

CSED551 PA#2

:Frequency Domain Processing

20220848 Minsu Sun

October 20, 2024

Problem 1

Algorithm - Primitives

아래는 Ideal Lowpass Filtering을 구현함에 있어서 필요함에 따라 작성한 함수들에 대한 설명이다.

```
7 def squaredDistanceMatrix(shape: tuple[int, int]) -> "np.ndarray[np.float32]":
8     """Get squared distance matrix shaped of given shape
9
10    Args:
11        shape (tuple[int, int]): shape of desired distance matrix
12
13    Returns:
14        np.ndarray[np.float32]: squared distance matrix
15    """
16    u: np.ndarray[np.float32] = (
17        np.arange(shape[0]).astype(np.float32) - (shape[0] - 1) / 2
18    )
19    v: np.ndarray[np.float32] = (
20        np.arange(shape[1]).astype(np.float32) - (shape[1] - 1) / 2
21    )
22    U: np.ndarray[np.float32]
23    V: np.ndarray[np.float32]
24    V, U = np.meshgrid(v, u) # U, V are both shaped like (u.shape[0], v.shape[0])
25    return U**2 + V**2 # distance^2 matrix from the middle
```

Primitive - squaredDistanceMatrix

위의 코드는 매개변수로 주어진 `shape`의 형태에 맞춘 Distance Matrix D 의 각 원소들의 제곱으로 이루어진 D^2 를 반환하는 함수이다.

```
28 def frequencyDomainFiltering(
29     image: "np.ndarray[np.uint8 | np.float32]",
30     filter: "np.ndarray[np.uint8 | np.float32]",
31 ) -> "np.ndarray[np.uint8]":
32     """Filter given image in frequency domain using FFT
33
34     Args:
35         image (np.ndarray[np.uint8 | np.float32]): image to filter
36         filter (np.ndarray[np.uint8 | np.float32]): filter to use, filter shape should be same
37             with image in the perspective of width and height
38
39     Returns:
40         np.ndarray[np.uint8]: filtered image
41     """
42     assert len(image.shape) == 3
43     assert len(filter.shape) == 2
44     assert (
45         image.shape[:2] == filter.shape
46     ) # image and filter should be matched to be broadcasted
```

```

46
47     result_channels: list[np.ndarray[np.uint8]] = []
48
49     for channel in cv2.split(image):
50         # move channel to frequency domain
51         channel_f: np.ndarray[np.complex128] = np.fft.fft2(channel)
52         channel_f_s: np.ndarray[np.complex128] = np.fft.fftshift(channel_f)
53
54         result_f_s: np.ndarray[np.complex128] = channel_f_s * filter
55
56         # move back to spatial domain
57         result_f: np.ndarray[np.complex128] = np.fft.ifftshift(result_f_s)
58         result: np.ndarray[np.float64] = np.real(np.fft.ifft2(result_f))
59
60         # normalize the values with interval of original channel image
61         result: np.ndarray[np.uint8] = cv2.normalize(
62             result,
63             None,
64             np.min(channel),
65             np.max(channel),
66             cv2.NORM_MINMAX,
67             -1,
68         ).astype(np.uint8)
69
70     result_channels.append(result)
71
72 return cv2.merge(result_channels)

```

Primitive - frequencyDomainFiltering

위의 함수는 매개변수로 주어진 이미지를 Frequency Domain에서 매개변수의 필터를 이용하여 필터링을 진행 후, 결과를 반환하는 함수이다. 이 때, 매개변수로 전달되는 필터는 주어진 형태 그대로 사용한다. 따로 필터를 Frequency Domain으로 변환하여 사용하지 않는다.

- 주어지는 매개변수들은 각각 컬러 이미지인지, 2차원의 필터인지 그리고 이미지의 높이 및 너비가 필터와 상응하는지 `assert`를 이용해 체크한다.
- 이후 이미지의 각 채널별로 나누어서 필터링을 진행한다.
- 각 채널은 FFT 적용 이후, `np.fft.fftshift`에 의해 변환된다.
- FFT에 의해 Frequency Domain으로 변환된 이미지에 주어진 필터를 적용시킨다.
- 이후 필터가 적용된 이미지를 다시 `np.fft.ifftshift`와 `np.fft.ifft2`를 이용하여 다시 Spatial Domain으로 변환한다.
- 이 때, Spatial Domain의 이미지는 픽셀의 표현값인 255를 넘어갈 수 있으므로, 필터가 적용되기 전의 픽셀 범위를 이용하여 정규화한다.
- 필터링 및 정규화가 완료된 각 채널들은 모두 다시 `cv2.merge`를 통해 합쳐진 후 반환된다.

```

78 def idealLowPassFilter(
79     shape: tuple[int, int],
80     threshold: float,
81 ) -> "np.ndarray[np.float32]":
82     """Build `shape` Shaped **Ideal Low Pass Filter** with Threshold.
83
84     Ideal Low Pass Filter can be formulated below:
85
86     f(u,v) = 1 if D(u,v) <= D_0
87

```

```

88     f(u,v) = 0 otherwise
89
90     Args:
91         shape (tuple[int, int]): desired shape of ideal low pass filter
92         threshold (float): threshold for ideal low pass filter, D_0
93
94     Returns:
95         np.ndarray[np.float32]: Ideal Low Pass Filter
96         """
97     D: np.ndarray[np.float32] = squaredDistanceMatrix(shape)
98     return (D <= threshold**2).astype(np.float32)

                                         Primitive - idealLowPassFilter

```

Ideal Lowpass Filter를 주어진 `shape`, `threshold`에 맞추어 생성 후 반환하는 함수이다. `shape`에 맞는 거리 행렬을 `squaredDistanceMatrix`를 통해 구하고 이를 $threshold^2$ 를 이용해 필터링하여 필터를 생성한다.

Algorithm - Main Process

아래는 Problem 1을 해결하기 위한 함수를 설명한다.

```

101    def idealLowPassFiltering(
102        image: "np.ndarray[np.uint8]",
103        padding: int,
104        threshold: float,
105        strip_padding: bool = True,
106    ) -> "np.ndarray[np.uint8]":
107        """Filter the image with ideal low pass filter
108
109        Args:
110            image (np.ndarray[np.uint8]): image to filter
111            padding (int): size of padding
112            threshold (float): threshold of filter, D_0
113            strip_padding (bool): flag to strip the padding of result image. Defaults to True
114
115        Returns:
116            np.ndarray[np.uint8]: filtered image
117            """
118        assert len(image.shape) == 3 # Colored Image
119        assert image.shape[2] == 3 # RGB
120        assert padding >= 0
121        assert threshold >= 0
122
123        padded_image: np.ndarray[np.uint8] = cv2.copyMakeBorder(
124            image,
125            padding,
126            padding,
127            padding,
128            padding,
129            DEFAULT_BORDER_TYPE,
130        )
131
132        # ideal low pass filter
133        ilf: np.ndarray[np.float32] = idealLowPassFilter(padded_image.shape[:2], threshold)
134
135        # result is (height+2*padding, width+2*padding, channels) shaped
136        result: np.ndarray[np.uint8] = frequencyDomainFiltering(padded_image, ilf)
137

```

```

138     if not strip_padding or padding == 0:
139         return result
140     else:
141         return result[padding:-padding, padding:-padding, :]
142         idealLowPassFiltering

```

위의 함수는 Ideal Lowpass Filter를 주어진 이미지에 적용하는 함수이다. 필터는 `padding`, `threshold`에 의하여 변화하며, `strip_padding`은 Visualization을 위한 매개변수로, 기본값은 `True`며, 결과 이미지의 패딩을 제거할지를 결정한다.

- 주어진 이미지를 `padding`만큼 패딩하여 이미지를 준비한다.
- 이때, 사용하는 `DEFAULT_BORDER_TYPE`는 `cv2.BORDER_REPLICATE`이다.
- 해당 함수는 Ideal Lowpass Filter를 사용하므로, `idealLowPassFiltering`을 이용해 필터를 생성 후, `frequencyDoMainFiltering`을 이용해 필터링을 진행한다.
- 이후 결과 이미지를 `strip_padding`이나 `padding`에 따라 패딩을 제거하거나 제거하지 않은 상태 그대로 반환한다.
- `padding`이 0일 경우에는 `result[padding:-padding, padding:-padding, :]`에서 빈 이미지를 만드므로 따로 처리한다.

Visualization

Input Images



(a) color1.jpg



(b) color3.jpg



(c) color4.jpg

Input Images

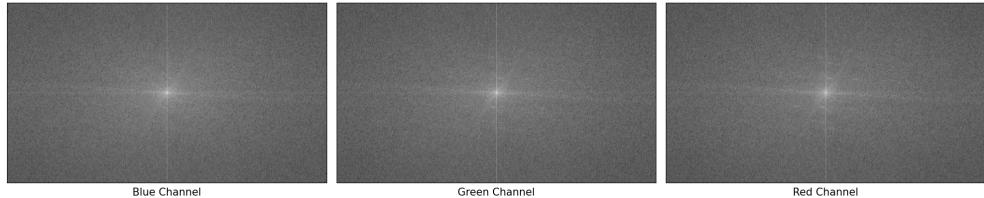
입력으로 사용한 이미지는 위와 같다. 주어진 `color3.jpg` 이외에 지난 Assignment 1에서 사용한 `color1.jpg`와 `color4.jpg`를 추가로 사용한다. 이는 이후의 문제에서도 동일하게 사용하며, 이미지의 해상도는 아래와 같다.

- `color1.jpg` : 800px x 450px
- `color3.jpg` : 800px x 533px
- `color4.jpg` : 3024 x 3024px

FFT of Input Images

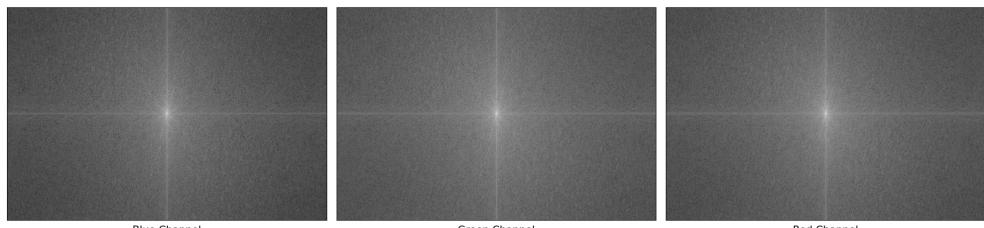
아래는 입력으로 사용한 각 이미지들의 각 채널별로 Frequency Domain을 나타낸 것이다.

Original color1.jpg in Frequency Domain



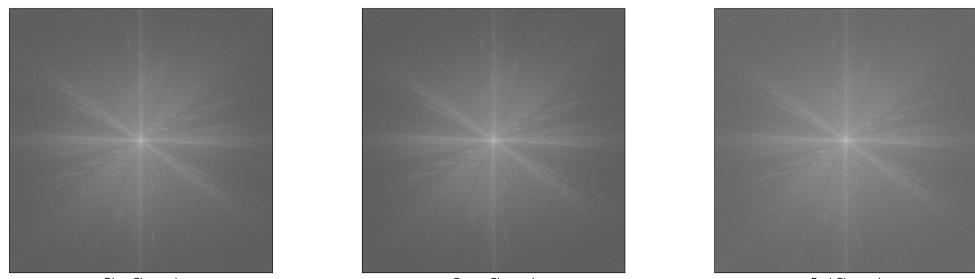
(a) color1

Original color3.jpg in Frequency Domain



(b) color3

Original color4.jpg in Frequency Domain



(c) color4

FFT of Input Images

Ideal Lowpass Filter in the Frequency Domain

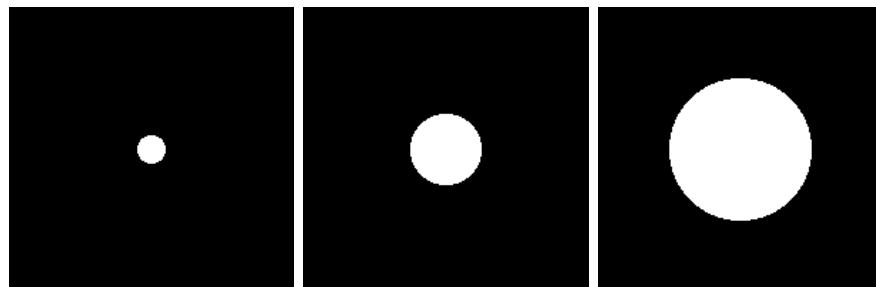
다음은 Ideal Lowpass Filter를 Frequency Domain에서 나타낸 이미지이다. 각 필터는 임의로 200px x 200px의 크기로 생성되었다.

Filtering Result in the Frequency Domain

다음은 Ideal Lowpass Filtering을 적용한 결과이다. 본문에는 각 이미지 별로 Lowpass Filtering이 잘 드러나는 이미지만을 첨부한다.

Result

아래는 Ideal Lowpass Filter의 Threshold에 따른 필터링 결과를 나타낸 것이다.



(a) threshold = 10

(b) threshold = 25

(c) threshold = 50

Ideal Lowpass Filter in the Frequency Domain



(a) color1 - padding=30,threshold=10

Filtered color1.jpg in Frequency Domain



Blue Channel

Green Channel

Red Channel

(b) frequency domain



(c) color3 - padding=30,threshold=10

Filtered color3.jpg in Frequency Domain

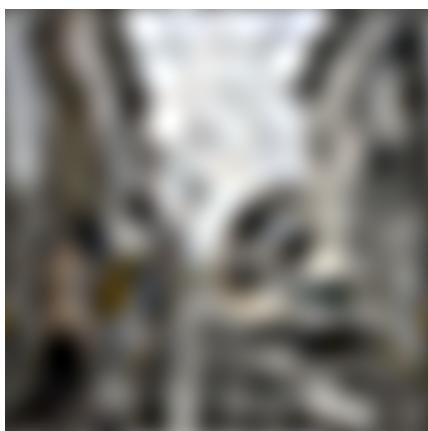


Blue Channel

Green Channel

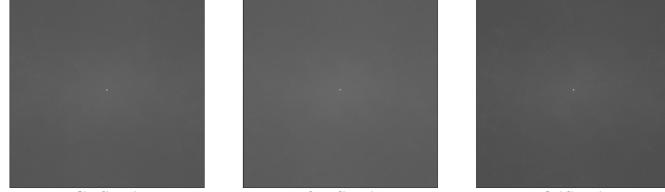
Red Channel

(d) frequency domain



(e) color4 - padding=30,threshold=10

Filtered color4.jpg in Frequency Domain



Blue Channel

Green Channel

Red Channel

(f) frequency domain

Result of Ideal Lospass Filtering in the Frequency Domain



(a) color3 - padding=30,threshold=10

(b) color3 - padding=30,threshold=25

(c) color3 - padding=30,threshold=50

Result of Ideal Lospass Filtering on Threshold of Filter

Discussion

Boundary Handling

다음은 동일 threshold 하에서 padding에 따른 필터링 결과를 나타낸 것이다.



(a) color3 - padding=10,threshold=10

(b) color3 - padding=20,threshold=25

(c) color3 - padding=30,threshold=50

Result of Ideal Lospass Filtering on Padding of Image

Appropriate Padding Size With Respect to the Filter Size

위의 필터링 결과를 보면, 작은 padding의 크기가 작을 수록 이미지의 테두리에서 어두운 부분을 확인할 수 있다. 이는 FFT의 Periodic Signal을 가정함에 따라서 나타나는 결과로, Spatial Domain과 Frequency Domain 사이의 변환에서 일어나는 현상이다. Frequency Domain에서 Periodic한 이미지 기반으로 연산을 진행하였으나, Spatial Domain에서 봤을 때, 이미지의 두 주기 사이에서 급격한 신호의 변화가 있는 경우 발생하게 된다. 위의 이미지들은 모두 cv2.copyMakeBorder를 이용하여 패딩을 진행하였지만, 앞의 두 이미지는 패딩이 부족해 테두리에서 어두운 결과를 보인다. 이를 통해 Frequency Filtering의 적절한 패딩은 필터의 threshold 이상의 값을 설정하는 것이 적절하다고 생각된다.

Problem 2

Algorithm - Primitives

아래는 Gaussian Lowpass Filtering을 구현함에 있어서 필요함에 따라 작성한 함수들에 대한 설명이다.

```

147 def gaussianLowPassFilter(
148     shape: tuple[int, int],
149     threshold: float,
150 ) -> "np.ndarray[np.float32]":
151     """Build `shape` Shaped **Gaussian Low Pass Filter** with Threshold.
152
153     Gaussian Low Pass Filter can be formulated below:
154

```

```

155 f(u,v) = exp(-D(u,v)^2/D_0^2)
156
157 Args:
158     shape (tuple[int, int]): desired shape of gaussian low pass filter
159     threshold (float): threshold for gaussian low pass filter, D_0
160
161 Returns:
162     np.ndarray[np.float32]: Gaussian Low Pass Filter
163 """
164 D: np.ndarray[np.float32] = squaredDistanceMatrix(shape)
165 return np.exp(-D / threshold**2)

Primitive - gaussianLowPassFilter

```

위 함수는 Gaussian Lowpass Filter를 구현하여 반환하는 함수이다. `squaredDistanceMatrix`를 기반으로 주어진 `shape` 형태의 Guassian Lowpass Filter를 제작해 반환한다.

Algorithm - Main Process

아래는 Problem 2를 해결하기 위한 함수를 설명한다.

```

168 def gaussianLowPassFiltering(
169     image: "np.ndarray[np.uint8]",
170     padding: int,
171     threshold: float,
172     strip_padding: bool = True,
173 ) -> "np.ndarray[np.uint8]":
174     """Filter the image with gaussian low pass filter
175
176     Args:
177         image (np.ndarray[np.uint8]): image to filter
178         padding (int): size of padding
179         threshold (float): threshold of filter, D_0
180         strip_padding (bool): flag to strip the padding of result image. Defaults to True
181
182     Returns:
183         np.ndarray[np.uint8]: filtered image
184     """
185     assert len(image.shape) == 3 # Colored Image
186     assert image.shape[2] == 3 # RGB
187     assert padding >= 0
188     assert threshold >= 0
189
190     padded_image: np.ndarray[np.uint8] = cv2.copyMakeBorder(
191         image,
192         padding,
193         padding,
194         padding,
195         padding,
196         DEFAULT_BORDER_TYPE,
197     )
198
199     # gaussian low pass filter
200     glf: np.ndarray[np.float32] = gaussianLowPassFilter(
201         padded_image.shape[:2], threshold
202     )
203
204     # result is (height+2*padding, width+2*padding, channels) shaped

```

```

205     result: np.ndarray[np.uint8] = frequencyDomainFiltering(padded_image, glf)
206
207     if not strip_padding or padding == 0:
208         return result
209     else:
210         return result[padding:-padding, padding:-padding, :]
211
212         gaussianLowPassFiltering

```

위의 함수는 Gaussian Lowpass Filter를 주어진 이미지에 적용하는 함수이다. 필터는 `padding`, `threshold`에 의하여 변화하며, `strip_padding`은 Visualization을 위한 매개변수로, 기본값은 `True`며, 결과 이미지의 패딩을 제거할지를 결정한다.

- 주어진 이미지를 `padding`만큼 패딩하여 이미지를 준비한다.
- 이때, 사용하는 `DEFAULT_BORDER_TYPE`는 `cv2.BORDER_REPLICATE`이다.
- 해당 함수는 Ideal Lowpass Filter를 사용하므로, `gaussianLowPassFilter`를 이용해 필터를 생성 후, `frequencyDomainFiltering`을 이용해 필터링을 진행한다.
- 이후 결과 이미지를 `strip_padding`이나 `padding`에 따라 패딩을 제거하거나 제거하지 않은 상태 그대로 반환한다.
- `padding`이 0일 경우에는 `result[padding:-padding, padding:-padding, :]`에서 빈 이미지를 만드므로 따로 처리한다.

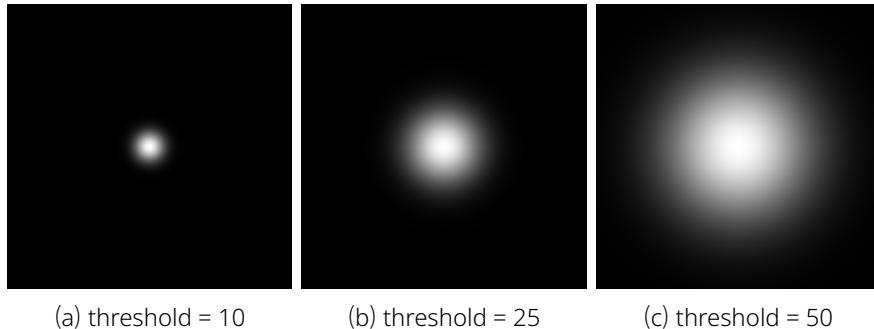
Visualization

Input Images & FFT of Input Images

입력 이미지는 이전 Problem 1과 동일하므로 이에 대한 시각화는 생략한다.

Gaussian Lowpass Filter in the Frequency Domain

다음은 Gaussian Lowpass Filter를 Frequency Domain에서 나타낸 이미지이다. 각 필터는 임의로 200px x 200px의 크기로 생성되었다.



Gaussian Lowpass Filter in the Frequency Domain

Filtering Result in the Frequency Domain

다음은 Gaussian Lowpass Filtering을 적용한 결과이다. 본문에는 각 이미지 별로 Lowpass Filtering이 잘 드러나는 이미지만을 첨부한다.

Result

아래는 Ideal Lowpass Filter의 Threshold에 따른 필터링 결과를 나타낸 것이다.



(a) color1 - padding=30,threshold=10

Filtered color1.jpg in Frequency Domain



Blue Channel

Green Channel

Red Channel

(b) frequency domain



(c) color3 - padding=30,threshold=10

Filtered color3.jpg in Frequency Domain

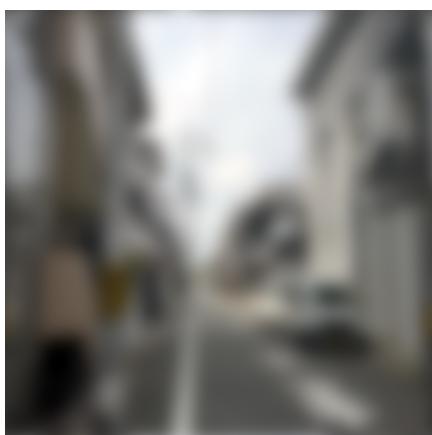


Blue Channel

Green Channel

Red Channel

(d) frequency domain



(e) color4 - padding=30,threshold=10

Filtered color4.jpg in Frequency Domain



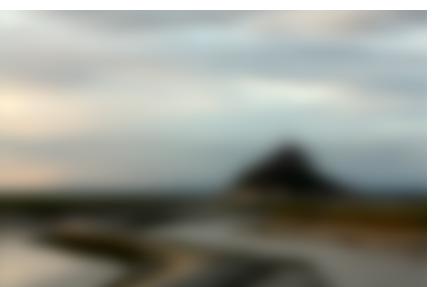
Blue Channel

Green Channel

Red Channel

(f) frequency domain

Result of Gaussian Lowpass Filtering in the Frequency Domain



(a) color3 - padding=30,threshold=10



(b) color3 - padding=30,threshold=25



(c) color3 - padding=30,threshold=50

Result of Gaussian Lospass Filtering on Threshold of Filter

Discussion

Ideal Lowpass Filter와 Gaussian Lowpass Filter는 아래와 같이 Ringing Effect 혹은 Artifact의 차이점이 존재한다.



(a) Ideal - padding=30,threshold=10

(b) Gaussian - padding=30,threshold=25

Comparison of Ideal and Gaussian Lowpass Filter

왼쪽의 Ideal Lowpass Filtering의 경우에는 Ringing Artifact를 관찰할 수 있으나, 오른쪽의 Gaussian Lowpass Filtering의 경우에는 그러한 현상이 보이지 않는다. 이는 수업자료에서 확인할 수 있는 Ideal Lowpass Filter와 Gaussian Lowpass Filter의 Spatial Response 차이가 원인이 된다. Ideal Lowpass Filter의 Spatial Response는 Ripple을 만들어 내지만 Gaussian Lowpass Filter의 경우에는 Spatial Response 또한 Gaussian Distribution을 보여준다. 이로 인해, Ideal Lowpass Filter는 Ringing Artifacts를 만들어내고, Gaussian Lowpass Filter는 그렇지 않다.

Problem 3

Algorithm - Primitives

```
216 def gauss(  
217     n: int,  
218     sigma: float,  
219 ) -> "np.ndarray[np.float32]":  
220     """Gaussian Distribution 1D Kernel  
221  
222     Args:  
223         n (int): Size of kernel  
224         sigma (float): Standard deviation of gaussian distribution  
225  
226     Returns:  
227         np.ndarray[np.float32]: 1D gaussian distribution kernel  
228     """  
229     r: np.ndarray[np.float32] = np.arange(n, dtype=np.float32) - (n - 1) / 2  
230     r = np.exp(-(r**2) / (2 * sigma**2))  
231     return r / r.sum()
```

Primitive - gauss

Gaussian Distribution을 가지는 1차원 벡터를 만들어 반환한다.

```
235 def gauss2d(  
236     shape: tuple[int, int],  
237     sigma: float,  
238 ) -> "np.ndarray[np.float32]":  
239     """Gaussian Distribution 2D Kernel  
240
```

```

241     Args:
242         shape (tuple[int, int]): Shape of kernel
243         sigma (float): Standard deviation of gaussian distribution
244
245     Returns:
246         np.ndarray[np.float32]: 2D gaussian distribution kernel
247         """
248         g1: np.ndarray[np.float32] = gauss(shape[0], sigma).reshape(shape[0], 1)
249         g2: np.ndarray[np.float32] = gauss(shape[1], sigma).reshape(1, shape[1])
250     return np.matmul(g1, g2)

```

Primitive - gauss2d

`gauss` 함수를 이용하여 2D Gaussian Kernel을 만들어 반환한다.

```

252 def psf2otf(
253     filter: "np.ndarray[np.float32]",
254     shape: tuple[int, int],
255 ) -> "np.ndarray[np.complex128]":
256     """Pad and shift the filter, then return with result of FFT of it
257
258     Args:
259         filter (np.ndarray[np.float32]): psf, filter
260         shape (tuple[int, int]): desired shape of output
261
262     Returns:
263         np.ndarray[np.complex128]: 2d numpy array otf
264         """
265
266     top = filter.shape[0] // 2
267     bottom = filter.shape[0] - top
268     left = filter.shape[1] // 2
269     right = filter.shape[1] - left
270
271     psf = np.zeros(shape, dtype=filter.dtype)
272
273     psf[:bottom, :right] = filter[top:, left:]
274     psf[:bottom, shape[1] - left :] = filter[top:, :left]
275     psf[shape[0] - top :, :right] = filter[:top, left:]
276     psf[shape[0] - top :, shape[1] - left :] = filter[:top, :left]
277
278     # return otf
279     return np.fft.fft2(psf)

```

Primitive - psf2otf

위의 코드는 주어진 필터를 주어진 `shape`에 맞추어 패딩한 후, 필터를 shift하여 fft한 결과를 반환한다.

Algorithm - Main Process

```

282 def unsharpMasking(
283     image: "np.ndarray[np.uint8]",
284     alpha: float,
285     padding: int,
286     sigma: float,
287     domain: str,
288 ) -> "np.ndarray[np.uint8]":
289     """Unsharp Masking with Various?(`spatial`, `frequency`) Domains
290

```

```

291 Unsharpening is formulated like below
292
293 I' = I + A * (I - F * I)
294
295 I for input image , I' for result image
296
297 A for sharpening strength alpha
298
299 F for low-pass filter, usually it is gaussian filter
300
301 Args:
302     image (np.ndarray[np.uint8]): Image
303     alpha (float): Sharpening strength
304     padding (int): Padding of the image, filter size will be `padding * 2 + 1`
305     sigma (float): Standard deviation of gaussian filter
306     domain (str): Domain to perform the unsharpening, it can be spatial or frequency
307
308 Returns:
309     np.ndarray[np.uint8]: Unsharpened Image
310 """
311 assert len(image.shape) == 3 # Colored Image
312 assert image.shape[2] == 3 # RGB
313 assert domain in ["spatial", "frequency"]
314 assert padding >= 0
315 assert sigma > 0 # valid sigma
316
317 filter_size: int = 2 * padding + 1
318 padded_image: np.ndarray[np.uint8] = cv2.copyMakeBorder(
319     image,
320     padding,
321     padding,
322     padding,
323     padding,
324     DEFAULT_BORDER_TYPE,
325 )
326 filter: np.ndarray[np.float32] = gauss2d((filter_size, filter_size), sigma)
327
328 result_image: np.ndarray[np.uint8]
329
330 if domain == "frequency":
331     # filter on frequency domain
332     filter_f: np.ndarray[np.complex128] = psf2otf(filter, padded_image.shape[:2])
333
334     result_channels: list["np.ndarray[np.uint8]"] = []
335
336     for channel in cv2.split(padded_image):
337         # move on frequency domain
338         channel_f: np.ndarray[np.complex128] = np.fft.fft2(channel)
339
340         result_f: np.ndarray[np.complex128] = channel_f + alpha * (
341             channel_f - filter_f * channel_f
342         )
343
344         result: np.ndarray[np.float64] = np.real(np.fft.ifft2(result_f))
345         result = np.clip(
346             result, 0, 255
347         )
348         result: np.ndarray[np.uint8] = result.astype(np.uint8)[
349             padding:-padding, padding:-padding

```

```

350     ]
351
352     result_channels.append(result)
353
354     result_image = cv2.merge(result_channels)
355 else: # domain == "spatial"
356     height: int
357     width: int
358     channel: int
359
360     height, width, channel = image.shape
361     filtered_image: np.ndarray[np.float32] = np.zeros_like(image).astype(np.float32)
362
363     for i in range(height):
364         for j in range(width):
365             for k in range(channel):
366                 filtered_image[i, j, k] = np.sum(
367                     padded_image[i : i + filter_size :, j : j + filter_size, k]
368                     * filter
369                 )
370
371     result: np.ndarray[np.float32] = image + alpha * (image - filtered_image)
372
373     result_image = np.clip(result, 0, 255).astype(np.uint8)
374
375 return result_image

```

unsharpMasking

위의 함수는 주어진 이미지를 명시된 `domain`에 맞추어서 Unsharpening을 진행하는 함수이다.

- `assert`를 이용하여 이미지의 형태, `domain`의 종류, 패딩과 `sigma`의 유효성을 검사한다.
- `padding`을 이용하여 주어진 이미지에 패딩을 진행한다.
- 필터는 Lowpass Filter인 Gaussian Lowpass Filter를 사용한다.
- `domain`이 "frequency"인 경우에는 필터를 `psf2otf`를 이용해 frequency domain으로 변환 후, 이미지 또한 frequency domain으로 전환하여 곱한 이후, unsharpening 식에 맞추어 연산 후 다시 Spatial Domain으로 돌려 반환한다.
- `domain`이 "spatial"인 경우에는 Spatial Domain의 Gaussian Lowpass Filtering을 진행한 후, unsharpening의 식에 맞추어 연산을 진행한다. 연산 이후 픽셀 표현 범위를 넘어갈 수 있으므로 `np.clip`를 이용해 값을 절삭한다.

Result

Input Images

입력 이미지는 이전 Problem 1과 동일하므로 생략한다.

Output

결과 이미지 중에서 효과가 잘 드러나는 이미지만을 첨부한다. 파라미터는 아래와 같다.

- Alpha: 3.0
- Padding: 50
- Sigma: 8.3

(전체 이미지들은 `images/output/problem3`에서 확인할 수 있다.)



(a) color1



(b) color3



(c) color4

Unsharpening in Spatial Domain



(a) color1



(b) color3



(c) color4

Unsharpening in Frequency Domain

Discussion

Effects of the parameters

아래는 alpha 값에 따른 결과를 나타낸 것이다. alpha 값이 증가함에 따라 경계값이 더욱 뚜렷해보임을 확인할 수 있다.



(a) alpha=1.0



(b) alpha=2.0



(c) alpha=3.0

Effect of Alpha(padding=50,sigma=8.3,domain=frequency)

Gaussian Lowpass Filter의 Sigma는 수업자료에 따라 padding/6.0의 크기로 설정하여 필터링을 진행하였다.

아래는 padding 값에 따른 결과를 나타낸 것이다. 위에서 언급한대로, sigma는 padding에 의존하도록 설정되었다. sigma의 값이 증가함에 따라 더욱 뚜렷한 결과를 보이는 것을 확인할 수 있다.



Effect of Sigma(Padding)(alpha=3.0,domain=frequency)

아래는 domain 값에 따른 결과를 나타낸 것이다. 큰 차이를 확인할 수 없으며, Convolution Theorem을 확인할 수 있다.

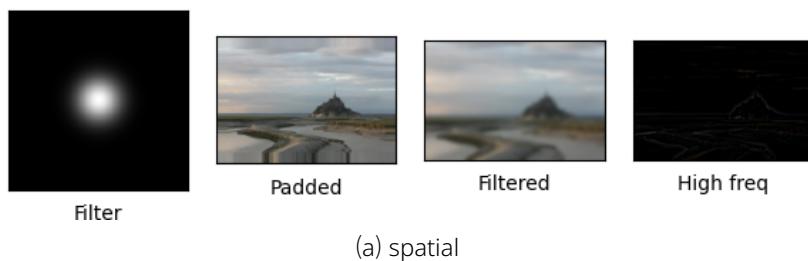


Effect of Domain(padding=50,alpha=3.0,sigma=8.3)

Result of Each Step - Spatial Domain

다음은 Spatial Domain에서 Unsharpening에 따른 각 단계를 나타낸 그림이다.

Spatial Domain Unsharpening



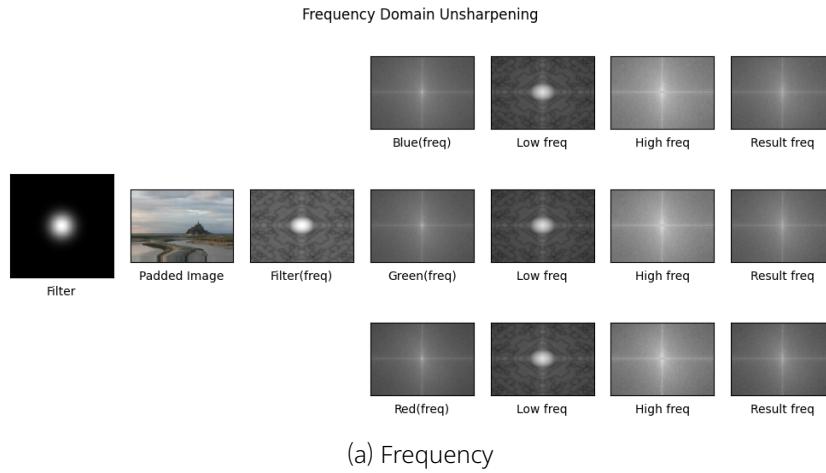
(a) spatial

Each Steps of Unsharpening - Spatial Domain

패딩된 이미지에 필터를 이용하여 Spatial Domain에서 Lowpass Filtering 진행 이후 해당 이미지를 기준 이미지에서 빼서 마지막 이미지의 High Frequency 이미지를 만든다. High Frequency 이미지를 더해 Unsharpening을 진행 한다.

Result of Each Step - Frequency Domain

다음은 Frequency Domain에서 Unsharpening에 따른 각 단계를 나타낸 그림이다.



Each Steps of Unsharpening - Frequency Domain

패딩된 이미지에서 각 채널별로 Frequency Domain으로 변환 후, Frequency Domain으로 변환된 Gaussian Filter를 적용해 Low Frequency를 필터링한다. 기존의 각 채널에서 Low Frequency 채널 이미지를 빼 High Frequency 채널을 얻은 이후, 이를 각 채널에 더해 Unsharpening을 한다. 마지막으로 Spatial Domain으로 변환하여 마무리한다.

Comparison Between the Spatial-Domain and Frequency-Domain

두 domain의 연산 시간 차이는 `padding` 크기 차이에 의해 발생한다. 각 domain의 시간복잡도를 계산하면 다음과 같다.

- **Spatial Domain:** 각 픽셀별로 필터의 픽셀 수만큼 곱셈 연산이 필요하다. 이후 Unsharpening에 따른 연산은 전체 이미지의 픽셀 수에 비례하는 계산 횟수를 가진다. 이에 따라 전체 시간 복잡도는 $O(MN)$ 이 된다. 이때, M 과 N 은 각각 필터의 픽셀 수, 이미지의 픽셀 수가 된다.
- **Frequency Domain:** Frequency Domain으로의 변환과 역변환에 대해 `numpy`에 의하면 $O(N \log N)$ 이 소요된다. Domain 변환 이후에는 연산마다 전체 이미지의 픽셀 수만큼 연산이 소요된다. 따라서 전체 과정을 정리하면 $O(N \log N + N) = O(N \log N)$ 의 시간복잡도가 계산된다.

다음은 Domain에 따른 실제 연산에 걸린 시간을 정리한 표이다. (`Color4.jpg`, `alpha=3.0`, `padding=50`, `sigma=8.3`) Spatial Domain에서의 Unsharpening이 더 오래걸리는 것을 확인할 수 있다.

Domain	T
Spatial	270.14
Frequency	3.50

Computational Time Depending on Domain

다음은 `padding`에 따른 연산 시간을 정리한 표이다. (`Color4.jpg`, `alpha=3.0`) Spatial Domain의 경우 `padding` 크기가 증가함에 따라 이미지에 대해 처리해야 할 픽셀의 수가 늘어나 연산 시간 또한 늘어나는 것을 확인할 수 있다. Frequency Domain 또한 증가하기는 하지만, Spatial Domain 대비 그 폭이 적은 것을 확인할 수 있다.

Domain	T_{10}	T_{25}	T_{50}
Spatial	78.49	118.48	270.14
Frequency	3.44	3.49	4.63

Computational Time Depending on Padding

`alpha` 값의 경우 연산 시간에 영향을 미치지 않는다.