گزارش پروژه سیگنال‌ها و سیستم‌ها - بخش پردازش تصویر

۱. بخش اول: پیاده‌سازی و اعمال کانوولوشن دو-بعدی روی تصاویر

خب همونطور که در شرح پروژه اومده، در این بخش باید فاز رو به جلو (اعمال فیلتر) یک CNN رو پیاده‌سازی کنیم. برای این کار از کتابخونه‌های cv2, numpy و matplotlib استفاده کردیم.

اول توابعی که نیاز داریم رو تعریف می‌کنیم و بعد لایه‌های مختلف رو طراحی می‌کنیم.

- تابع کانوولوشن:

تعریف تابع و doc string:

```
def apply_convolution(image, kernel, stride=1, padding='valid'):
    """
    Apply a convolution operation on an image using a given kernel.

    Parameters:
    image (numpy.ndarray): Input image, a 3D array (height, width, channels).
    kernel (numpy.ndarray): Convolution kernel, a 2D array (kernel_height, kernel_width).
    stride (int, optional): Stride for the convolution. Default is 1.
    padding (str, optional): Padding type: 'valid' or 'same'. Default is 'valid'.

    Returns:
    numpy.ndarray: Output of the convolution, a 2D array (output_height, output_width).
    """
```

در مرحله‌ی بعدی، تصویر و کرنل داده‌شده رو برای اطمینان به numpy array تبدیل می‌کنیم تا بتونیم از قابلیت‌های numpy بهره ببریم. بعد ابعاد تصویر و کرنل رو با تابع shape به دست می‌آریم.

حالا باید نوع پدینگ خواسته شده رو چک کنیم. اگر مساوی با same باشه، به این معنیه که می‌خوایم خروجی ابعاد ورودی رو داشته باشه و با استفاده از ابعاد کرنل، طول و عرض پدینگ رو حساب می‌کنیم. در آخر هم به تصویر پدینگ با مقدار صفر می‌دیم.

اگر هم مساوی با valid باشه، تصویر رو تغییر نمی‌دیم و طول پدینگ هم برابر با صفره.

اگر حالتی غیر از این دو باشه هم ValueError رخ داده:

```
image = np.array(image)
kernel = np.array(kernel)
kernel_height, kernel_width = kernel.shape
image_height, image_width, channels = image.shape

# Handle padding
if padding == 'same':
    pad_height = kernel_height // 2
    pad_width = kernel_width // 2
    padded_image = np.zeros((image_height + 2 * pad_height, image_width + 2 * pad_width, channels))
    padded_image[pad_height:-pad_height, pad_width:-pad_width, :] = image
elif padding == 'valid':
    pad_height = 0
    pad_width = 0
    padded_image = image
else:
    raise ValueError('Invalid padding')
```

حالا با توجه به فرمولی که داشتیم، ابعاد تصویر خروجی رو محاسبه می‌کنیم. در آخر هم با توجه به stride کانوولوشن رو اجرا می‌کنیم:

```
# Calculate output dimensions
output_height = (image_height + 2 * pad_height - kernel_height) // stride + 1
output_width = (image_width + 2 * pad_width - kernel_width) // stride + 1
output = np.zeros((output_height, output_width))

# Perform convolution
for i in range(output_height):
    for j in range(output_width):
        for c in range(channels):
            start_i = i * stride
            start_j = j * stride
            end_i = start_i + kernel_height
            end_j = start_j + kernel_width
            output[i, j] += np.sum(padded_image[start_i:end_i, start_j:end_j, c] * kernel)

return output
```

فرمول:

$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

- تابع ماکس پولینگ:

تعریف تابع و doc string:

```
def max_pooling(image, size, stride):
    """
    Apply max pooling to an image.

    Parameters:
    image (numpy.ndarray): Input image, a 3D array (height, width, channels).
    size (int): Size of the pooling filter.
    stride (int): Stride for the pooling operation.

    Returns:
    numpy.ndarray: Output of the max pooling, a 3D array (output_height, output_width, channels).
    """
```

ابتدا تصویر داده‌شده رو به numpy array تبدیل می‌کنیم. بعد ابعاد تصویر خروجی رو به دست می‌آریم و خروجی رو initialize می‌کنیم:

```
image = np.array(image)
image_height, image_width, channels = image.shape

output_height = (image_height - size) // stride + 1
output_width = (image_width - size) // stride + 1

pooled_image = np.zeros((output_height, output_width, channels))
```

در آخر هم استخراج مقادیر maximum از تصویر:

```
for i in range(output_height):
    for j in range(output_width):
        for c in range(channels):
            start_i = i * stride
            start_j = j * stride
            end_i = start_i + size
            end_j = start_j + size
            pooled_image[i, j, c] = np.max(image[start_i:end_i, start_j:end_j, c])

return pooled_image
```

- پیش‌پردازش تصویر:

شامل لود تصویر، resize، و نمایش اون:

```
# ——————————— Preprocessing ———————————
# Load image
image = cv2.imread('image/yann_lecun.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Resize to 600×600
image = cv2.resize(image, (600, 600))

# Show the original image
plt.figure(figsize=(16, 8))
plt.subplot(2, 4, 1)
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')
```

- لایه‌ی اول:

در این مرحله باید فیلترهای Edge Enhancement، Horizontal Edge Detection و Vertical Edge Detection رو به کمک تابع کانوولوشن روی تصویر اعمال کنیم.

به ترتیب هر کدوم رو روی تصویر اصلی اعمال می‌کنیم و با subplot نمایش می‌دیم.

Edge Enhancement:

```
# ——————————— Layer 1 ———————————
# Edge enhancement kernel
edge_enhancement_kernel = np.array([[-1, -1, -1],
                                    [-1, 8, -1],
                                    [-1, -1, -1]])
# Apply convolution
edge_enhanced_image = apply_convolution(image, edge_enhancement_kernel, padding='same')
plt.subplot(2, 4, 2)
plt.title('Edge Enhanced Image')
plt.imshow(edge_enhanced_image, cmap='gray')
plt.axis('off')
```

Horizontal Edge Detection:

```
# Horizontal edge detection kernel
horizontal_edge_kernel = np.array([[1, 1, 1],
                                   [0, 0, 0],
                                   [-1, -1, -1]])
# Apply convolution
horizontal_edge_image = apply_convolution(image, horizontal_edge_kernel, padding='same')
plt.subplot(2, 4, 3)
plt.title('Horizontal Edge Image')
plt.imshow(horizontal_edge_image, cmap='gray')
plt.axis('off')
```

Vertical Edge Detection:

```
# Vertical edge detection kernel
vertical_edge_kernel = np.array([[1, 0, -1],
                                 [1, 0, -1],
                                 [1, 0, -1]])
# Apply convolution
vertical_edge_image = apply_convolution(image, vertical_edge_kernel, padding='same')
plt.subplot(2, 4, 4)
plt.title('Vertical Edge Image')
plt.imshow(vertical_edge_image, cmap='gray')
plt.axis('off')
```

در آخر خروجی این سه فیلتر باید به صورت سه بعدی استک بشه تا یک feature map

داشته باشیم که لبه‌های تصویر رو به خوبی بولد کرده:

```
# Stack images to create a feature map
feature_map = np.stack((edge_enhanced_image, horizontal_edge_image, vertical_edge_image), axis=-1)
```

- لایه‌ی دوم:

در این لایه از فیلترهای Gaussian Blur و Large Edge Enhancement استفاده می‌کنیم.

مراحل مشابه لایه‌ی اوله، با این تفاوت که فیلترها روی feature map حاصل از مرحله‌ی قبل اعمال می‌شن.

Gaussian Blur:

```
# ——————————— Layer 2 ———————————
# Gaussian blur kernel
gaussian_blur_kernel = np.array([[1, 4, 7, 4, 1],
                                 [4, 16, 26, 16, 4],
                                 [7, 26, 41, 26, 7],
                                 [4, 16, 26, 16, 4],
                                 [1, 4, 7, 4, 1]])

# Apply convolution to the feature map
blurred_feature_map = apply_convolution(feature_map, gaussian_blur_kernel, padding='same')
plt.subplot(2, 4, 5)
plt.title('Blurred Feature Map')
plt.imshow(blurred_feature_map, cmap='gray')
plt.axis('off')
```

Large Edge Enhancement:

```
# Large edge enhancement kernel
large_edge_enhancement_kernel = np.array([[0, 0, -1, 0, 0],
                                          [0, 0, -1, 0, 0],
                                          [-1, -1, 8, -1, -1],
                                          [0, 0, -1, 0, 0],
                                          [0, 0, -1, 0, 0]])

# Apply convolution to the feature map
large_edge_enhanced_feature_map = apply_convolution(feature_map, large_edge_enhancement_kernel, padding='same')
plt.subplot(2, 4, 6)
plt.title('Large Edge Enhanced Feature Map')
plt.imshow(large_edge_enhanced_feature_map, cmap='gray')
plt.axis('off')
```

در آخر هم درست‌کردن یک feature map جدید:

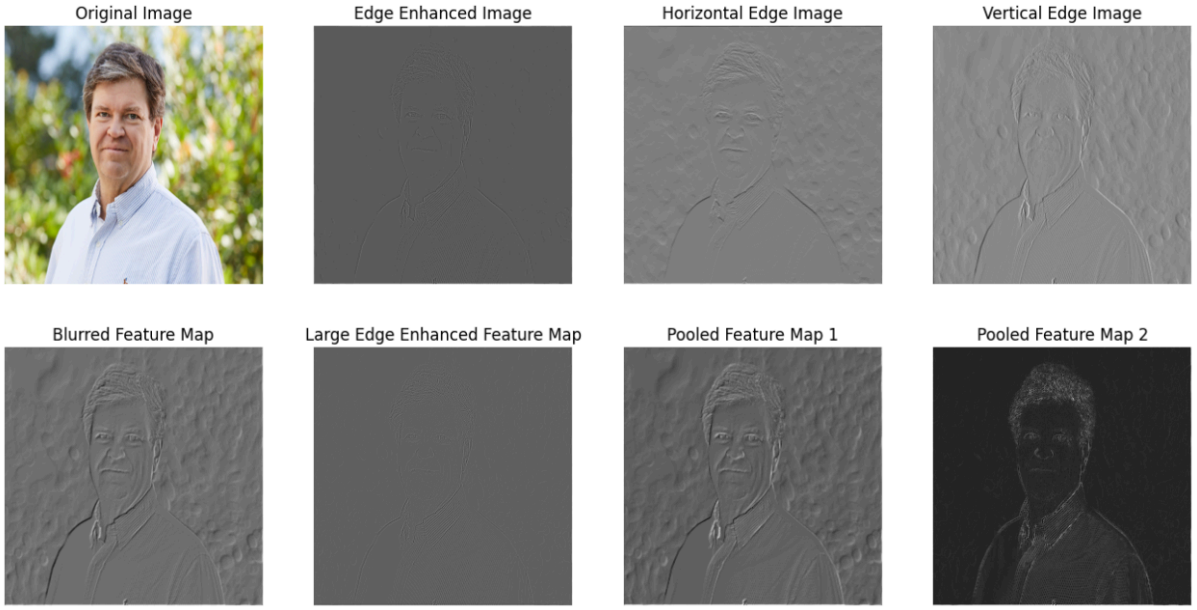
```
# Stack images to create a feature map
feature_map2 = np.stack((blurred_feature_map, large_edge_enhanced_feature_map), axis=-1)
```

- لایه‌ی سوم:

در لایه‌ی آخر تابع ماکس پولینگ رو برای feature map لایه‌ی دو اجرا می‌کنیم و هر کانال رو جداگانه نمایش می‌دیم.

```
# ——————————— Layer 3 ———————————
# Max pooling
pooled_feature_map = max_pooling(feature_map2, 2, 2)
# show each channel separately
plt.subplot(2, 4, 7)
plt.title('Pooled Feature Map 1')
plt.imshow(pooled_feature_map[:, :, 0], cmap='gray')
plt.axis('off')

plt.subplot(2, 4, 8)
plt.title('Pooled Feature Map 2')
plt.imshow(pooled_feature_map[:, :, 1], cmap='gray')
plt.axis('off')

plt.show()
```

- خروجی:


Original Image
Edge Enhanced Image
Horizontal Edge Image
Vertical Edge Image
Blurred Feature Map
Large Edge Enhanced Feature Map
Pooled Feature Map 1
Pooled Feature Map 2

۲. بخش دوم: پردازش تصویر در حوزه‌ی فرکانس

در این بخش فیلترهای مختلفی رو در حوزه‌ی فرکانس روی تصویر اعمال می‌کنیم و تاثیر اونها رو می‌بینیم. همونطور که می‌دونیم، این روش بار محاسباتی بسیار کمتری داره چون کانوولوشن در حوزه‌ی زمان به ضرب در حوزه‌ی فرکانس تبدیل می‌شه.

- پیش‌پردازش تصویر

شامل لود تصویر، resize و نمایش‌دادن اون:

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

# Load image as grayscale
image = cv2.imread('image/yann_lecun.jpg', cv2.IMREAD_GRAYSCALE)

# Resize to 600×600
image = cv2.resize(image, (600, 600))

# Display the original image
plt.figure(figsize=(18, 12))
plt.subplot(4, 4, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')
```

- بردن تصویر به حوزه‌ی فرکانس

تبدیل فوریه‌ی سریع تصویر رو محاسبه می‌کنیم و اندازه و فاز رو پلات می‌کنیم. فرکانس صفر رو جابه‌جا می‌کنیم تا مرکز طیف قرار بگیره و خروجی زیباتری داشته باشیم:

```
# Compute the Fast Fourier Transform (FFT)
fft = np.fft.fft2(image)

# Shift the zero-frequency component to the center of the spectrum
fft_shifted = np.fft.fftshift(fft)

# Compute magnitude and phase spectrum
magnitude_spectrum = np.abs(fft_shifted)
phase_spectrum = np.angle(fft_shifted)

# Display the magnitude and phase spectrum
plt.subplot(4, 4, 2)
plt.title('Original Magnitude Spectrum')
plt.imshow(np.log(1 + magnitude_spectrum), cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 3)
plt.title('Original Phase Spectrum')
plt.imshow(phase_spectrum, cmap='gray')
plt.axis('off')
```

- اعمال ۵ فیلتر روی تصویر

حالا ۵ فیلتر پایین‌گذر، بالاگذر، Band-stop، Band-pass و Laplacian رو تعریف می‌کنیم و روی تصویر در حوزه‌ی فرکانس ضرب می‌کنیم.

فیلترهای پایین/بالاگذر:

این جفت فیلتر رو در یک تابع پیاده‌سازی کردیم. ابتدا تبدیل فوریه‌ی تصویر رو حساب می‌کنیم و یک ماسک ایجاد می‌کنیم که همون فیلترمونه. با توجه به بالا یا پایین‌گذربودن فیلتر، مقادیر لازم رو صفر یا یک می‌کنیم.

```python
# Low/High-pass filter
def low_high_pass_filter(image, cutoff, low=True):
    fft = np.fft.fft2(image)
    fft_shifted = np.fft.fftshift(fft)

    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2   # center of the frequency domain

    # Create a mask with ones in the low frequencies and zeros in the high frequencies (for low-pass)
    # and vice versa for high-pass
    mask = np.zeros((rows, cols), np.uint8)
    if low:
        mask[crow - cutoff:crow + cutoff, ccol - cutoff:ccol + cutoff] = 1
    else:
        mask = np.ones((rows, cols), np.uint8)
        mask[crow - cutoff:crow + cutoff, ccol - cutoff:ccol + cutoff] = 0
```

در آخر ضرب فیلتر در تصویر و تبدیل فوریه‌ی معکوس:

```python
# Apply the mask to the FFT of the image
filtered_fft_shifted = fft_shifted * mask

# Shift the zero-frequency component back to the top-left
fft = np.fft.ifftshift(filtered_fft_shifted)

# Compute the inverse FFT
filtered_image = np.fft.ifft2(fft)
filtered_image = np.abs(filtered_image)

return filtered_image, filtered_fft_shifted
```

فیلترهای Band-pass و Band-stop:

مراحل اینجا هم مشابه بالاست و فقط mask تغییر می‌کنه، یک فرکانس cutoff_high هم لازم داریم.

```python
# band-pass/band-stop filter
def band_pass_stop_filter(image, cutoff_low, cutoff_high, filter_type='pass'):
    fft = np.fft.fft2(image)
    fft_shifted = np.fft.fftshift(fft)

    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2   # center of the frequency domain

    # Create a mask with ones for band-pass, will be inverted for band-stop
    mask = np.ones((rows, cols), np.uint8)
    mask[crow - cutoff_high:crow + cutoff_high, ccol - cutoff_high:ccol + cutoff_high] = 0
    mask[crow - cutoff_low:crow + cutoff_low, ccol - cutoff_low:ccol + cutoff_low] = 1

    if filter_type == 'stop':
        mask = 1 - mask  # Invert mask for band-stop filter
```

ضرب فیلتر در تصویر و تبدیل فوریه‌ی معکوس:

```python
    # Apply the mask to the FFT of the image
    filtered_fft_shifted = fft_shifted * mask

    # Shift the zero-frequency component back to the top-left
    filtered_fft = np.fft.ifftshift(filtered_fft_shifted)

    # Compute the inverse FFT
    filtered_image = np.fft.ifft2(filtered_fft)
    filtered_image = np.abs(filtered_image)

    return filtered_image, filtered_fft_shifted
```

فیلتر Laplacian:

این فیلتر یک فیلتر مرتبه دوم مشتق‌گیره که به تشخیص لبه‌ها در تصویر کمک می‌کنه.

فرمول در حوزه‌ی فرکانس:

H(u, v) = -4 * pi^2 * (u^2 + v^2)

که مقادیر u و v مختصات حوزه‌ی فرکانس هستن.

باقی مراحل مشابه فیلترهای قبلی‌ان:

```python
# Laplacian filter
def laplacian_filter(image):
    fft = np.fft.fft2(image)
    fft_shifted = np.fft.fftshift(fft)

    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2  # center of the frequency domain

    # Create a Laplacian filter in the frequency domain
    u = np.arange(rows) - crow
    v = np.arange(cols) - ccol
    U, V = np.meshgrid(u, v)
    laplacian_filter = -4 * np.pi ** 2 * (U ** 2 + V ** 2)
```

ضرب فیلتر در تصویر و تبدیل فوریه‌ی معکوس:

```
# Apply the Laplacian filter to the FFT of the image
laplacian_fft_shifted = fft_shifted * laplacian_filter

# Shift the zero-frequency component back to the top-left
fft = np.fft.ifftshift(laplacian_fft_shifted)

# Compute the inverse FFT
filtered_image = np.fft.ifft2(fft)
filtered_image = np.abs(filtered_image)

return filtered_image, laplacian_fft_shifted
```

در آخر هم تمام فیلترهای تعریف‌شده رو در تصویر اعمال می‌کنیم و نمایش می‌دیم.

```
# Apply filters
low_pass_image, low_pass_fft_shifted = low_high_pass_filter(image, cutoff=50, low=True)
high_pass_image, high_pass_fft_shifted = low_high_pass_filter(image, cutoff=50, low=False)
band_pass_image, band_pass_fft_shifted = band_pass_stop_filter(image, cutoff_low=30, cutoff_high=60, filter_type='pass')
band_stop_image, band_stop_fft_shifted = band_pass_stop_filter(image, cutoff_low=30, cutoff_high=60, filter_type='stop')
laplacian_image, laplacian_fft_shifted = laplacian_filter(image)

# Compute phase spectra
low_pass_phase_spectrum = np.angle(low_pass_fft_shifted)
high_pass_phase_spectrum = np.angle(high_pass_fft_shifted)
band_pass_phase_spectrum = np.angle(band_pass_fft_shifted)
band_stop_phase_spectrum = np.angle(band_stop_fft_shifted)
laplacian_phase_spectrum = np.angle(laplacian_fft_shifted)
```

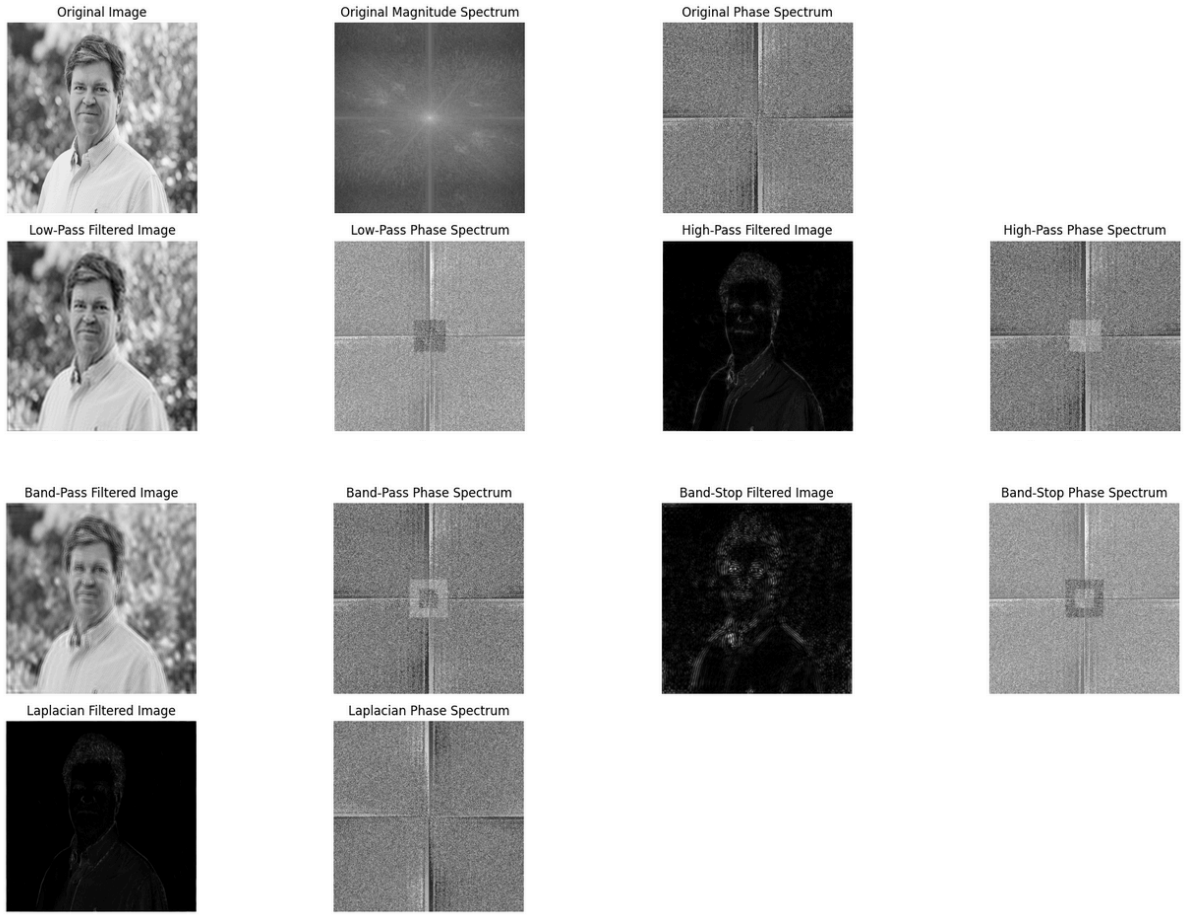
```
plt.subplot(4, 4, 8)
plt.title('High-Pass Phase Spectrum')
plt.imshow(high_pass_phase_spectrum, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 9)
plt.title('Band-Pass Filtered Image')
plt.imshow(band_pass_image, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 10)
plt.title('Band-Pass Phase Spectrum')
plt.imshow(band_pass_phase_spectrum, cmap='gray')
plt.axis('off')
```

```
# Display the filtered images and their phase spectra
plt.subplot(4, 4, 5)
plt.title('Low-Pass Filtered Image')
plt.imshow(low_pass_image, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 6)
plt.title('Low-Pass Phase Spectrum')
plt.imshow(low_pass_phase_spectrum, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 7)
plt.title('High-Pass Filtered Image')
plt.imshow(high_pass_image, cmap='gray')
plt.axis('off')
```

```
plt.subplot(4, 4, 11)
plt.title('Band-Stop Filtered Image')
plt.imshow(band_stop_image, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 12)
plt.title('Band-Stop Phase Spectrum')
plt.imshow(band_stop_phase_spectrum, cmap='gray')
plt.axis('off')

plt.subplot(4, 4, 13)
plt.title('Laplacian Filtered Image')
plt.imshow(laplacian_image, cmap='gray')
plt.axis('off')
```

```
plt.subplot(4, 4, 14)
plt.title('Laplacian Phase Spectrum')
plt.imshow(laplacian_phase_spectrum, cmap='gray')
plt.axis('off')

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

و خروجی:



- بررسی تغییرات هر فیلتر روی تصویر

فیلتر پایین‌گذر:

این فیلتر فقط به فرکانس‌های پایین اجازه‌ی عبور می‌ده. این باعث می‌شه تصویر مقداری blurred باشه چون لبه‌ها و بخش‌های تیز تصویر و همچنین نویزها صفر شدن.

فیلتر بالاگذر:

این فیلتر برعکس پایین‌گذر هست و فقط به فرکانس‌های بالا اجازه‌ی عبور می‌ده. این باعث می‌شه تصویر به اندازه‌ی قبل smooth نباشه و لبه‌ها و جزئیات نمایان‌تر باشن.

فیلتر Band-pass:

این فیلتر به فرکانس‌های بین cutoff_low و cutoff_high اجازه‌ی عبور می‌ده. با توجه به انتخاب فرکانس‌های قطع، تصویر نهایی جزئیاتی که در band هستن رو هایلایت می‌کنه. به دلیل قطع فرکانس‌های بالا و پایین، تصویر نهایی نه کاملا blurred شده و نه کاملا تیز.

فیلتر Band-stop:

این فیلتر به فرکانس‌های بین cutoff_low و cutoff_high اجازه‌ی عبور نمی‌ده. با توجه به انتخاب فرکانس‌های قطع، تصویر نهایی جزئیاتی که در band هستن رو حذف می‌کنه.

فیلتر لاپلاسی:

تصویر نهایی این فیلتر لبه‌ها رو به خوبی تشخیص داده و جاهایی که تصویر تغییرات شدیدی داشته رو هایلایت می‌کنه.