画像工学 レポート

— 画像の 2 値化 —

IE5 (9) 片岡 駿之介

2016 年 11 月 21 日

1　課題

画像処理の中に，2 値化と呼ばれるものがある．2 値化とは，濃淡のある画像を白と黒の 2 階調に変換する処理のことである．ある閾値を決め，各画素ごとの輝度値が閾値を上回っていれば白，下回っていれば黒に置き換える．この処理を全画素に適用することにより，画像全体で 2 値化を行うことが出来る．2 値化は主に，画像から文字や特定色の領域を抽出すると行った目的で用いられる．
本課題では，閾値の計算にモード法と判別分析法（大津の 2 値化）を用いた 2 値化処理を実際に実装し，結果を確認する．

2　処理の流れ

今回の演習では，PGM ファイルを以下の手順で処理していくこととした．

1. 入力されたコマンドが正しいかチェック（正しくない場合は Usage を表示して終了）
2. コマンドで指定された PGM ファイルを読み込む
3. 読み込んだ PGM ファイルのヘッダ解析
4. ヘッダが正しければ画素値を 2 次元配列に取り込む
5. コマンドで指定された方法で閾値の計算を行う
6. 計算した閾値を元に 2 値化処理を行う
7. コマンドで指定されたファイルへ画像を出力する

この処理によって得られた PGM ファイルを ImageMagick を用いて表示し，2 値化の結果を比較検討する．

3 ソースリスト

ソースコード 1　binarization.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  FILE *fp;
7
8  int width;
9  int height;
10 int max_value;
11 int status;
12 char buffer[128];
13
14 void image_open(char *file_name) {
15   if ((fp = fopen(file_name, "r")) == NULL) {
16               printf("file␣open␣error!!\n");
17               exit(-1); /* (3) エラーの場合は通常､異常終了する */
18         }
19 }
20
21 int main(int argc,char *argv[]) {
22
23   /* 入力コマンドのチェック */
24   if(argc != 4) {
25     printf("Usage␣:␣./binarization␣<binarization␣type>␣<input␣pgm␣filename>␣<output␣pgm␣filename>\n");
26     exit(0);
27   } else if (argc == 4) {
28     image_open(argv[2]);
29     printf("mode␣:␣%s\n", argv[1]);
30   }
31
32   /* ヘッダ取得部 */
33   int ch;
34
35   while (status < 3) {
36     ch = getc(fp);
37
38     if(ch == '#') { // コメントのスキップ
39       while (( ch = getc(fp)) != '\n')
40         break;
41     }
42
43     if(ch == 'P') { // マジックナンバーのチェック
44       if(getc(fp) != '5') {
45         printf("Magic␣Number␣is␣wrong.\n");
46         break;
47       } else {
48         // printf("Magic Number is P5\n");
49       }
50     }
51
52     if(isdigit((unsigned char)ch)) { // 数値取得部
```

```
53      buffer[0] = ch;
54      int i=1;
55      while(1) {
56          char c = getc(fp);
57          if(isdigit((unsigned char)c)) {
58            buffer[i]=c;
59            i++;
60          } else
61            break;
62      }
63      buffer[i] = '\0';
64
65      switch (status) {
66        case 0: // width 取得部
67          width = atoi(buffer);
68          printf("width=%d\n", width);
69          break;
70        case 1: // height 取得部
71          height = atoi(buffer);
72          printf("height=%d\n", height);
73          break;
74        case 2: // max_value 取得部
75          max_value = atoi(buffer);
76          printf("max_value=%d\n", max_value);
77          break;
78      }
79      status++; // 次のステータスへ
80    }
81  }
82
83  /* 画素値取得部 */
84  int image[width][height];
85
86  for(int i = 0; i < height; i++) {
87    for(int j = 0; j < width; j++) {
88      image[j][i] = (int)getc(fp);
89    }
90  }
91
92  /* ヒストグラム生成部 */
93  int histogram[max_value];
94  for(int i=0; i <= max_value; i++)
95    histogram[i] = 0; //ヒストグラム用配列の初期化
96
97  for(int i=0; i < width; i++) {
98    for(int j=0; j < height; j++) {
99      histogram[image[j][i]]++;
100   }
101 }
102
103 /* 閾値計算部 */
104
105 int threshold = 0;
106
107 int output_image[width][height];
108
109 for(int i=0; i < height; i++) {
```

```
110     for(int j=0; j < width; j++) {
111       output_image[j][i] = image[j][i];
112     }
113   }
114
115   if(strcmp(argv[1], "mode") == 0) { //モード法
116
117     int dHist[max_value-1]; //微分ヒストグラム用の配列
118     for(int i=0; i <= (max_value-1); i++) {
119       dHist[i] = histogram[i] - histogram[i+1]; //微分ヒストグラム用配列の計算
120       // printf("i=%d, dHist[i]=%d \n", i, dHist[i]);
121     }
122
123     int range = 40; //最頻値検出用の微分ヒストグラムの探索幅
124     int valley = 0; //谷の画素値
125     int valley_first = 1;
126
127     for(int i = range; i < (max_value - range); i++) {
128       int minus_range = 0;
129       int plus_range = 0;
130
131       for(int j = 0; j <= range; j++) { //両側レンジの微分ヒストグラムの合計値の計算
132         minus_range = minus_range + dHist[i-j];
133         plus_range = plus_range + dHist[i+j];
134       }
135
136       printf("i␣=␣%d,␣minus_range␣=␣%d,␣plus_range␣=␣%d\n", i, minus_range, plus_range);
137
138       if(minus_range >= 0 && plus_range <= 0) { //マイナス側レンジが右下がりの傾き かつ プラス側
    レンジが右上がりの傾きのとき
139         if(valley_first == 1 || histogram[valley] > histogram[i]) //現在のヒストグラムの値が
    valley より低ければ valley を更新
140           valley = i;
141           if(valley_first == 1)
142             valley_first = 0;
143       }
144     }
145
146     printf("valley=%d\n", valley);
147
148     threshold = valley;
149
150     for(int k=0; k <= max_value; k++) {
151       // printf("%4d %5d\n", k, histogram[k]);
152     }
153
154   } else if(strcmp(argv[1], "otsu") == 0) { //判別分析法（大津の 2 値化）
155
156     int min = -1; //最小値
157     int max = -1; //最大値
158     int sum = 0;
159
160     for(int i = 0; i <= max_value; i++) { //平均値、最小値、最大値を求める
161       sum += (histogram[i] * i);
162
163       if(histogram[i] != 0) { //その輝度値に画素が存在している場合
164         if(min == -1)
165           min = i;
```

```
166
167         max = i;
168       }
169     }
170
171     if(min == -1 || max == -1)
172       exit(-1);
173
174     float ave = sum / (width * height);
175     printf("Average␣:␣%f,␣Max␣:␣%d,␣Min␣:␣%d\n", ave, max, min);
176
177     /* 分離度 S を求め、閾値を決める */
178     int s = 0; //分離度 S
179
180     for(int t = min; t < max; t++) {
181       int n1 = 0; //t より左側の画素数の合計
182       int v1 = 0; //t より左側の画素値の合計
183       int n2 = 0; //t より右側の画素数の合計
184       int v2 = 0; //t より右側の画素値の合計
185
186       for(int j = min; j <= max; j++) {
187         if(j <= t) {
188           n1 += histogram[j];
189           v1 += (histogram[j] * j);
190           // printf("j : %d, n1 : %d, v1 : %d\n", j, n1, v1);
191         } else {
192           n2 += histogram[j];
193           v2 += (histogram[j] * j);
194           // printf("j : %d, n2 : %d, v2 : %d\n", j, n2, v2);
195         }
196       }
197
198       float ave1 = (float)(v1 / n1); //t より左側の画素値の平均
199       float ave2 = (float)(v2 / n2); //t より右側の画素値の平均
200
201       float sum1 = 0; //t より左側の平均との差の二乗を足し込む変数
202       float sum2 = 0; //t より右側の平均との差の二乗を足し込む変数
203
204       for(int j = min; j <= max; j++) {
205         if(j <= t)
206           sum1 += ((j - ave1) * (j - ave1));
207         else
208           sum2 += ((j - ave2) * (j - ave2));
209       }
210
211       float sigma_1 = sum1 / n1; //t より左側の分散
212       float sigma_2 = sum2 / n2; //t より右側の分散
213
214       float sigma_w = (n1 * sigma_1 + n2 * sigma_2) / (n1 + n2); //クラス内分散
215       float sigma_b = (n1 * ((ave1 - ave) * (ave1 - ave)) + n2 * ((ave2 - ave) * (ave2 - ave))) / (n1 + n2)
    クラス間分散
216
217       int s_local = sigma_b / sigma_w; //分離度の計算
218
219       if(s_local > s) {//分離度が大きければ閾値を更新
220         s = s_local;
221         threshold = t;
```

```
222       }
223     }
224   }
225
226   printf("threshold␣:␣%d\n", threshold);
227   /* 2値化処理 */
228
229   for(int i = 0; i < height; i++) {
230     for(int j = 0; j < width; j++) {
231       if(image[j][i] > threshold)
232         output_image[j][i] = max_value;
233       else
234         output_image[j][i] = 0;
235     }
236   }
237
238   /* 画像出力部 */
239
240   FILE *output = fopen(argv[3], "wb");
241   char header[30];
242   sprintf(header, "P5␣%d␣%d␣%d\n", width, height, max_value);
243   fputs(header, output);
244   for(int i = 0; i < height; i++) {
245     for(int j = 0; j < width; j++) {
246       fputc((char)output_image[j][i], output);
247     }
248   }
249   printf("image␣was␣output.\n");
250   fclose(output);
251 }
```

4　実験

検証用画像として，「cameraman.pgm」「source.pgm」「town.pgm」を用意した．
それぞれの検証用画像を以下に示す．

図 1　cameraman.pgm

図 2　source.pgm

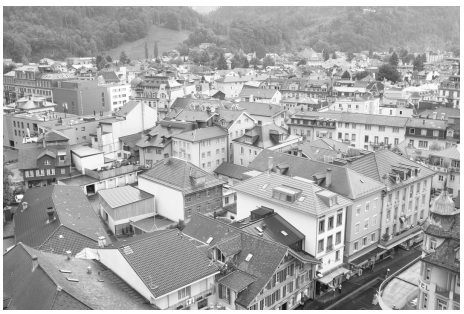

図 3　town.pgm

また，これらの検証用画像のヒストグラムを以下に示す．

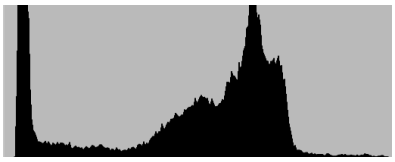

図 4　cameraman.pgm

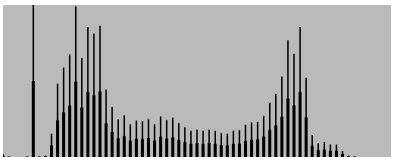

図 5　source.pgm

図 6　town.pgm

4.1　モード法

モード法により閾値を決定する際には，双峰型ヒストグラムの谷を検出することが重要である．今回，谷の検出には以下の手法を用いた．

1. 画像のヒストグラムを微分していき，傾きを dHist 配列に格納する
2. 1 の処理を 0〜max_value で繰り返し行う
3. 対象とする輝度値から左右 range 変数分の合計値を求め，minus_range 変数と plus_range 変数にそれぞれ代入する
4. もしマイナス側レンジが 0 以上かつプラス側レンジが 0 以下だった場合（すなわち，極小値を取る場合），谷の輝度値を更新する
5. 3 と 4 の処理を繰り返し行い，最後に谷の輝度値を閾値として設定する

この方法を用いて，検証用画像に対して 2 値化処理を行った結果を以下に示す．

図 7　cameraman.pgm

図 8　source.pgm

図 9　town.pgm

4.2　判別分析法（大津の 2 値化）

判別分析法は，クラス間分散とクラス内分散の比で求められる分離度を求め，この分離度が最大となるときの輝度値を閾値とするものである．今回，判別分析法を以下の方法に寄って実装した．

1. 画像全体の平均値，最小値，最大値を求める
2. 分離度を求める
3. もし分離度が現在の分離度よりも大きければ，分離度を更新し，そのときの輝度値を記録
4. 2 と 3 の処理を 0〜max_value で繰り返し行う
5. 分離度が最大のときの輝度値を閾値とする

この方法を用いて，検証用画像に対して 2 値化処理を行った結果を以下に示す．

図 10　cameraman.pgm

図 11　source.pgm

図 12　town.pgm

5　考察

5.1　モード法

連続的な双峰型ヒストグラムとなっている「cameraman.pgm」では，理想的な 2 値化が行われているように感じられる．
「source.pgm」は双峰型ヒストグラムであるが，輝度値が離散的なため，今回の実装手法では谷の探索が正確

に行えず，理想的な 2 値化とは言い難い結果となった．
「town.pgm」はそもそも双峰型ヒストグラムではないため，モード法では理想的な 2 値化が行われていない．

「cameraman.pgm」のような，なだらかな双峰型ヒストグラムでは良好な結果が得られるが，他の 2 枚のような状況では良好な結果が得られず，使い所がかなり限られてしまうことがわかった．

5.2 判別分析法（大津の 2 値化）

大津の 2 値化では，3 枚ともかなり良好な結果が得られた．こちらはモード法に比べて，どのようなヒストグラムでも安定的に 2 値化を行うことが出来るため，非情に使い易いアルゴリズムであることがわかった．

6 感想

今回は 2 値化処理の実装を行ったが，モード法では自分でアルゴリズムを考案し，今まで自分の中ではあやふやだった画像の微分処理の本質を理解することができたので良かった．ただ，このモード法の実装方法は必ずしも良いモノとは言えないので，さらに突き詰める必要が有るようにも感じる．
また，大津の 2 値化は工学セミナーなどでアルゴリズム自体は聞いたことがあったが，実際に自分で実装したことはなかったので良い経験になった．