

画像処理・画像処理工学 レポート課題 2

画像処理工学科 学籍番号: 21239 組番号: 234 5E 氏名: 柳原 魁人

2026 年 1 月 24 日

1 課題 1

1.1 問題 1-1: メディアンカット量子化法

1.1.1 理論

ピクセル値をソートして目標色数のグループに均等分割し、各グループの中央値を代表色として設定する手法です。

1.1.2 計算・導出過程

図 A-1 に示す 4×5 画素の画像に対してメディアンカット量子化法を適用し、4 色に量子化します。

102	179	92	14	106
74	202	87	116	99
151	130	149	52	1
235	157	37	129	191

図 1 量子化前の 4×5 ピクセル画像 (数値表示)

全ピクセル値は以下の通りです。

102, 179, 92, 14, 106, 74, 202, 87, 116, 99, 151, 130, 149, 52, 1, 235, 157, 37, 129, 191

これをソートすると以下のようになります。

1, 14, 37, 52, 74, 87, 92, 99, 102, 106, 116, 129, 130, 149, 151, 157, 179, 191, 202, 235

これを 4 つのグループに均等分割（各 5 ピクセル）すると、以下の表のようになります。

グループ	ピクセル値	代表色 (中央値)
1	1, 14, 37 , 52, 74	37
2	87, 92, 99 , 102, 106	99
3	116, 129, 130 , 149, 151	130
4	157, 179, 191 , 202, 235	191

上記に基づく量子化前後の比較を図で示します。

102	179	92	14	106	(a) 量子化前	99	191	99	37	99
74	202	87	116	99		37	191	99	130	99
151	130	149	52	1		130	130	130	37	37
235	157	37	129	191		191	191	37	130	191
(グループ別色分け)					(代表色)					

図 2 量子化前 (左) と量子化後 (右) の比較

1.1.3 結果

問題1-1 限定色表示

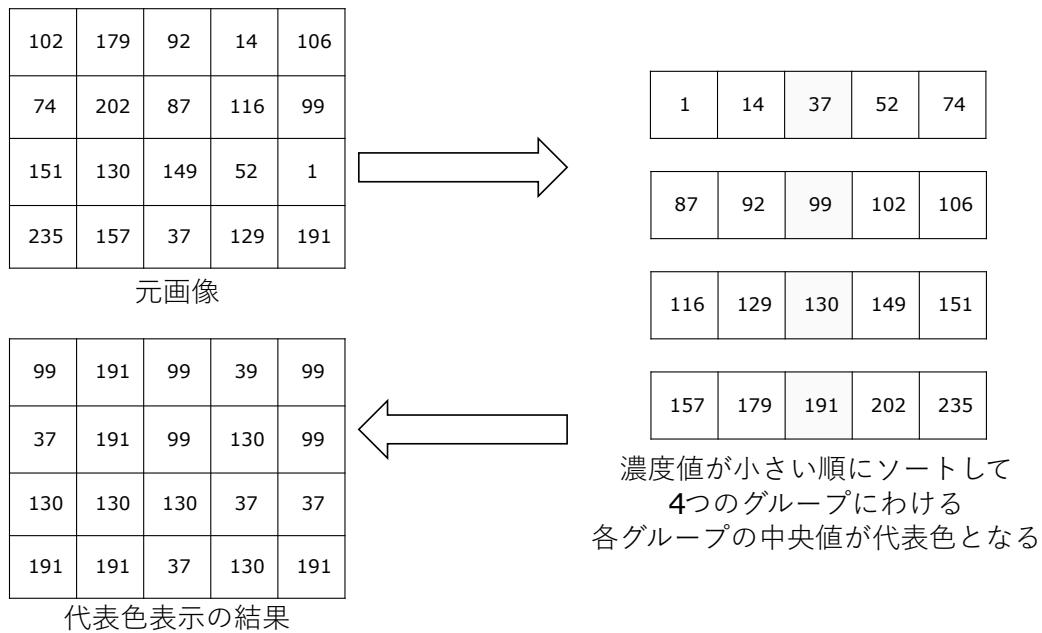


図3 メディアンカット量子化の結果

1.2 問題 1-2: ラベリング処理

1.2.1 理論

2 値画像で連結成分に同一ラベルを割り当てます。2 回走査法では、第 1 回で仮ラベル割当と等価関係記録を行い、第 2 回で統合します。

1.2.2 計算・導出過程

図 A-2 の 10×10 画素画像に対してラベリングを実行。

第 1 回走査では、左上から右下へ走査し、各白ピクセルについて左上・上・右上・左のピクセルを確認します。

- 全て黒 → 新規ラベル割当
- いずれかが白 → そのラベル継承
- 複数が異なるラベル → 最小値割当、等価関係記録

等価ラベル関係 : $A - B - C, E - F, D$

第 2 回走査 : 等価関係に基づいて代表ラベルに統一。

仮ラベル	等価関係	代表ラベル
A, B, C	$A - B - C$	A
D	単独	D
E, F	$E - F$	E

1.2.3 結果

問題1-2 ラベリング

1回目用								2回目用							
			A	A											
			A	A	A	A		B							
			A	A	A	A	A	A							
			A	A	A	A	A	A							
C	A	A	A	A	A				D						
		E		F		D	D	D							
		E	E	E			D								

同じ連結成分であると記録された組:
A-B-C, E-F, D

図4 ラベリング処理の結果

1.3 問題 1-3: ハフマン符号化

1.3.1 理論

高確率シンボルに短い符号を割り当てる可変長符号化を行います。ハフマン木を構築することで符号を生成します。

1.3.2 計算・導出過程

8個のシンボルの出現確率を以下に示します。

シンボル	確率	符号
0	0.30	10
1	0.02	00001
2	0.06	0011
3	0.04	0001
4	0.01	00000
5	0.05	0010
6	0.20	01
7	0.32	11

結合ステップを低い確率から順に実行すると以下の通りです。

1. $4(0.01)+1(0.02)=0.03 \rightarrow A$
2. $A(0.03)+3(0.04)=0.07 \rightarrow B$
3. $5(0.05)+2(0.06)=0.11 \rightarrow C$
4. $B(0.07)+C(0.11)=0.18 \rightarrow D$
5. $D(0.18)+6(0.20)=0.38 \rightarrow E$
6. $0(0.30)+7(0.32)=0.62 \rightarrow F$
7. $E(0.38)+F(0.62)=1.00 \rightarrow G$

平均符号長計算：

$$\begin{aligned}
 L &= 0.30 \times 2 + 0.02 \times 5 + 0.06 \times 4 + 0.04 \times 4 \\
 &\quad + 0.01 \times 5 + 0.05 \times 4 + 0.20 \times 2 + 0.32 \times 2 \\
 &= 2.39 \text{ ビット}
 \end{aligned}$$

等長符号（3ビット）との比較：削減量 = $3 - 2.39 = 0.61$ ビット（約 20% 削減）

1.3.3 結果

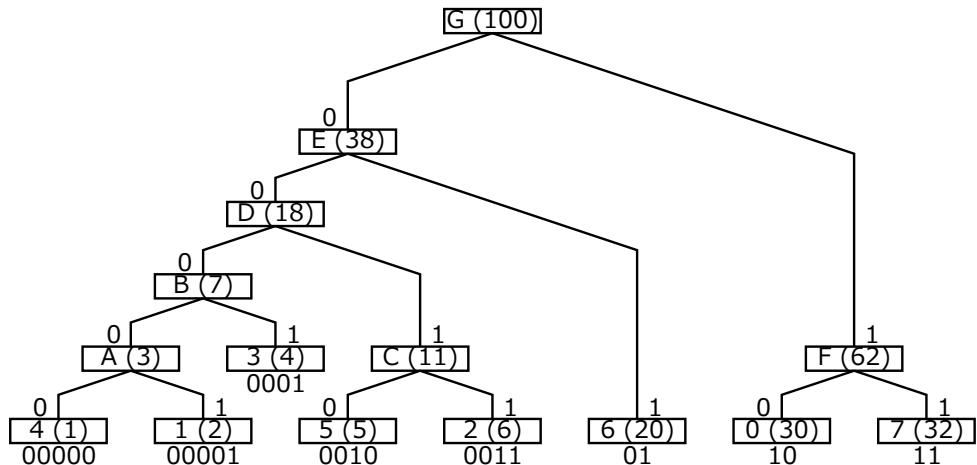


図 5 ハフマン木の構造

2 課題 2

3 問題 1: モルフォロジー処理によるノイズ除去

3.1 概要

開処理と閉処理を用いてノイズを除去。

3.1.1 理論

開処理（収縮→膨張）で明色の孤立ノイズを除去し、閉処理（膨張→収縮）で暗色ノイズを埋める。本実装では矩形カーネルと楕円カーネルを用い、カーネルサイズ k を 3–15（奇数）で走査し、矩形→楕円の順で開処理を二段適用した。

3.2 結果

開処理 (Opening = 収縮→膨張) により、白色の孤立ノイズが除去された。収縮処理で小領域が消失し、その後の膨張で主要な図形領域が復元された。本画像では白色ノイズが優勢であったため開処理を採用した。実験により、矩形構造要素 (MORPH_RECT) は十字構造要素 (MORPH_CROSS) よりも孤立した小ノイズの除去に有効であることが示された。実践手順として、矩形カーネルによる開処理の後に楕円形カーネル (MORPH_ELLIPSE) で追処理を行うことで、矩形処理後に残存する微小ノイズをさらに低減できることが確認された。矩形→楕円の順で処理を行うと形状保持と雑音除去のバランスが良好となる。閉処理 (Closing = 膨張→収縮) は黒色ノイズの埋め込みに有効であるが、本課題の対象画像では適用しなかった。

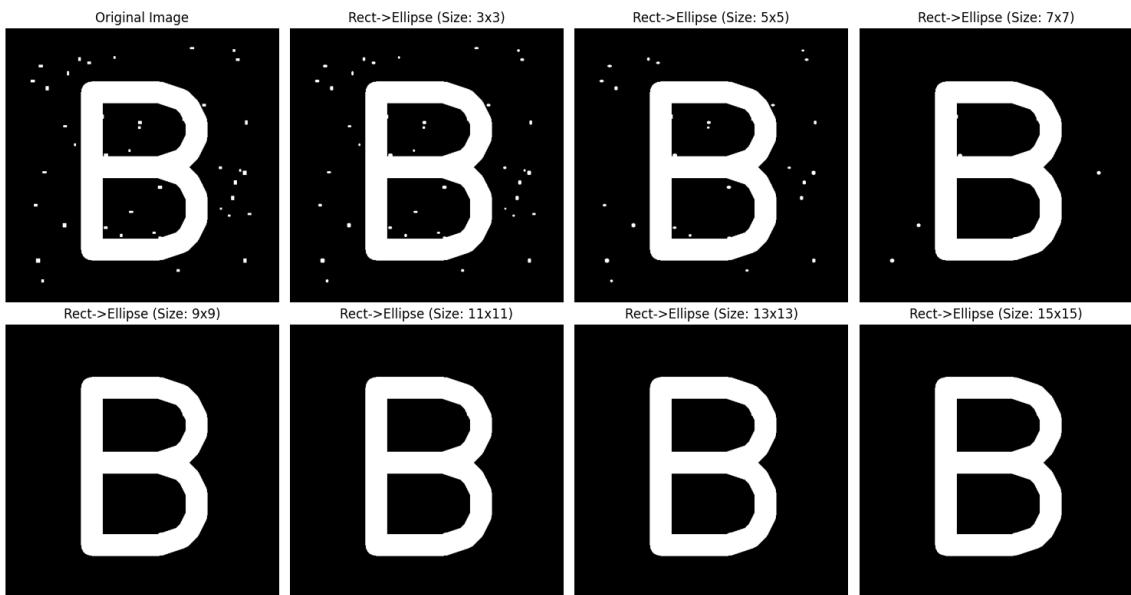


図 6 Kernel size 15 での開閉処理結果（ノイズ除去後の 2 値画像）

3.2.1 考察

矩形→楕円の二段開処理で細粒ノイズが安定して除去できた。 $k = 9$ 付近がノイズ除去と形状保持の折衷点であり、 $k = 15$ では効果が飽和し変化が小さい。なお、黒ノイズ主体の画像に対しては閉処理の適用が有効である。

4 問題 2: JPEG 品質と圧縮率の関係

4.1 概要

JPEG 品質と圧縮率、SSIM 値の関係を調査。

4.1.1 理論

品質パラメータ Q を 0–100 で設定し、元 PNG サイズ S_0 と圧縮後サイズ S_Q からサイズ比 $100 \times S_Q/S_0$ を算出。画質は RGB で構造類似度指数 (SSIM, `win_size=3, channel_axis=2, data_range=255`) を評価した。

4.2 結果

下表は本レポートで得られた数値（品質 Q 、サイズ比、SSIM）を整理したもの。

表 1 品質 Q によるサイズ比と SSIM

Q	サイズ比 (%)	SSIM
0	0.9	0.327
10	2.2	0.520
20	3.6	0.609
30	4.8	0.656
40	5.8	0.686
50	6.7	0.709
60	7.7	0.729
70	9.3	0.753
80	11.8	0.783
90	17.7	0.825
100	45.5	0.871



図 7 JPEG 品質 Q ごとの視覚比較とサイズ比・SSIM（提示画像）

4.2.1 考察

SSIM は Q に単調増加し、**Q=80** で 0.783、**Q=90** で 0.825 となるがサイズ比は 11.8% → 17.7% へ増加する。視覚・数値の両面から実用的には **Q=80～90** が現実的で、**Q=100** は 45.5% まで肥大し利得が小さい。容量重視なら **Q=10**（サイズ比 2.2%、SSIM 0.520）も用途次第で許容される。

5 問題 3: 2 次元 FFT と振幅スペクトル

5.1 概要

グレースケール画像を 2 次元 FFT で周波数領域に変換し、振幅スペクトルを分析。

5.1.1 理論

各入力画像を中心から最大の正方形にトリミングし、2 次元 FFT 後に `fftshift` で低周波を中心化。振幅スペクトルは $20 \log_{10}(|F| + 1)$ の対数スケールで可視化した。

5.2 結果

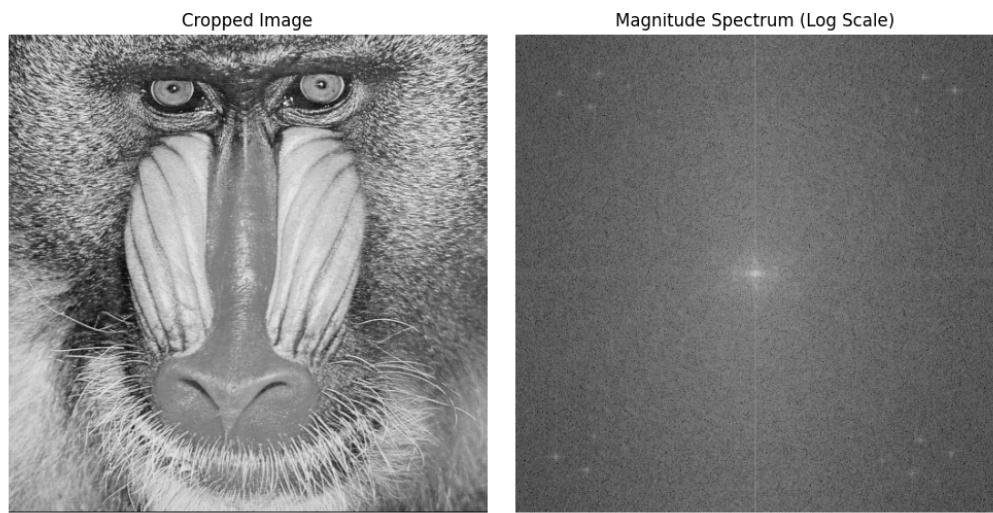


図 8 グレースケール画像の 2 次元振幅スペクトル（中央化）

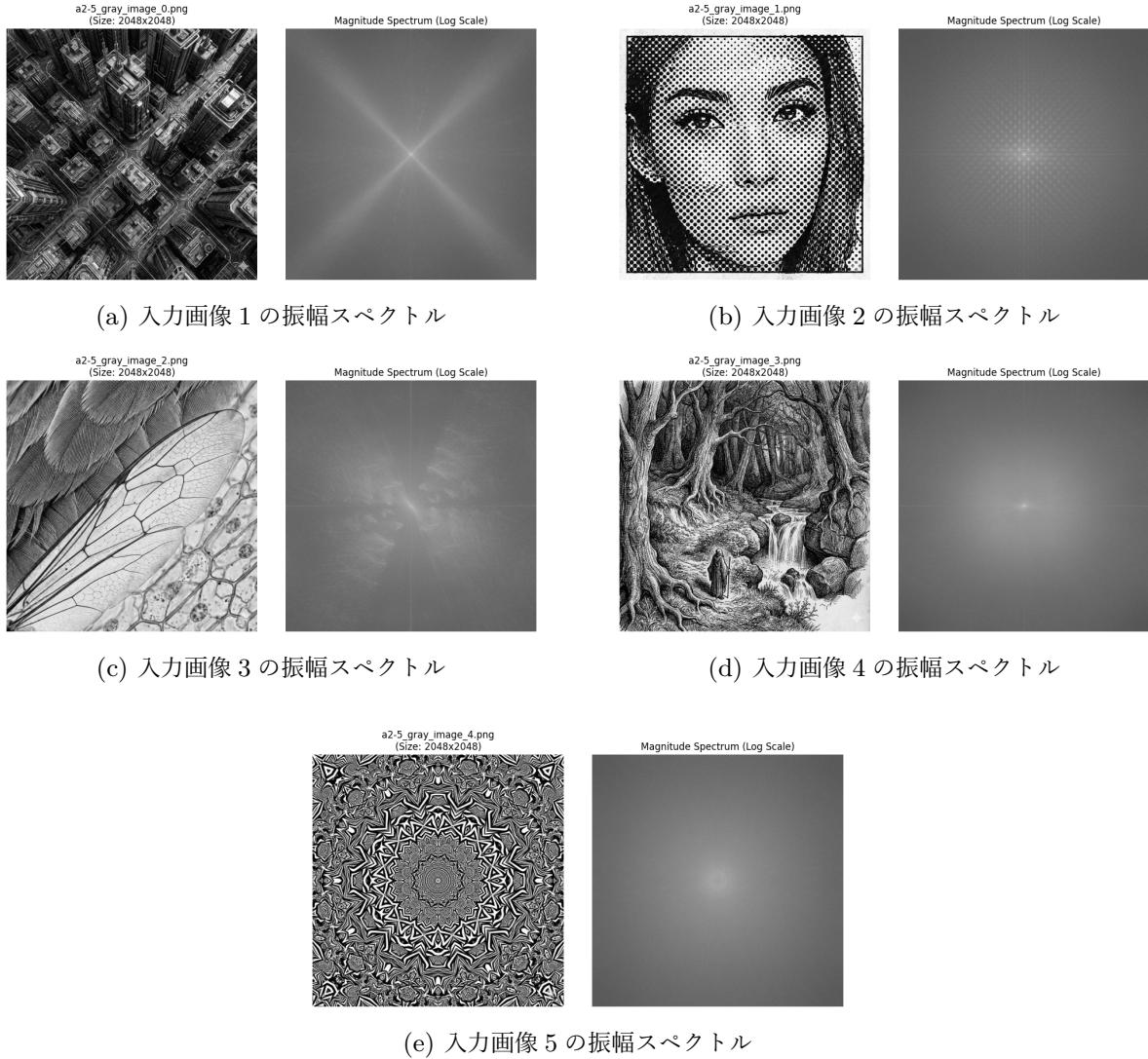


図9 同一の2次元FFT処理を別入力画像に適用した結果（左上→右下：(a)～(e))。

図9は、図8と同一の処理を別入力画像に適用した結果である。各サブ図はそれぞれ異なる入力画像に対する結果を示す。

5.2.1 考察

周期性の強い幾何パターンでは明瞭な格子状ピークが現れ、細かい線描は高周波まで広がるスペクトルとなった。自然テクスチャはエネルギーが中心寄りに分布し、モアレ風パターンでは対角方向のピークが強調された。

■画像タイプ別プロンプト例 今回のレポートで示した、特に問題2-3で処理対象とした入力画像は、以下に示すプロンプトを用いて NanoBananaPro で生成したものです。

パターン1：幾何学的・周期的な構造

Prompt:

Hyper-intricate kaleidoscope pattern, complex geometric fractal, high contrast black and white, mathematical symmetry, sharp edges, detailed

mandala, optical illusion art, 8k resolution --ar 1:1

パターン2：細かい線描・版画調

Prompt:

Detailed cross-hatching illustration in the style of Gustave Doré,
antique etching of a dense forest, intricate ink lines, engraving
style, high detail, sharp texture, masterpiece --ar 1:1

パターン3：人工的なグリッド・都市・回路

Prompt:

Aerial top-down view of a futuristic cyber city, dense skyscrapers,
intricate circuit board texture, metallic surfaces, high contrast,
greeble details, sci-fi architecture, sharp focus --ar 1:1

パターン4：自然界のランダムなテクスチャ

Prompt:

Macro photography of intricate bird feathers, extreme close-up of
insect wing texture, biological cells patterns, fibrous texture, sharp
details, high texture quality, monochrome photography --ar 1:1

パターン5：ハーフトーン・モアレ（ドット）

Prompt:

Halftone dot pattern portrait, pop art style, comic book shading,
distinct dots, moire pattern effect, glitch art, high contrast,
monochrome --ar 1:1

■生成時のコツ

- ”Intricate”、”Detailed”、”Dense”、”High contrast”といった語を含めると、周波数的に特徴が出やすくなる。
- コントラストを高めるとエッジ成分が強調され、FFT結果が見やすくなる。

6 問題4: 周波数フィルタの応用

6.1 概要

理想的ローパスフィルタ (ILPF) とガウシアンハイパスフィルタ (GHPF) を、カットオフ周波数 D_0 を 10/30/60 と変化させ適用し、効果を比較。

6.1.1 理論

ILPF は周波数距離 $D \leq D_0$ を 1、その他を 0 とするマスク $H_{ILPF}(D)$ を掛ける。GHPF は $H_{GHPF}(D) = 1 - \exp\{-D^2/(2D_0^2)\}$ で低周波を抑圧し高周波を通過させる。いずれも `fftshift` 後の周波数平面でマスクを乗算し、逆 FFT で復元した。

6.2 結果

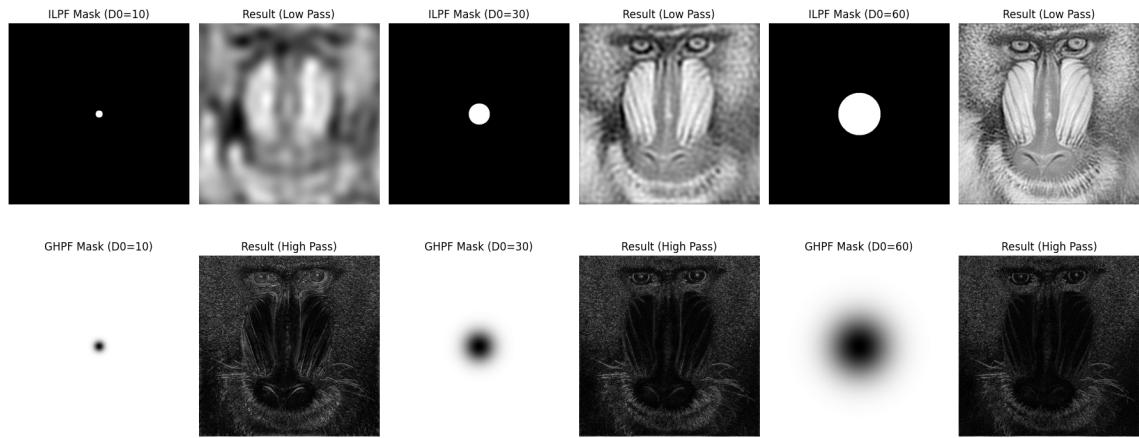


図 10 ローパスフィルタとハイパスフィルタの適用結果比較（代表例）

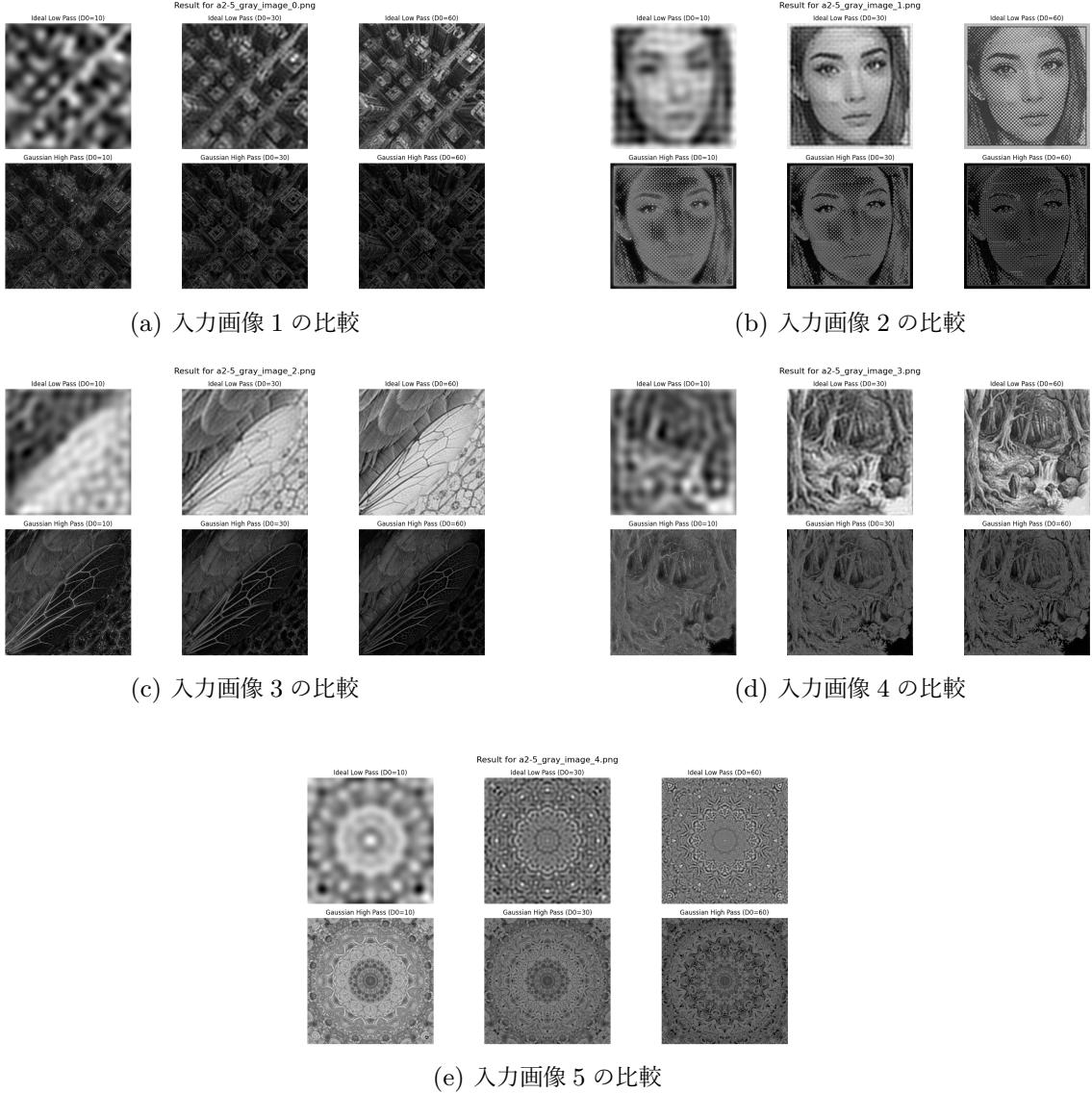


図 11 同一のフィルタ処理を別入力画像で試した結果 (左上→右下: (a)~(e))。

図 11 は、図 10 と同一の処理を別入力画像に適用した結果である。各サブ図に示したカットオフ周波数 D_0 はサブキャプション ((a)~(e)) に記載してある。

ローパスフィルタ (ILPF) : 低周波成分のみを通し、高周波（ノイズ・細部）を除去。結果は全体的にぼかされ、目や顔の大型構造は保持される一方、毛並みなどの細かいテクスチャは消失。ノイズ除去やスムージングに有効。

ハイパスフィルタ (GHPF) : DC 成分と低周波を除去し、高周波のみを通す。結果はエッジが明るく浮き出ち、目・口・毛並みなどの細部が強調される。大きな領域は暗くなり、画像全体はコントラストが低下。エッジ検出と細部抽出に有効。

カットオフ周波数 D0 の影響: D_0 を 10/30/60 と変化させた結果、 D_0 が小さいほど通過する周波数帯域が狭まる。ローパスでは $D_0=10$ で極端なぼかし効果、 $D_0=60$ で細部を一定程度保持する。ハイパスでは $D_0=10$ でエッジ成分のみが抽出され、 $D_0=60$ では中間周波数も強調されより豊かなテクスチャが得られた。 D_0 の選択は目的（ノイズ除去 vs 細部保持）に応じて調整すべきである。

6.2.1 考察

ILPF は $D_0=10$ で強いぼかしとなり輪郭も鈍化、 $D_0=60$ で顔や毛並みの細部が残る。GHPF は $D_0=10$ でエッジ抽出的になり暗部が増えるが、 $D_0=60$ では中間周波も残り質感強調が得られた。目的がノイズ除去なら小さめ、質感強調なら大きめの D_0 が適切。

付録: プログラムリスト

問題1: モルフォロジー処理

```
1 # -*- coding: utf-8 -*-
2 """問題2_1.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import math
14
15 # 共通のディレクトリパス
16 common_path = '/content/drive/MyDrive/img2025/image/'
17 filename = 'a2-3_binary_image.png'
18
19 # 画像読み込み
20 img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
21
22 # カーネルサイズのリスト
23 kernel_sizes = list(range(3, 17, 2))
24 total_images = 1 + len(kernel_sizes)
25
26 # レイアウト計算
27 cols = 4
28 rows = math.ceil(total_images / cols)
29
30 # 実行結果の表示設定
31 plt.figure(figsize=(15, 4 * rows))
32
33 # オリジナル画像
34 plt.subplot(rows, cols, 1)
35 plt.title('Original Image')
36 plt.imshow(img, cmap='gray')
37 plt.axis('off')
38
39 # カーネルサイズを変えて処理
40 for i, k in enumerate(kernel_sizes):
41     # 1. 矩形カーネルでオープニング処理
42     kernel_rect = cv2.getStructuringElement(cv2.MORPH_RECT, (k, k))
43     img_rect = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel_rect)
```

```
44
45 # 2. その結果に楕円カーネルでオープニング処理
46 kernel_ellipse = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (k, k))
47 result = cv2.morphologyEx(img_rect, cv2.MORPH_OPEN, kernel_ellipse)
48
49 # 表示
50 plt.subplot(rows, cols, i + 2)
51 plt.title(f'Rect->Ellipse (Size: {k}x{k})')
52 plt.imshow(result, cmap='gray')
53 plt.axis('off')
54
55 plt.tight_layout()
56
57 # 画像を保存してダウンロード
58 save_filename = 'problem2_1_rect_ellipse.png'
59 plt.savefig(save_filename)
60 plt.show()
61 files.download(save_filename)
```

Listing 1 問題 1 モルフォロジー処理によるノイズ除去

問題 2: JPEG 品質と圧縮率

```
1 # -*- coding: utf-8 -*-
2 """問題2_2.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import os
14 from skimage.metrics import structural_similarity as ssim
15
16 # 共通のディレクトリパス
17 common_path = '/content/drive/MyDrive/img2025/image/'
18 filename = 'a2-4_color_image.png'
19
20 # 画像を読み込む
21 original_img = cv2.imread(common_path + filename)
22 original_size = os.path.getsize(common_path + filename)
23 original_img_rgb = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
24
25 # 画像表示の準備
26 plt.figure(figsize=(20, 15))
27
28 # 品質0から100まで10刻みでループ
29 for i, quality in enumerate(range(0, 101, 10)):
30     # JPEG圧縮保存（一時ファイル）
31     output_path = common_path + f'compressed_{quality}.jpg'
32     cv2.imwrite(output_path, original_img, [int(cv2.IMWRITE_JPEG_QUALITY), quality])
33
34     # 圧縮後のファイルサイズ取得と圧縮率計算
35     comp_size = os.path.getsize(output_path)
36     comp_ratio = (comp_size / original_size) * 100
37
38     # 画像読み込み
39     compressed_img = cv2.imread(output_path)
40     compressed_img_rgb = cv2.cvtColor(compressed_img, cv2.COLOR_BGR2RGB)
41
42     # SSIM計算
43     score = ssim(original_img_rgb, compressed_img_rgb, win_size=3, channel_axis=2,
44                   data_range=255)
45
46     # 表示
```

```
46 plt.subplot(3, 4, i + 1)
47 plt.imshow(compressed_img_rgb)
48 plt.title(f"Q={quality}, Size={comp_ratio:.1f}%, SSIM={score:.3f}")
49 plt.axis('off')
50
51 plt.tight_layout()
52
53 # 画像を保存してダウンロード
54 save_filename = 'problem2_2_result.png'
55 plt.savefig(save_filename)
56 plt.show()
57 files.download(save_filename)
```

Listing 2 問題 2 JPEG 品質と圧縮率の関係調査

問題 3: 2 次元 FFT と振幅スペクトル

```
1 # -*- coding: utf-8 -*-
2 """問題2_3.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 # 共通のディレクトリパス
15 common_path = '/content/drive/MyDrive/img2025/image/'
16
17 # 処理する画像ファイル名のリスト
18 filenames = [
19     'a2-5_gray_image_0.png',
20     'a2-5_gray_image_1.png',
21     'a2-5_gray_image_2.png',
22     'a2-5_gray_image_3.png',
23     'a2-5_gray_image_4.png',
24     'a2-5_gray_image.png'
25 ]
26
27 # ループで各画像を個別に処理・保存
28 for i, filename in enumerate(filenames):
29     # 画像を読み込む
30     gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
31
32     # 画像サイズから最大の正方形サイズを決定
33     h, w = gray_img.shape
34     crop_size = min(h, w)
35
36     # 中心から最大サイズでトリミング
37     start_y = h // 2 - crop_size // 2
38     start_x = w // 2 - crop_size // 2
39     cropped_img = gray_img[start_y:start_y+crop_size, start_x:start_x+crop_size]
40
41     # 2次元高速フーリエ変換 (2D-FFT)
42     fourier = np.fft.fft2(cropped_img)
43     fshift = np.fft.fftshift(fourier)
44
45     # 振幅スペクトル (対数スケール)
46     amp_spectrum = 20 * np.log10(np.abs(fshift) + 1)
```

```
47
48 # --- 表示と保存（1枚ずつ個別のFigureを作成） ---
49 plt.figure(figsize=(10, 5))
50
51 # 左側：トリミング画像
52 plt.subplot(1, 2, 1)
53 plt.title(f'{filename}\n(Size: {crop_size}x{crop_size})')
54 plt.imshow(cropped_img, cmap='gray')
55 plt.axis('off')
56
57 # 右側：振幅スペクトル
58 plt.subplot(1, 2, 2)
59 plt.title('Magnitude Spectrum (Log Scale)')
60 plt.imshow(amp_spectrum, cmap='gray')
61 plt.axis('off')
62
63 plt.tight_layout()
64
65 # 個別に画像を保存してダウンロード
66 save_filename = f'problem2_3_result_{i}.png'
67 plt.savefig(save_filename)
68 plt.show()
69 files.download(save_filename)
```

Listing 3 問題 3 2 次元 FFT と振幅スペクトル

問題 4: 周波数フィルタの応用

```
1 # -*- coding: utf-8 -*-
2 """問題2_4.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 # 共通のディレクトリパス
15 common_path = '/content/drive/MyDrive/img2025/image/'
16
17 # 処理する画像ファイル名のリスト
18 filenames = [
19     'a2-5_gray_image_0.png',
20     'a2-5_gray_image_1.png',
21     'a2-5_gray_image_2.png',
22     'a2-5_gray_image_3.png',
23     'a2-5_gray_image_4.png'
24 ]
25
26 # フィルタ生成関数の定義
27 def create_ideal_lowpass(shape, cutoff):
28     rows, cols = shape
29     crow, ccol = rows // 2, cols // 2
30     y, x = np.ogrid[:rows, :cols]
31     dist_sq = (x - ccol)**2 + (y - crow)**2
32     mask = np.zeros(shape)
33     mask[dist_sq <= cutoff**2] = 1
34     return mask
35
36 def create_gaussian_highpass(shape, cutoff):
37     rows, cols = shape
38     crow, ccol = rows // 2, cols // 2
39     y, x = np.ogrid[:rows, :cols]
40     dist_sq = (x - ccol)**2 + (y - crow)**2
41     mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
42     return mask
43
44 # 試行する遮断周波数リスト
45 cutoffs = [10, 30, 60]
46
```

```

47 # ループで各画像を個別に処理
48 for file_idx, filename in enumerate(filenames):
49     # 画像読み込み
50     gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
51
52     # 画像サイズから最大の正方形サイズを決定してトリミング
53     h, w = gray_img.shape
54     crop_size = min(h, w)
55     start_y = h // 2 - crop_size // 2
56     start_x = w // 2 - crop_size // 2
57     img = gray_img[start_y:start_y+crop_size, start_x:start_x+crop_size]
58
59     # 2次元FFTとシフト
60     fshift = np.fft.fftshift(np.fft.fft2(img))
61
62     # 表示設定 (列数 = 遮断周波数の数)
63     cols = len(cutoffs)
64     plt.figure(figsize=(15, 8))
65     plt.suptitle(f'Result for {filename}', fontsize=16)
66
67     # --- 上段：理想ローパスフィルタ ---
68     for i, d0 in enumerate(cutoffs):
69         # マスク作成と適用
70         mask_lp = create_ideal_lowpass(img.shape, d0)
71         img_lp = np.fft.ifft2(np.fft.ifftshift(fshift * mask_lp)).real
72
73         # 結果表示
74         plt.subplot(2, cols, i + 1)
75         plt.title(f'Ideal Low Pass (D0={d0})')
76         plt.imshow(img_lp, cmap='gray')
77         plt.axis('off')
78
79     # --- 下段：ガウシアンハイパスフィルタ ---
80     for i, d0 in enumerate(cutoffs):
81         # マスク作成と適用
82         mask_hp = create_gaussian_highpass(img.shape, d0)
83         img_hp = np.fft.ifft2(np.fft.ifftshift(fshift * mask_hp)).real
84
85         # 結果表示
86         plt.subplot(2, cols, cols + i + 1)
87         plt.title(f'Gaussian High Pass (D0={d0})')
88         plt.imshow(np.abs(img_hp), cmap='gray')
89         plt.axis('off')
90
91     plt.tight_layout()
92
93     # 画像を保存してダウンロード
94     save_filename = f'problem2_4_result_{file_idx}.png'
95     plt.savefig(save_filename)

```

```
96     plt.show()  
97     files.download(save_filename)
```

Listing 4 問題 4 周波数フィルタの応用