# 情報メディアプロジェクトⅠ：最終課題レポート 1

**澤 祐里**（21–1–037–0801）

2021 年 7 月 17 日

# 目次

1 **目的** 1

2 **課題（類似画像検索システム）の概要** 2

2.1 ソースコードとアルゴリズムの説明 . . . . . 2

2.2 何を入力して、どのように出力されるのか . . . . . 7

3 **画像特徴量の計算結果** 8

4 **類似画像検索の実験結果** 9

5 **考察** 10

6 **むすび** 11

# 1　目的

以下のような画像の特徴量から、類似の画像を検索できるようなプログラムを作成する。

- 濃度
- 縦横比
- 水平方向ラン数平均
- 垂直方向ラン数平均
- 水平方向ラン数標準偏差
- 垂直方向ラン数標準偏差
- 重心の x 座標
- 重心の y 座標

検索結果の 1 位から 10 位までをテキストファイルに出力する。また、それらの画像の検索に用いた特徴量、検索キーの画像番号を表示させ、その下に検索順位の上位順に結果を出す。検索結果は、検索順位、図形番号、（正規化前の）特徴量の値、検索キーとの距離として、10 位までの画像すべてに対して出力する。さらに、10 個の画像をディレクトリにまとめて保存できるようにする。

# 2　課題（類似画像検索システム）の概要

## 2.1　ソースコードとアルゴリズムの説明

1. 検索に使用する特徴量、キー画像の番号をキーボードから入力する。
   → まず、以下の図 1 のようにして、変数「検索に使用する特徴の番号」と「キー画像番号」にキーボードから入力された番号を入れる。

```
select = Featureex.inputkeyboard(0);        //キーボードから選択された特徴を変数に格納
selectpicture = Featureex.inputkeyboard(1); //キーボードから入力されたキー画像番号を格納
selectnumber = Featureex.select(select, fenum); //選択された特徴を単一値に変換して配列に格納
fenum = Featureex.fenumselect(selectnumber);  //選択された特徴の数を変数に格納
```

図 1　キーボードから入力された番号を変数に格納

   → キーボードから入力された検索に使用する特徴の番号、キー画像の番号を取得するメソッドは、図 2 のようになっている。

```
public static int inputkeyboard(int i){
    if(i == 0){
        System.out.println(
        "検索に利用する特徴を「68341」のように入力してください"+line+
        "1:濃度値"+line+
        "2:縦横比"+line+
        "3:水平方向ラン数平均"+line+
        "4:垂直方向ラン数平均"+line+
        "5:水平方向ラン数標準偏差"+line+
        "6:垂直方向ラン数標準偏差"+line+
        "7:重心のx座標"+line+
        "8:重心のy座標"+line
        );
    }
    if(i == 1){
        System.out.println(
        "検索に利用するキー画像番号を入力してください"+line
        );
    }
    Scanner scan = new Scanner(System.in);
    int num = scan.nextInt();


    if(i == 0){
        System.out.println("入力された文字は「" + num + "」です");
    }
    if(i == 1){
        System.out.println("選択された画像は「" + num + ".bmp」です");
        scan.close();
    }
    return num;
}
```

図 2　キーボードから入力された番号を取得するメソッド [2]

2. 100 枚の画像を読み込み、特徴量を抽出する。
   → まず、以下の図 3 のようにして画像を読み込み、図 4 のようなメソッドを呼び出して、画像の特徴量を 2 次元配列に格納する。

```
for(int i=0; i<num; i++){ //全画像データから特徴抽出
    String fileName = "last1data/" + Integer.toString(i+1) + ".bmp";
    GImage img= new GImage(fileName);
    for(int j=0;j<fenum;j++){
        Feature[j][i] =  Featureex.get_Feature(j,img,selectnumber);
    }
}
```

図 3　画像を読み込み、特徴量を抽出するプログラム

   → 図 4 のメソッドでは、検索に使用する特徴量を判別して、対応する特徴量抽出メソッドを呼び出している。

```
for(int i=0; i<num; i++){ //全画像データから特徴抽出
    String fileName = "last1data/" + Integer.toString(i+1) + ".bmp";
    GImage img= new GImage(fileName);
    for(int j=0;j<fenum;j++){
        Feature[j][i] =  Featureex.get_Feature(j,img,selectnumber);
    }
}
```

図 4　選択された特徴の特徴量抽出メソッドを呼び出すメソッド

→ 下記に各特徴量を抽出するメソッドを示す。

- 画像の濃度は以下の図 5 のようなメソッドで抽出する。
- 画像の縦横比は以下の図 6 のようなメソッドで抽出する。
- 画像の水平方向ラン数平均は以下の図 7 のようなメソッドで抽出する。
- 画像の垂直方向ラン数平均は以下の図 8 のようなメソッドで抽出する。
- 画像の水平方向ラン数標準偏差は以下の図 9 のようなメソッドで抽出する。
- 画像の垂直方向ラン数標準偏差は以下の図 10 のようなメソッドで抽出する。
- 画像の重心の x 座標は以下の図 11 のようなメソッドで抽出する。
- 画像の重心の y 座標は以下の図 12 のようなメソッドで抽出する。

```java
public static double get_F1(GImage img1){
    int width1, height1;
    width1 = img1.getWidth();
    height1 = img1.getHeight();
    double F1 = 0;
    int sum = 0;
    double area = get_area(img1);

    for(int y=0;y<height1;y++){
        for(int x=0;x<width1;x++){
            if(img1.pixel[y][x] == GImage.MIN_GRAY){
                sum++;
            }
        }
    }
    F1 = (double)sum / area;
    return F1;
}
```

図 5　濃度抽出メソッド

```java
public static double get_F2(GImage img1){
    int width1, height1;
    width1 = img1.getWidth();
    height1 = img1.getHeight();
    double F2 = 0;
    boolean flagY = true;
    int Ymin=0;
    int Ymax=0;
    for(int y=0;y<height1;y++){
        for(int x=0;x<width1;x++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Ymax=y;
                if(flagY==true){
                    Ymin=y;
                    flagY=false;
                }
            }
        }
    }

    boolean flagX = true;
    int Xmin=0;
    int Xmax=0;
    for(int x=0;x<width1;x++){
        for(int y=0;y<height1;y++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Xmax=x;
                if(flagX==true){
                    Xmin=x;
                    flagX=false;
                }
            }
        }
    }
    F2 = ((double)(Ymax-Ymin+1))/((double)(Xmax-Xmin+1));
    return F2;
}
```

図 6　縦横比抽出メソッド

```java
public static double get_F3(GImage img1){
    int Totalnumber = 0;
    double F3 = 0;
    int MinX = get_area(img1, "MinX");
    int MaxX = get_area(img1, "MaxX");
    int MinY = get_area(img1, "MinY");
    int MaxY = get_area(img1, "MaxY");
    double Ynumber = MaxY - MinY + 1;

    for(int y=MinY;y<=MaxY;y++) {
        for(int x=MinX;x<=MaxX;x++) {
            if(img1.pixel[y][x] == GImage.MIN_GRAY && img1.pixel[y][x-1] == GImage.MAX_GRAY)
            {
                Totalnumber++;
            }
        }
    }

    F3 = (double)Totalnumber / Ynumber;
    return F3;
}
```

図 7　水平方向ラン数平均抽出メソッド

```java
public static double get_F4(GImage img1){
    int Totalnumber = 0;
    double F4 = 0;
    int MinX = get_area(img1, "MinX");
    int MaxX = get_area(img1, "MaxX");
    int MinY = get_area(img1, "MinY");
    int MaxY = get_area(img1, "MaxY");
    double Xnumber = MaxX - MinX + 1;

    for(int x=MinX;x<=MaxX;x++) {
        for(int y=MinY;y<=MaxY;y++) {
            if(img1.pixel[y][x] == GImage.MIN_GRAY && img1.pixel[y-1][x] == GImage.MAX_GRAY)
            {
                Totalnumber++;
            }
        }
    }
    F4 = (double)Totalnumber / Xnumber;
    return F4;
}
```

図 8　垂直方向ラン数平均抽出メソッド

```java
public static double get_F5(GImage img1){
    double F5 = 0;
    int MinX = get_area(img1, "MinX");
    int MaxX = get_area(img1, "MaxX");
    int MinY = get_area(img1, "MinY");
    int MaxY = get_area(img1, "MaxY");
    double Ynumber = MaxY - MinY + 1;
    double[] data = new double[(int)Ynumber];

    for(int y=MinY;y<=MaxY;y++) {
        for(int x=MinX;x<=MaxX;x++) {
            if(img1.pixel[y][x] == GImage.MIN_GRAY && img1.pixel[y][x-1] == GImage.MAX_GRAY)
            {
                data[y - MinY]++;
            }
        }
    }
    F5 = get_dev(data);
    return F5;
}
```

図 9　水平方向ラン数標準偏差抽出メソッド

```java
public static double get_F6(GImage img1){
    double F6 = 0;
    int MinX = get_area(img1, "MinX");
    int MaxX = get_area(img1, "MaxX");
    int MinY = get_area(img1, "MinY");
    int MaxY = get_area(img1, "MaxY");
    double Xnumber = MaxX - MinX + 1;
    double[] data = new double[(int)Xnumber];

    for(int x=MinX;x<=MaxX;x++) {
        for(int y=MinY;y<=MaxY;y++) {
            if(img1.pixel[y][x] == GImage.MIN_GRAY && img1.pixel[y-1][x] == GImage.MAX_GRAY)
            {
                data[x - MinX]++;
            }
        }
    }
    F6 = get_dev(data);
    return F6;
}
```

図 10　垂直方向ラン数標準偏差抽出メソッド

```java
public static double get_F7(GImage img1){
    int Totalnumber = 0;
    int xtotal = 0;
    double F7 = 0;
    int MinX = (int)get_area(img1, "MinX");
    int MaxX = (int)get_area(img1, "MaxX");
    int MinY = (int)get_area(img1, "MinY");
    int MaxY = (int)get_area(img1, "MaxY");

    for(int y=MinY;y<=MaxY;y++){
        for(int x=MinX;x<=MaxX;x++){
            if(img1.pixel[y][x] == GImage.MIN_GRAY){
                xtotal += x-MinX;
                Totalnumber++;
            }
        }
    }

    F7 = (double)xtotal / (double)Totalnumber;
    return F7;
}
```

図 11　重心の x 座標抽出メソッド

```java
public static double get_F8(GImage img1){
    int Totalnumber = 0;
    int ytotal = 0;
    double F8 = 0;
    int MinX = (int)get_area(img1, "MinX");
    int MaxX = (int)get_area(img1, "MaxX");
    int MinY = (int)get_area(img1, "MinY");
    int MaxY = (int)get_area(img1, "MaxY");

    for(int x=MinX;x<=MaxX;x++){
        for(int y=MinY;y<=MaxY;y++){
            if(img1.pixel[y][x] == GImage.MIN_GRAY){
                ytotal += y-MinY;
                Totalnumber++;
            }
        }
    }

    F8 = (double)ytotal / (double)Totalnumber;
    return F8;
}
```

図 12　重心の y 座標抽出メソッド

3. 抽出した画像の特徴量を正規化する。
   → まず、以下の図 13 のようにして、100 枚の画像の全特徴を正規化できるまで、図 14 のような正規化メソッドを呼び出す。正規化した特徴は 2 次元配列に格納している。

```
for(int i=0; i<num; i++){ //全画像データの特徴を正規化
    for(int j=0;j<fenum;j++){
        NomalizedFeature[j][i] = Featureex.get_nor(Feature[j][i], Featureex.get_scalr(j, num, Feature));
    }
}
```

図 13　全画像の全特徴を正規化するプログラム

```
public static double get_nor(double target,double data[]){
    double normalized;
    normalized = (target - get_ave(data)) / get_dev(data);
    return normalized;
}
```

図 14　画像の特徴量を正規化するメソッド

4. キー画像を読み込み画像を抽出する。
   → 以下の図 15 のようにして、キー画像を読み込み、全画像の時と同じように特徴量を抽出して 2 次元配列に格納する。同じように、特徴量を正規化したものを 2 次元配列に格納する。

```
String inputfileName = "last1data/"+Integer.toString(selectpicture)+".bmp"; //以下はキー画像側の処理
GImage inputimg= new GImage(inputfileName);
for(int j=0;j<fenum;j++){    //キー画像の特徴を抽出
    inputFeature[j] =  Featureex.get_Feature(j,inputimg,selectnumber);
}
for(int j=0;j<fenum;j++){    //キー画像の特徴を正規化
    NomalizedinputFeature[j] = Featureex.get_nor(inputFeature[j], Featureex.get_scalr(j, num, Feature));
}
```

図 15　キー画像を読み込み、正規化するプログラム

5. キー画像と全画像の距離を取得する。
   → まず、以下の図 16 のようにして、キー画像と全画像の距離を取得する、図 17 のようなメソッドを呼び出す。求めた距離は配列に格納している。

```
for(int i=0; i<num; i++){ //全画像とキー画像の距離を取得
    distance[i] = Featureex.get_distance(fenum, NomalizedinputFeature, Featureex.get_fescalr(i, fenum, NomalizedFeature));
}
```

図 16　キー画像と全画像の距離を取得するプログラム

```
public static double get_distance(int fenum,double[] inputdata,double[] comparedata){
    double distance = 0;
    double total = 0;
    for(int i=0;i<fenum;i++){
        total += (inputdata[i]-comparedata[i]) * (inputdata[i]-comparedata[i]);
    }
    distance = Math.sqrt(total);
    return distance;
}
```

図 17　キー画像と全画像の距離を取得するメソッド

6. 全画像とキー画像の距離が近い順にソートする。
   → まず、以下の図 18 のようにして、キー画像と全画像の距離が近い順にソートする図 19 のようなメソッドを呼び出す。また、ソートした結果の上位 10 個の画像番号を変数に格納する。

```
maxes = Featureex.get_maxes(resultnum, distance); //距離が小さい画像をresultnum個取得
```

図 18　キー画像と全画像の距離が近い順にソートするプログラム

```
public static int[] get_maxes(int number,double data[]){
    double[] posedata = new double[data.length];
    int[] maxes = new int[number];
    for(int i=0;i<data.length;i++){
        posedata[i] = data[i];
    }

    Arrays.sort(posedata);

    for(int i=0;i<maxes.length;i++){
        for(int j=0;j<posedata.length;j++){
            if(posedata[i + 1] == data[j]){
                maxes[i] = j + 1;
            }
        }
    }
    return maxes;
}
```

図 19　キー画像と全画像の距離が近い順にソートするメソッド

7. 検索結果の画像を全てディレクトリに保存する。テキストファイルに検索結果の 1 位から 10 位までを出力する。また、それらの画像の検索に用いた特徴量、検索キーの画像番号を表示させ、その下に検索順位の上位順に結果を出す。検索結果は、検索順位、図形番号、（正規化前の）特徴量の値、検索キーとの距離として、10 位までの画像すべてに対して出力する。
   → まず、以下の図 20 のようにして、検索結果の画像を全てディレクトリに保存する図 21 のようなメソッドを呼び出す。また、検索結果の全画像の全特徴を抽出する。その後、テキストファイルにテキストファイルに検索結果の 1 位から 10 位、それらの画像の検索に用いた特徴量、検索キーの画像番号を表示させ、その下に検索順位の上位順に結果、検索順位、図形番号、（正規化前の）特徴量の値、検索キーとの距離として、10 位までの画像すべてに対して出力する、図 22 のようなメソッドを呼び出す。

```
for(int i=0; i<maxes.length; i++){
    String fileName = "last1data/" + Integer.toString(maxes[i]) + ".bmp";
    GImage outputimg= new GImage(fileName);
    for(int j=0;j<fenum2;j++){
        outputFeature[j][i] = Featureex.get_Feature(j, outputimg, outputselect);
    }
    String outputfileName = "Lastreportoutput/outputimg/" + Integer.toString(maxes[i]);
    Featureex.imgoutput(outputimg, outputfileName);
}

for(int i=0;i<resultnum;i++){
    Featureex.filewrite(maxes,selectnumber,selectpicture,Feature,distance,fenum,outputFeature,fenum2);
}
```

図 20　出力用プログラム

```
public static void imgoutput(GImage img,String fileName02){ //画像出力
    String fileType02 = "bmp";
    img.output(fileName02,fileType02);
    fileName02 +="." + fileType02;
    System.out.println("Output file:"+fileName02);
}
```

図 21　画像をディレクトリに保存するメソッド

```
public static void filewrite(int[] data,int[] selectnumber,int selectpicture,double[][] Feature,double[] distance,int fenum,double[][] outputFeature,int fenum2){
    String path = "Lastreportoutput\\output.txt";

    try{
        File file = new File(path);
        FileWriter writefile = new FileWriter(file);
        writefile.write(
                "検索に使用した特徴量: "
        );
        for(int j=fenum-1;j>=0;j--){
            switch (selectnumber[j]) {
                case 1:
                    writefile.write("濃度");
                    break;
                case 2:
                    writefile.write("縦横比");
                    break;
                case 3:
                    writefile.write("水平方向ラン数平均");
                    break;
                case 4:
                    writefile.write("垂直方向ラン数平均");
                    break;
                case 5:
                    writefile.write("水平方向ラン数標準偏差");
                    break;
                case 6:
                    writefile.write("垂直方向ラン数標準偏差");
                    break;
                case 7:
                    writefile.write("重心のx座標");
                    break;
                case 8:
                    writefile.write("重心のy座標");
                    break;
                default:
                    break;
            }
            if(j > 0){
                writefile.write(",  ");
            }
        }
        writefile.write(
                line+"検索キーの画像番号: "+selectpicture+".bmp"+line
        );
        for(int i=0;i<data.length;i++){
            String graph = String.valueOf(data[i]);
            String distan = String.valueOf(distance[data[i]-1]);
            writefile.write(
                "検索結果第"+(i+1)+"位:  "+graph+".bmp  キー画像との距離: "+distan+",  "
            );
            for(int j=0;j<fenum2;j++){
                String feature = String.valueOf(outputFeature[j][i]);
                switch (j) {
                    case 0:
                        writefile.write("濃度: "+feature);
                        break;
                    case 1:
                        writefile.write("縦横比: "+feature);
                        break;
                    case 2:
                        writefile.write("水平方向ラン数平均: "+feature);
                        break;
                    case 3:
                        writefile.write("垂直方向ラン数平均: "+feature);
                        break;
                    case 4:
                        writefile.write("水平方向ラン数標準偏差: "+feature);
                        break;
                    case 5:
                        writefile.write("垂直方向ラン数標準偏差: "+feature);
                        break;
                    case 6:
                        writefile.write("重心のx座標: "+feature);
                        break;
                    case 7:
                        writefile.write("重心のy座標: "+feature);
                        break;
                    default:
                        break;
                }
                if(j < fenum2 -1){
                    writefile.write(",  ");
                }
                else{
                    writefile.write(line);
                }
            }
        }

        writefile.close();
    }
    catch(IOException e){
    }
}
```

図 22　テキストファイルに書き込むメソッド [1]

その他、課題で使用したメソッド

- 以下の図 23 のメソッドは、配列の平均を返す。
- 以下の図 24 のメソッドは、配列の標準偏差を返す。
- 以下の図 25 のメソッドは、画像の外接矩形の端の四辺を返す。
- 以下の図 26 のメソッドは、画像の外接矩形の面積を返す。
- 以下の図 27 のメソッドは、2 次元配列を 1 次元配列に変換する。
- 以下の図 28 のメソッドは、キーボードから入力された特徴の番号を配列化したものと選ばれた特徴の数を返す。

```
public static double get_ave(double data[]) { //全てのデータの平均値を返す
    double sum = 0;
    double ave;

    for (int i = 0; i < data.length; i++) {
        sum = sum + data[i];
    }
    ave = sum / ((double) data.length);

    return ave;
}
```

図 23　平均を返すメソッド

```
public static double get_dev(double data[]) { //全てのデータの標準偏差を返す
    double sum2 = 0;
    double sum = 0;
    double var, dev;

    for (int i = 0; i < data.length; i++) {
        sum2 = sum2 + data[i]*data[i];
        sum = sum + data[i];
    }
    var = sum2 - (sum*sum) / ((double) data.length);
    var = var / ((double) data.length);
    dev =Math.sqrt(var);

    return dev;
}
```

図 24　標準偏差を返すメソッド

```
public static int get_area(GImage img1,String where) { //外接矩形のMinX,MinY,MaxX,MaxYを返す
    int width1, height1;
    width1 = img1.getWidth();
    height1 = img1.getHeight();

    /** Decide vertical frame line */
    boolean flagY = true;
    int Ymin=0;
    int Ymax=0;
    for(int y=0;y<height1;y++){
        for(int x=0;x<width1;x++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Ymax=y;
                if(flagY==true){
                    Ymin=y;
                    flagY=false;
                }
            }
        }
    }

    boolean flagX = true;
    int Xmin=0;
    int Xmax=0;
    for(int x=0;x<width1;x++){
        for(int y=0;y<height1;y++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Xmax=x;
                if(flagX==true){
                    Xmin=x;
                    flagX=false;
                }
            }
        }
    }
    switch (where){
      case "MinY":
          return Ymin;
      case "MaxY":
          return Ymax;
      case "MinX":
          return Xmin;
      default:
          return Xmax;
    }
}
```

図 25　外接矩形の端の四辺を返すメソッド

```
public static double get_area(GImage img1) { //外接矩形の面積を返す
    int width1, height1;
    width1 = img1.getWidth();
    height1 = img1.getHeight();

    /** Decide vertical frame line */
    boolean flagY = true;
    int Ymin=0;
    int Ymax=0;
    for(int y=0;y<height1;y++){
        for(int x=0;x<width1;x++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Ymax=y;
                if(flagY==true){
                    Ymin=y;
                    flagY=false;
                }
            }
        }
    }

    boolean flagX = true;
    int Xmin=0;
    int Xmax=0;
    for(int x=0;x<width1;x++){
        for(int y=0;y<height1;y++){
            if(img1.pixel[y][x]==GImage.MIN_GRAY){
                Xmax=x;
                if(flagX==true){
                    Xmin=x;
                    flagX=false;
                }
            }
        }
    }

    double area;
    area = ((double)(Ymax-Ymin+1))*((double)(Xmax-Xmin+1));
    return area;
}
```

図 26　外接矩形の面積を返すメソッド

```
public static double[] get_fescalr(int fenum,int num,double[][] data){ //1枚の画像の特徴を配列化
    double[] newdata = new double[num];
    for(int i=0;i<num;i++){
        newdata[i] = data[i][fenum];
    }
    return newdata;
}
public static double[] get_scalr(int fenum,int num,double[][] data){
    double[] newdata = new double[num];
    for(int i=0;i<num;i++){
        newdata[i] = data[fenum][i];
    }
    return newdata;
}
```

図 27　2 次元配列を 1 次元配列に変換するメソッド

```
public static int fenumselect(int[] select) {
    int fenum = 0;
    for(int i=0;i<select.length;i++){
        if(select[i] != 0){
            fenum++;
        }
    }
    return fenum;
}
public static int[] select(int select,int fenum){
    int[] selectnumber = new int[fenum];
    int j = fenum - 1;
    int index = 1;
    for(int i = 1 ; i < selectnumber.length ; i++){
        index = index * 10;
    }
    for(int i=index;i>0;i/=10){
        selectnumber[j] = select / i;
        select = select % i;
        j--;
    }
    return selectnumber;
}
```

図 28　特徴の番号を配列化したものと選ばれた特徴の数を返すメソッド

## 2.2　何を入力して、どのように出力されるのか

1. プログラムを実行するとコンソールに図 29 のような画面が表示される。
2. 指示通りに検索に使う特徴量の番号を入力すると図 30 のような画面が表示される。
3. 次に、キー画像番号を入力すると図 31 のような画面が表示される。
4. コンソールに表示されたディレクトリを開くと図 32 のようになっている。
5. ディレクトリにあるテキストファイル「output.txt」を開くと図 33 のようになっている。内容は、画像の検索に用いた特徴量、検索キーの画像番号、その下に検索順位の上位順に結果があり、その横には検索順位、図形番号、（正規化前の）特徴量の値、検索キーとの距離がある。
6. もう一つのディレクトリ「outputimg」を開くと図 34 のようになっており、中にある画像は、すべて検索結果の画像となっている。

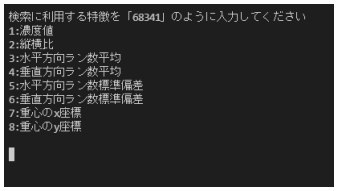


図 29　コンソールの初期画面

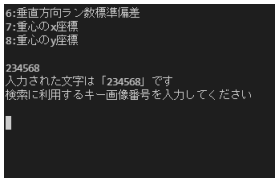


図 30　検索に使用する特徴入力後


図 31　キー画像番号入力後


図 32　出力ディレクトリの中身

図 33　output.txt の中身

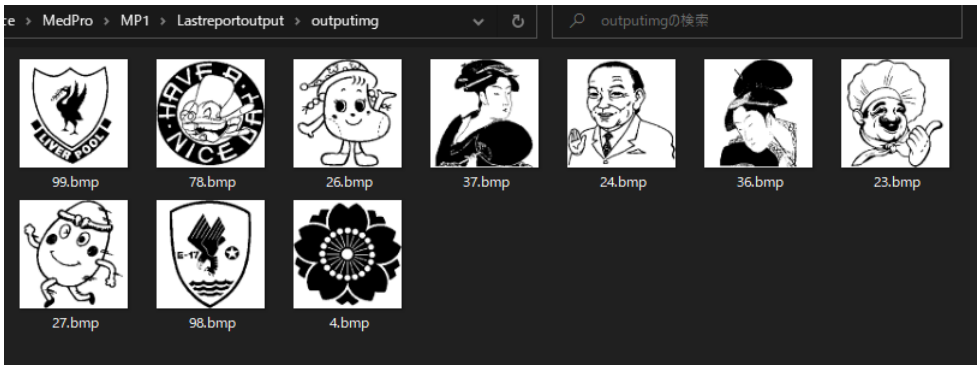


図 34　outputimg の中身

# 3　画像特徴量の計算結果

自身の学籍番号の下一桁 (1) と同じ下一桁の図形番号（ファイル名）を持つ画像すべての特徴量を表にして示す。

→ 以下の表 1 は、私の学籍番号の下一桁 (1) と同じ下一桁の図形番号の画像全ての特徴量の表である。

表 1　画像特徴量の計算結果

| 図形番号 | 濃度 (f1) | 縦横比 (f2) | 水平ラン平均 (f3) | 垂直ラン平均 (f4) | 水平ラン SD (f5) | 垂直ラン SD (f6) | 重心 x 座標 (f7) | 重心 y 座標 (f8) |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.688 | 0.972 | 2.246 | 2.196 | 1.416 | 1.283 | 123.287 | 120.192 |
| 11 | 0.358 | 0.366 | 4.634 | 1.826 | 1.577 | 0.653 | 123.753 | 47.256 |
| 21 | 0.126 | 1.149 | 4.145 | 4.574 | 2.082 | 1.923 | 96.624 | 118.42 |
| 31 | 0.306 | 1.468 | 20.814 | 29.479 | 8.146 | 7.452 | 82.307 | 139.802 |
| 41 | 0.299 | 4.792 | 1.334 | 7.396 | 0.96 | 4.448 | 26.305 | 93.717 |
| 51 | 0.182 | 0.22 | 5.357 | 1.066 | 1.767 | 0.607 | 139.961 | 28.444 |
| 61 | 0.285 | 0.468 | 7.168 | 3.952 | 4.828 | 2.362 | 121.481 | 65.084 |
| 71 | 0.55 | 1.028 | 3.753 | 2.91 | 2.989 | 1.01 | 122.182 | 121.069 |
| 81 | 0.449 | 0.996 | 5.539 | 6.162 | 3.192 | 3.44 | 126.806 | 134.005 |
| 91 | 0.297 | 0.846 | 17.181 | 10.956 | 7.823 | 3.147 | 127.472 | 111.291 |

## 4　類似画像検索の実験結果

自身の学籍番号の下二桁 (01) と同じ下二桁の図形番号（ファイル名）を持つ画像を検索キー（入力画像）としたときの検索結果（8 個すべての特徴量を用いて検索し上位 10 位まで）を示す。

→ 以下の図 35 から図 45 は、私の学籍番号の下二桁 (01) と同じ下二桁の図形番号を持つ画像をキーとした時の検索結果である。また、以下の表 2 は、私の学籍番号の下二桁 (01) と同じ下二桁の図形番号を持つ画像をキーとした時の検索結果の表である。

図 35　検索キー

図 36　第 1 位


図 37　第 2 位


図 38　第 3 位

図 39　第 4 位

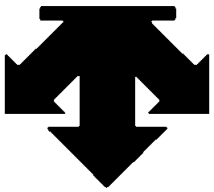

図 40　第 5 位


図 41　第 6 位

図 42　第 7 位

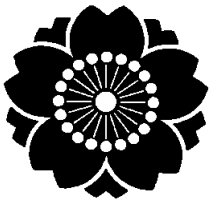

図 43　第 8 位

図 44　第 9 位

図 45　第 10 位

表 2　画像特徴量の計算結果

| 順位 | 図形番号 | 検索キーとの距離 |
|---|---|---|
| 検索キー | 1 | 0 |
| 1 | 3 | 0.415 |
| 2 | 75 | 0.854 |
| 3 | 74 | 0.935 |
| 4 | 2 | 1.088 |
| 5 | 12 | 1.162 |
| 6 | 71 | 1.287 |
| 7 | 9 | 1.377 |
| 8 | 4 | 1.432 |
| 9 | 10 | 1.581 |
| 10 | 6 | 1.694 |

# 5 考察

- 表 2 を見ると、検索結果 1 位と 10 位の検索キーとの距離が「0.415」と「1.694」となっており、約 4 倍も変わっていることが分かる。これは、検索に使う特徴ベクトルが 8 個もあるため、距離の計算の数値の振れ幅が大きくなったからであると考えられる。また、画像が 100 枚しか存在しないため、そもそもの類似画像が少なかったことも原因であると考えられる。
- 検索結果の第 3 位、第 4 位、第 5 位を見比べてみる。
  - 第 3 位の画像は四角形があり、その下に日本語が刻まれている。
  - 第 4 位の画像は円形であり、その中にアルファベットが刻まれている。
  - 第 5 位の画像は下向きの矢印が 3 本伸びている。

  → このように、文字にしてみると第 3 位から第 5 位は全く別の画像のように聞こえるものの、検索キーとの距離を見ると、それぞれ「0.935」、「1.088」、「1.162」となっており、距離は約 0.1 ずつほどしか変わっていないことが分かる。このような結果となった原因として、今回の課題で検索に使われた特徴は類似画像の検索に適していない、または、計算に使う特徴が足りていないということが考えられる。なぜなら、人間から見ると全く違う図形であるにも関わらず、距離が近いということは、「人間が画像を類似しているかどうかを判断している基準」をプログラムが計算に含めてきれていないと考えられるからである。

# 6　むすび

今回の実験では、濃度、縦横比、水平ラン平均、垂直ラン平均、水平ラン標準偏差、垂直ラン標準偏差、重心 x 座標、重心 y 座標の 8 つの特徴を用いて、キー画像の類似画像を検索するプログラムを作成した。その結果として、それらの特徴がキー画像に近かったとしても、人間から見て類似画像であるとは限らないことが分かった。今後、課題プログラムにおいて図形の検索精度を上げるためには、検索に使用する特徴の数を増やすことなどが挙げられる。

## 参考文献

[1] https://www.javadrive.jp/start/stream/index4.html

[2] https://www.javadrive.jp/start/scanner/index1.html