レポート提出票

科目名: 情報工学実験３

実験課題名: 課題４　画像変換

実施日: 2024年 7月 4日

学籍番号: 4622045

氏名: 小澤　翼

共同実験者:

1　要旨

空間フィルタリングの実装を行い、理解を深めることが出来た。

2　目的

畳み込み演算を用いた画像変換を実装することで空間フィルタリングについて理解すること。

3　課題 1

3.1　実験方法

1. 以下に挙げる 3 通りの方法で平均化フィルタを実装する

   - (a) **myaverage_naive(image, size)**：python の for 文を利用して、画像中のすべての画素について size*size ブロックの平均を計算する。OpenCV, scikit-image 等のライブラリ, numpy の関数 np.sum(), np.average(), np.mean() は使用しないで実装すること。4 重ループになる。
   - (b) **myaverage_numpy(image, size)**：python の for 文を利用して画像のすべての画素を走査するが、平均化の処理に numpy の np.mean() 関数を利用する。2 重ループになる。
   - (c) **myaverage_integral(image, filter_size)**：
     - 積分画像 (integral image) を用いて実装する。numpy の関数 np.pad(im, pad_width, mode="constant") は使用しても良い(詳しくはマニュアル参照)。ただし np.sum(), np.average(), np.mean() は使用しないこと。
     - scikit-image にも積分画像を求める関数 skimage.transform.integral_image(), skimage.transform.integrate() が存在する。

2. 実装した関数 (a),(b),(c) の出力結果が、scikit-image で実装した結果と一致することを確認する。また、具体的には平均二乗誤差 MSE が 1 未満であることを確認する。

3. 画像の拡大率 m を変化させながら、平均化フィルタ関数 (a), (b), (c) の処理時間を計測し、グラフを描いて比較するとともに、計算量の観点から考察する。

4. フィルタサイズ w（フィルタの幅・高さは w×w）を変化させながら、平均化フィルタ関数 (a), (b), (c) の処理時間を計測し、グラフを描いて比較するとともに、計算量の観点から考察する。

5. 平均化フィルタが分離可能フィルタ (separable filter) であることを用いて、以下の平均化フィルタ関数を実装し、scikit-image と結果が一致することを確認する。

   - (d) **myaverage_separable(image, filter_size)**

6. 画像サイズ及びフィルタサイズを変化させながら、平均化フィルタ (a)-(d) の処理時間を計測することで、処理時間を比較し計算量の観点から考察する。

3.2 実験結果

3.2.1 関数 (a),(b),(c) の平均二乗誤差 (MSE)

表 1: 平均二乗誤差 (MSE)

Method	MSE
(a)	$6.02001923766 \times 10^{-28}$
(b)	$5.21424278804 \times 10^{-28}$
(c)	$9.03145818362 \times 10^{-21}$

この表 1 より、平均二乗誤差 MSE がすべての方法で 1 未満となったため scikit-image で実装した結果と一致する。

3.2.2 画像の拡大率 m を変化

画像の拡大率 m の $[0.5, 1, 1.5, 2, 2.5, 3]$ を横軸に、その拡大率に対して要した実装時間を横軸にしたグラフは以下の図 1 の通りである (フィルタサイズは 5 である)。

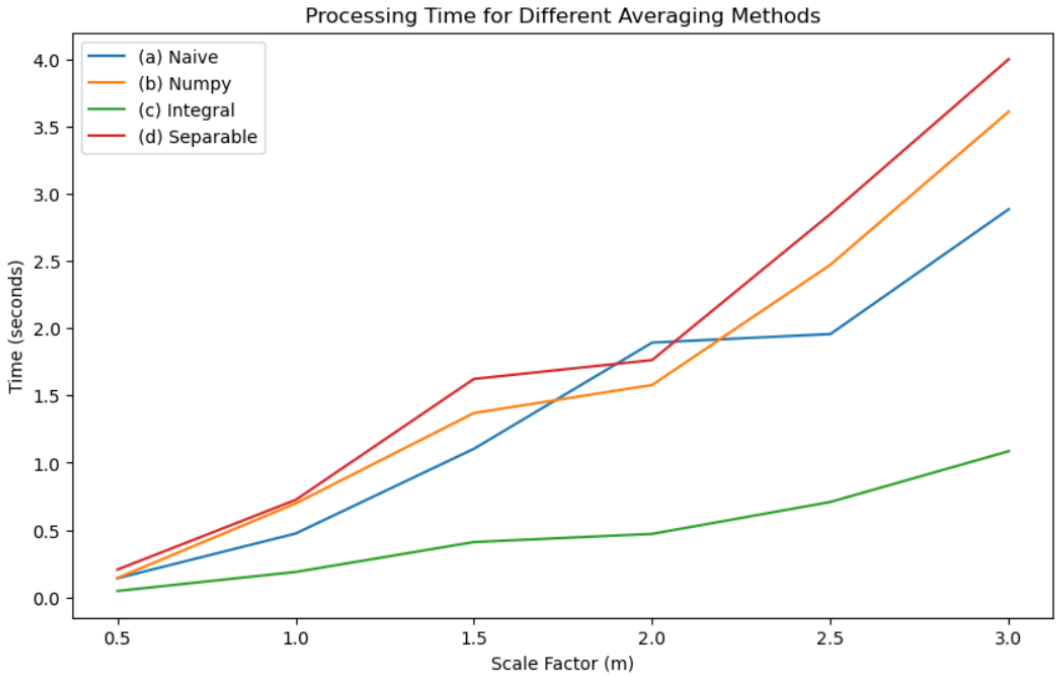

図 1: 各方法の実装時間

3.2.3 フィルタサイズ w を変化

フィルタサイズ w の $[1, 3, 5, 7, 9]$ を横軸に、そのフィルタサイズに対して要した実装時間を横軸にしたグラフは以下の図 2 の通りである。

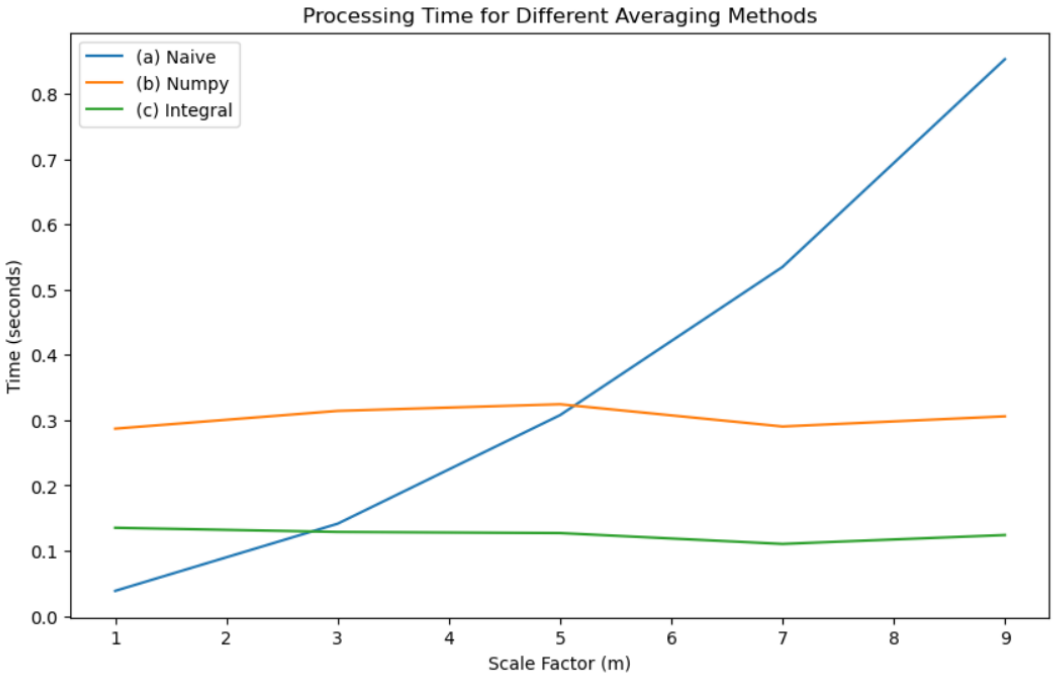

図 2: 各方法の実装時間

3.3 検討・考察

各方法の計算量は以下の表 2 の通りである (m は画像の高さ、n は画像の幅、k はフィルタサイズ)。

表 2: 計算量

Method	Time Complexity
(a)	$O(m \times n \times k^2)$
(b)	$O(m \times n)$
(c)	$O(m \times n)$
(d)	$O(m \times n \times m \times n)$

図 1 と表 2 を比較すると、図 1 では実装に要した時間は $(d) \geq (b) \geq (a) \geq (c)$ の順となっている。しかし、表 2 の計算量から考えると $(d) \geq (a) \geq (b) \geq (c)$ か $(d) \geq (a) \geq (c) \geq (b)$ とならなければならない。この違いが生まれてしまった原因として考えられるのは、np.mean() 関数の計算量がフィルタサイズの計算量を超えてしまったためだと思われる。なので、フィルタサイズが小さい時は (b) の方法を使用するより (a) の方法を用いた方が良い。ただし、図 1 から見て分かる通り、(c) の方法が一番効率が良いので、条件が無い限り (c) の方法を用いて平均化フィルタを行うことを推奨する (精度は (a), (b) に比べて少し劣る (表 1 参照))。

3.3.1 拡大率 m を変化させたとき

図 1 から見て分かるように拡大率 m を変化させたとき、(a),(b),(d) の方法は拡大率の影響をかなり受けているが、(c) の方法では (a),(b),(d) の方法に比べてあまり受けていない。この原因

は、(c) の方法のみ積分画像の作成に一次計算量がかかるだけで、その後のフィルタリングは定数時間で行えるため、あまり影響を受けないのではないかと考える。

3.3.2 フィルタサイズ w を変化させたとき

図 2 を見て分かる通り (b),(c) の方法は全く影響を受けていない一方で、(a) の方法はかなり影響を受けている。これは表 2 を見れば一目瞭然で、(a) の方法にはフィルタサイズが影響を及ぼす計算量になっていて (b),(c) はなっていない。これが原因だと考えられる。

4 まとめ

さまざまなフィルタリングを実装することで、どの方法が精度が高いか、どの方法が実装時間がが短いか知ることが出来た。この知識を今後活かしていければ良いと感じた。

A ソースコード

課題 1: 平均化フィルタ

```
import time
import numpy as np
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.transform import resize
from scipy.ndimage.filters import correlate
import matplotlib.pyplot as plt
%matplotlib inline

def mse(y1, y2):
    return ((y1 - y2)**2).mean()

def myaverage_naive(im, filter_size = 5):
    ''' im : 入力画像, filter_size : 平均化フィルタのサイズ(奇数) '''
    iheight, iwidth = im.shape[:2]
    imout = np.zeros((iheight, iwidth))

    # ↓ここにコードを追加
    pad_size = filter_size // 2

    # 画像をパディングする
    padded_im = np.zeros((iheight + 2 * pad_size, iwidth + 2 * pad_size))
    padded_im[pad_size:pad_size + iheight, pad_size:pad_size + iwidth] = im

    for i in range(iheight):
        for j in range(iwidth):
            # フィルタウィンドウ内の合計値を計算
            sum_value = 0.0
            for k in range(filter_size):
                for l in range(filter_size):
                    sum_value += padded_im[i + k, j + l]
```

```
32              # 平均値を計算
33              imout[i, j] = sum_value / (filter_size * filter_size)
34      # ↑ここまで
35
36      return imout
37
38  def myaverage_numpy(im, filter_size = 5):
39      ''' im : 入力画像, filter_size : 平均化フィルタのサイズ(奇数) '''
40      iheight, iwidth = im.shape[:2]
41      imout = np.zeros((iheight, iwidth))
42
43      # ↓ここにコードを追加
44      pad_size = filter_size // 2
45
46      # 画像をパディングする
47      padded_im = np.zeros((iheight + 2 * pad_size, iwidth + 2 * pad_size))
48      padded_im[pad_size:pad_size + iheight, pad_size:pad_size + iwidth] = im
49
50      for i in range(iheight):
51          for j in range(iwidth):
52              # フィルタウィンドウ内の平均値を計算
53              window = padded_im[i:i + filter_size, j:j + filter_size]
54              imout[i, j] = np.mean(window)
55      # ↑ここまで
56
57      return imout
58
59  def myaverage_integral(im, filter_size=5):
60      ''' im : 入力画像, filter_size : 平均化フィルタのサイズ(奇数) '''
61      def integral_image(im):
62          ''' 積分画像の作成 '''
63          iheight, iwidth = im.shape[:2]
64          s = np.zeros_like(im)
65
66          for i in range(iheight):
67              for j in range(iwidth):
68                  s[i, j] = im[i, j]
69                  if i > 0:
70                      s[i, j] += s[i - 1, j]
71                  if j > 0:
72                      s[i, j] += s[i, j - 1]
73                  if i > 0 and j > 0:
74                      s[i, j] -= s[i - 1, j - 1]
75
76          return s
77
78      iheight, iwidth = im.shape[:2]
79      imout = np.zeros((iheight, iwidth))
80
81      # パディングのサイズを計算
82      pad_size = filter_size // 2
83
84      # 画像をパディングする
```

```
85      padded_im = np.pad(im, pad_width=pad_size, mode='constant',
            constant_values=0)
86
87      # 積分画像の計算
88      integral_im = integral_image(padded_im)
89
90      for i in range(iheight):
91          for j in range(iwidth):
92              r1 = i
93              c1 = j
94              r2 = i + filter_size - 1
95              c2 = j + filter_size - 1
96
97              total = integral_im[r2, c2]
98              if r1 > 0:
99                  total -= integral_im[r1 - 1, c2]
100             if c1 > 0:
101                 total -= integral_im[r2, c1 - 1]
102             if r1 > 0 and c1 > 0:
103                 total += integral_im[r1 - 1, c1 - 1]
104
105             imout[i, j] = total / (filter_size * filter_size)
106
107     return imout
108
109 def myaverage_separable(im, filter_size=5):
110     '''im : 入力画像, filter_size : 平均化フィルタのサイズ(奇数) '''
111     iheight, iwidth = im.shape[:2]
112     imout = np.zeros((iheight, iwidth))
113
114     pad_size = filter_size // 2
115     kernel = np.ones(filter_size) / filter_size
116
117     # 横方向のフィルタリング
118     padded_im = np.pad(im, ((0, 0), (pad_size, pad_size)), mode='constant',
            constant_values=0)
119     im_temp = np.zeros_like(padded_im)
120
121     for i in range(iheight):
122         for j in range(iwidth):
123             im_temp[i, j] = np.sum(padded_im[i, j:j + filter_size] * kernel)
124
125     # 縦方向のフィルタリング
126     padded_im_temp = np.pad(im_temp[:, pad_size:-pad_size], ((pad_size,
            pad_size), (0, 0)), mode='constant', constant_values=0)
127
128     for i in range(iheight):
129         for j in range(iwidth):
130             imout[i, j] = np.sum(padded_im_temp[i:i + filter_size, j] *
                kernel)
131
132     return imout
133
```

```
134
135
136 #  拡大率変化用
137 def resize_and_time(image, scales, filter_size):
138     times_naive = []
139     times_numpy = []
140     times_integral = []
141     times_separable = []
142
143     for scale in scales:
144         resized_image = resize(image, (int(image.shape[0] * scale), int(
                image.shape[1] * scale)))
145
146         start = time.time()
147         myaverage_naive(resized_image, filter_size)
148         elapsed_time = time.time() - start
149         times_naive.append(elapsed_time)
150
151         start = time.time()
152         myaverage_numpy(resized_image, filter_size)
153         elapsed_time = time.time() - start
154         times_numpy.append(elapsed_time)
155
156         start = time.time()
157         myaverage_integral(resized_image, filter_size)
158         elapsed_time = time.time() - start
159         times_integral.append(elapsed_time)
160
161         start = time.time()
162         myaverage_separable(resized_image, filter_size)
163         elapsed_time = time.time() - start
164         times_separable.append(elapsed_time)
165
166     return times_naive, times_numpy, times_integral, times_separable
167
168 '''
169 #フィルタサイズ変化用
170 def resize_and_time(image, scales, filter_size):
171     times_naive = []
172     times_numpy = []
173     times_integral = []
174     times_separable = []
175
176     for filter_size in filter_sizes:
177         resized_image = resize(image, (int(image.shape[0] * scales), int(
                image.shape[1] * scales)))
178
179         start = time.time()
180         myaverage_naive(resized_image, filter_size)
181         elapsed_time = time.time() - start
182         times_naive.append(elapsed_time)
183
184         start = time.time()
```

```
        myaverage_numpy(resized_image, filter_size)
        elapsed_time = time.time() - start
        times_numpy.append(elapsed_time)

        start = time.time()
        myaverage_integral(resized_image, filter_size)
        elapsed_time = time.time() - start
        times_integral.append(elapsed_time)

        start = time.time()
        myaverage_separable(resized_image, filter_size)
        elapsed_time = time.time() - start
        times_separable.append(elapsed_time)

    return times_naive, times_numpy, times_integral, times_separable
'''

im = 255 * rgb2gray(imread("data/himeji_noise.png"))
filter_sizes = 5
scales = [0.5, 1, 1.5, 2, 2.5, 3]

# 処理時間の計測
times_naive, times_numpy, times_integral, times_separable = resize_and_time(
    im, scales, filter_sizes)

#  拡大率変化用
kernel = np.ones((filter_sizes, filter_sizes)) / (filter_sizes ** 2)
im2a = myaverage_naive(im, filter_sizes)
im2b = myaverage_numpy(im, filter_sizes)
im2c = myaverage_integral(im, filter_sizes)
im2d = myaverage_separable(im, filter_sizes) # オプション
im2_gt = correlate(im, kernel, mode="constant")

print("mse_naive=", mse(im2_gt, im2a)) # 1未満であればOK
print("mse_numpy=", mse(im2_gt, im2b)) # 1未満であればOK
print("mse_integral=", mse(im2_gt, im2c)) # 1未満であればOK
print("mse_separable=", mse(im2_gt, im2d)) # オプション

'''
# フィルタサイズ変化用
for size in filter_sizes:
    kernel = np.ones((size, size)) / (size ** 2)
    im2a = myaverage_naive(im, size)
    im2b = myaverage_numpy(im, size)
    im2c = myaverage_integral(im, size)
    im2_gt = correlate(im, kernel, mode="constant")

    print(f"mse_naive (filter size {size})=", mse(im2_gt, im2a)) # 1未満であればOK
    print(f"mse_numpy (filter size {size})=", mse(im2_gt, im2b)) # 1未満であればOK
    print(f"mse_integral (filter size {size})=", mse(im2_gt, im2c)) # 1未満であればOK
```

```
234
235 '''
236 # 結果のプロット
237 plt.figure(figsize=(10, 6))
238
239 #  拡大率変化用
240 plt.plot(scales, times_naive, label='(a) Naive')
241 plt.plot(scales, times_numpy, label='(b) Numpy')
242 plt.plot(scales, times_integral, label='(c) Integral')
243 plt.plot(scales, times_separable, label='(d) Separable')
244
245 '''
246 #フィルタサイズ変化用
247 plt.plot(filter_sizes, times_naive, label='(a) Naive')
248 plt.plot(filter_sizes, times_numpy, label='(b) Numpy')
249 plt.plot(filter_sizes, times_integral, label='(c) Integral')
250 #plt.plot(filter_sizes, times_separable, label='(d) Separable')
251 '''
252
253 plt.xlabel('Scale Factor (m)')
254 plt.ylabel('Time (seconds)')
255 plt.title('Processing Time for Different Averaging Methods')
256 plt.legend()
257 plt.show()
```