)))
    return mask

def gaussian_high_pass_filter(shape, radius):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    mask = np.ones((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - crow)**2 + (j - ccol)**2)
            mask[i, j] = 1 - np.exp(-(distance**2) / (2 *
(radius**2)))
    return mask
```

Apply the Filters and Visualize the Results

```python
def apply_filter(dft_shift, filter_mask):
    # Apply the filter mask to the shifted DFT
    filtered_dft_shift = dft_shift * filter_mask

    # Shift back the zero frequency component to its original position
    f_ishift = np.fft.ifftshift(filtered_dft_shift)

    # Perform inverse FFT to get the image back
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)

    return img_back

# Define the cutoff radius
radius = 50

# Apply Ideal Low Pass Filter
ideal_lp_mask = ideal_low_pass_filter(image.shape, radius)
ideal_lp_filtered_image = apply_filter(dft_shift, ideal_lp_mask)

# Apply Ideal High Pass Filter
ideal_hp_mask = ideal_high_pass_filter(image.shape, radius)
ideal_hp_filtered_image = apply_filter(dft_shift, ideal_hp_mask)

plt.figure(figsize=(12,6))
plt.subplot(121), plt.imshow(ideal_lp_filtered_image, cmap='gray'),
plt.title('Ideal Low Pass Filter')
plt.subplot(122), plt.imshow(ideal_hp_filtered_image, cmap='gray'),
plt.title('Ideal High Pass Filter')
plt.show()
```





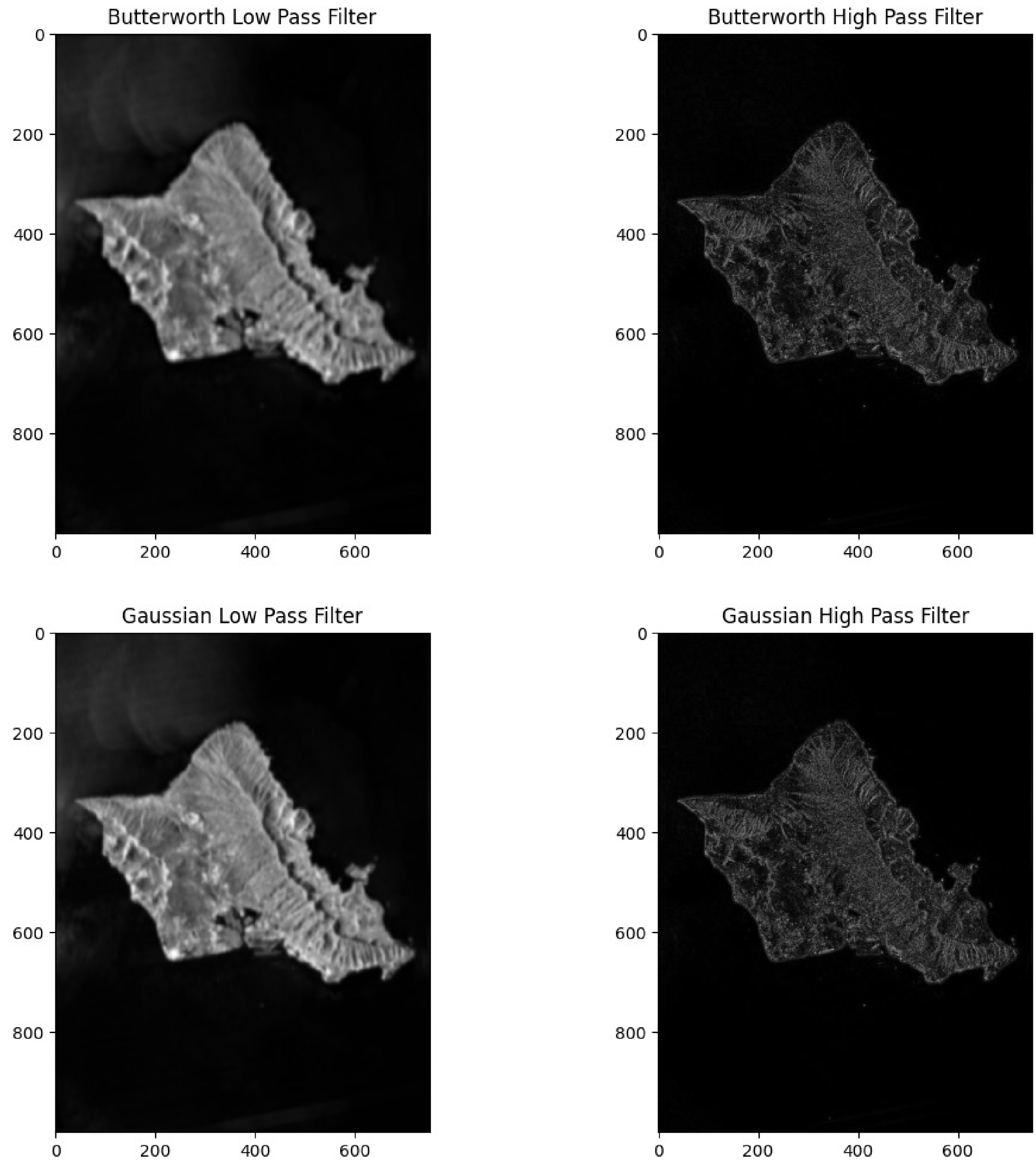

```
Ideal Low Pass Filter:
This filter passes low-frequency components and blocks high
frequencies. The image appears blurred,
emphasizing broader shapes while eliminating fine details like edges
and textures.
Ideal High Pass Filter:
This filter removes low-frequency components and retains high-
frequency details. The resulting image highlights edges
 and fine textures, while the larger, smooth areas are suppressed.
```

Apply Butterworth and Gaussian Filters

```
# Butterworth Low Pass Filter
butter_lp_mask = butterworth_low_pass_filter(image.shape, radius,
order=2)
butter_lp_filtered_image = apply_filter(dft_shift, butter_lp_mask)

# Butterworth High Pass Filter
butter_hp_mask = butterworth_high_pass_filter(image.shape, radius,
order=2)
butter_hp_filtered_image = apply_filter(dft_shift, butter_hp_mask)

# Gaussian Low Pass Filter
gaussian_lp_mask = gaussian_low_pass_filter(image.shape, radius)
gaussian_lp_filtered_image = apply_filter(dft_shift, gaussian_lp_mask)

# Gaussian High Pass Filter
```

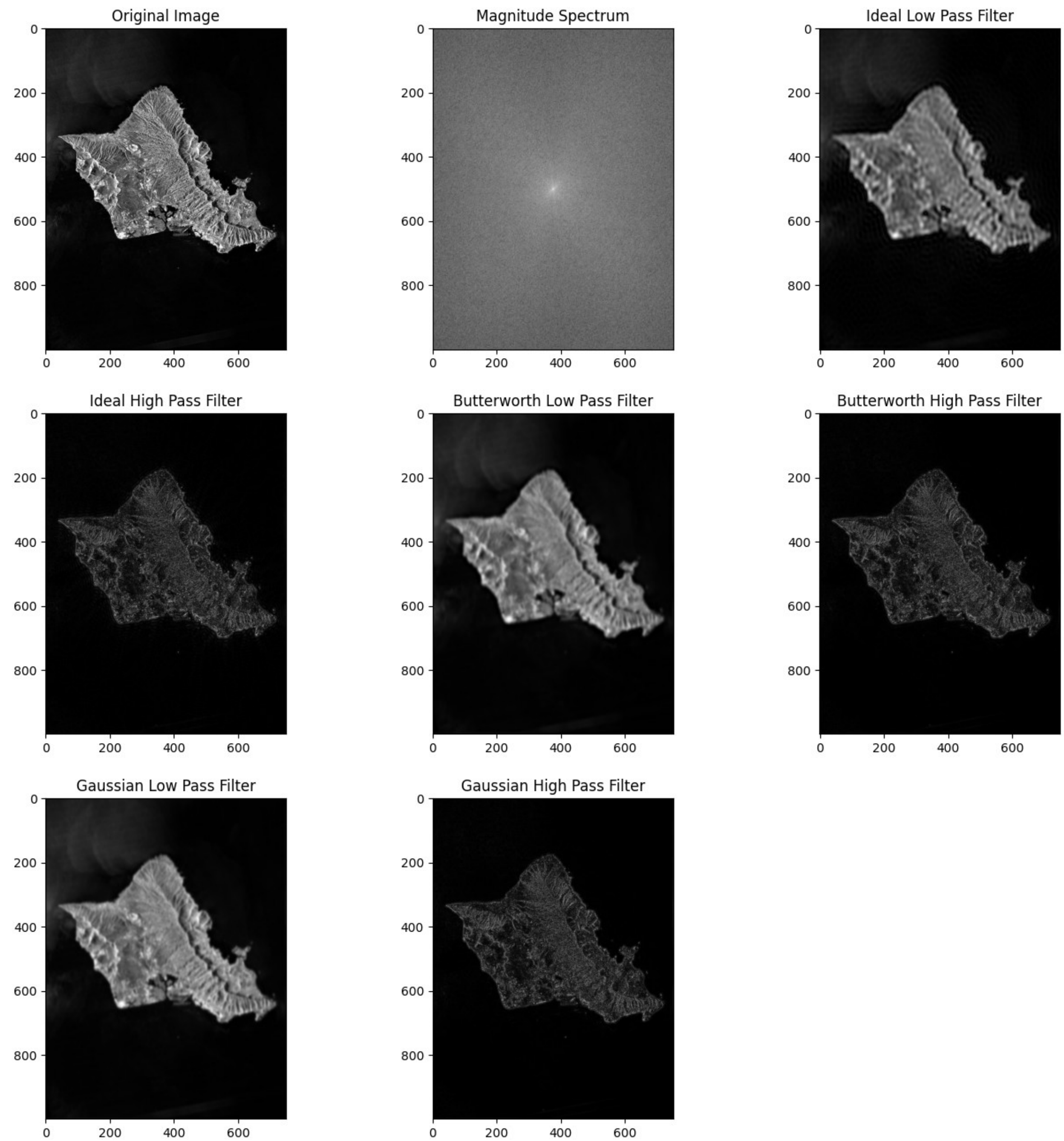
```python
gaussian_hp_mask = gaussian_high_pass_filter(image.shape, radius)
gaussian_hp_filtered_image = apply_filter(dft_shift, gaussian_hp_mask)

# Visualize Butterworth and Gaussian filters
plt.figure(figsize=(12,12))
plt.subplot(221), plt.imshow(butter_lp_filtered_image, cmap='gray'),
plt.title('Butterworth Low Pass Filter')
plt.subplot(222), plt.imshow(butter_hp_filtered_image, cmap='gray'),
plt.title('Butterworth High Pass Filter')
plt.subplot(223), plt.imshow(gaussian_lp_filtered_image, cmap='gray'),
plt.title('Gaussian Low Pass Filter')
plt.subplot(224), plt.imshow(gaussian_hp_filtered_image, cmap='gray'),
plt.title('Gaussian High Pass Filter')
plt.show()
```

```
<ipython-input-16-f410b506e6b2>:8: RuntimeWarning: divide by zero
encountered in scalar divide
  mask[i, j] = 1 / (1 + (radius / distance)**(2 * order))
```




```
Butterworth Low Pass Filter:
Similar to the Ideal Low Pass, but the Butterworth filter has a more
gradual cutoff.
 The result is smoother, preserving some high-frequency components
while still emphasizing broader structures.
Butterworth High Pass Filter:
This filter highlights edges and textures while gradually suppressing
```

```
low frequencies.
 It preserves fine details while maintaining smoother transitions
compared to the Ideal High Pass filter.

Gaussian Low Pass Filter:
The Gaussian Low Pass produces the smoothest result, with a gradual
reduction in high-frequency content.
 The image retains most of the broad structures, with less sharpness
in fine details.
Gaussian High Pass Filter:
This filter enhances high-frequency details like edges and fine
textures, with smoother transitions than the Ideal High Pass filter.
 The image looks sharp but not as harsh as with the Ideal High Pass
filter.

plt.figure(figsize=(16,16))

plt.subplot(331), plt.imshow(image, cmap='gray'), plt.title('Original
Image')
plt.subplot(332), plt.imshow(magnitude_spectrum, cmap='gray'),
plt.title('Magnitude Spectrum')
plt.subplot(333), plt.imshow(ideal_lp_filtered_image, cmap='gray'),
plt.title('Ideal Low Pass Filter')
plt.subplot(334), plt.imshow(ideal_hp_filtered_image, cmap='gray'),
plt.title('Ideal High Pass Filter')
plt.subplot(335), plt.imshow(butter_lp_filtered_image, cmap='gray'),
plt.title('Butterworth Low Pass Filter')
plt.subplot(336), plt.imshow(butter_hp_filtered_image, cmap='gray'),
plt.title('Butterworth High Pass Filter')
plt.subplot(337), plt.imshow(gaussian_lp_filtered_image, cmap='gray'),
plt.title('Gaussian Low Pass Filter')
plt.subplot(338), plt.imshow(gaussian_hp_filtered_image, cmap='gray'),
plt.title('Gaussian High Pass Filter')

plt.show()
```



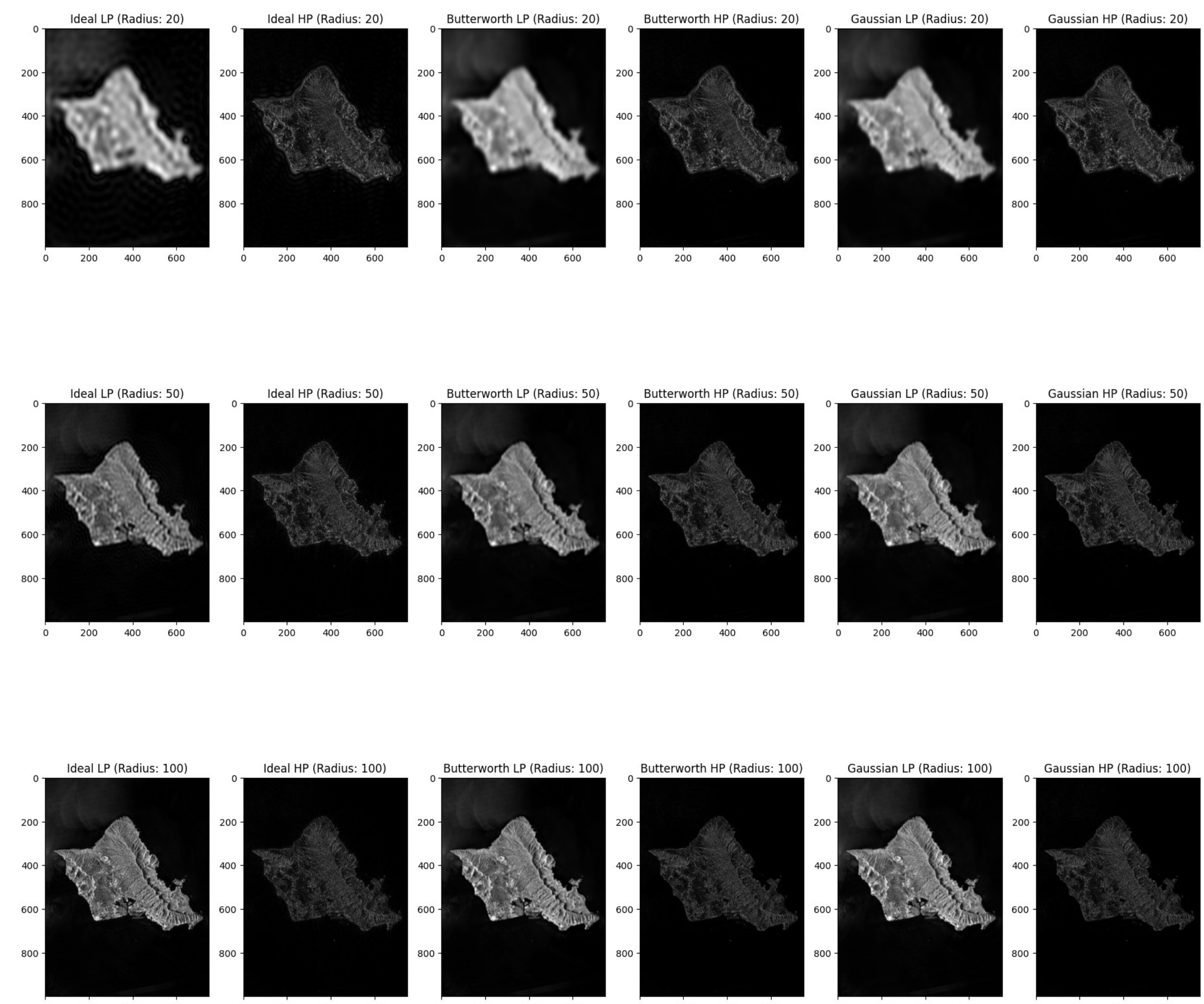

```
Low Pass Filters (Ideal, Butterworth, Gaussian) smooth the image by
removing high-frequency details.
The Gaussian filter provides the smoothest result, followed by
Butterworth and then Ideal.

High Pass Filters (Ideal, Butterworth, Gaussian) emphasize edges and
fine textures.
The Ideal filter is the sharpest and harshest, while the Butterworth
```

```
and Gaussian filters provide
 smoother transitions and more subtle edge enhancement.
```

Analyze Filter Radius Impact

```
import matplotlib.pyplot as plt
radii = [20, 50, 100]

plt.figure(figsize=(18, 18))

for i, radius in enumerate(radii):
    # Ideal Low Pass Filter
    ideal_lp_mask = ideal_low_pass_filter(image.shape, radius)
    ideal_lp_filtered_image = apply_filter(dft_shift, ideal_lp_mask)

    # Ideal High Pass Filter
    ideal_hp_mask = ideal_high_pass_filter(image.shape, radius)
    ideal_hp_filtered_image = apply_filter(dft_shift, ideal_hp_mask)

    # Butterworth Low Pass Filter
    butter_lp_mask = butterworth_low_pass_filter(image.shape, radius,
order=2)
    butter_lp_filtered_image = apply_filter(dft_shift, butter_lp_mask)

    # Butterworth High Pass Filter
    butter_hp_mask = butterworth_high_pass_filter(image.shape, radius,
order=2)
    butter_hp_filtered_image = apply_filter(dft_shift, butter_hp_mask)

    # Gaussian Low Pass Filter
    gaussian_lp_mask = gaussian_low_pass_filter(image.shape, radius)
    gaussian_lp_filtered_image = apply_filter(dft_shift,
gaussian_lp_mask)

    # Gaussian High Pass Filter
    gaussian_hp_mask = gaussian_high_pass_filter(image.shape, radius)
    gaussian_hp_filtered_image = apply_filter(dft_shift,
gaussian_hp_mask)


    plt.subplot(len(radii), 6, i * 6 + 1),
plt.imshow(ideal_lp_filtered_image, cmap='gray'), plt.title(f'Ideal LP
(Radius: {radius})')
    plt.subplot(len(radii), 6, i * 6 + 2),
plt.imshow(ideal_hp_filtered_image, cmap='gray'), plt.title(f'Ideal HP
(Radius: {radius})')
    plt.subplot(len(radii), 6, i * 6 + 3),
plt.imshow(butter_lp_filtered_image, cmap='gray'),
plt.title(f'Butterworth LP (Radius: {radius})')
    plt.subplot(len(radii), 6, i * 6 + 4),
```

```
plt.imshow(butter_hp_filtered_image, cmap='gray'),
plt.title(f'Butterworth HP (Radius: {radius})')
    plt.subplot(len(radii), 6, i * 6 + 5),
plt.imshow(gaussian_lp_filtered_image, cmap='gray'),
plt.title(f'Gaussian LP (Radius: {radius})')
    plt.subplot(len(radii), 6, i * 6 + 6),
plt.imshow(gaussian_hp_filtered_image, cmap='gray'),
plt.title(f'Gaussian HP (Radius: {radius})')


plt.tight_layout()
plt.show()

<ipython-input-16-f410b506e6b2>:8: RuntimeWarning: divide by zero
encountered in scalar divide
  mask[i, j] = 1 / (1 + (radius / distance)**(2 * order))
```



```
Low Pass Filters: As the radius increases, more frequencies are
allowed to pass, resulting in less blurring and
 a more detailed image.Gaussian filters provide the smoothest and most
natural results, while the ideal filters creat more abrupt
 transitions between passed and blocked frequencies, leading to more
noticeable artifacts.

High Pass Filters: With increasing radii, more of the image structure
is preserved.
 Ideal filters tend to produce sharper, more abrupt edges with more
noise, while Gaussian and
  Butterworth filters give smoother results, preserving edges without
harshness.
```