麻雀点数計算プログラム 注釈書

14EC004 飯田頌平

2016 年 3 月 10 日

1　はじめに

本稿は吉田さんの麻雀点数計算プログラム [1] の注釈書にあたる。

このプログラムはカメラで麻雀の手牌を撮影すると点数を自動で計算するというものであり、大まかにサーバ、画像認識、点数計算の三つのモジュールに分かれている。

このうち本稿では画像認識の部分について触れる。画像認識の実装ファイル TemplateMatching.scala およびユーザ定義関数の実装ファイル package.scala を付録に掲載するので、参考にしながら読むこと。

2　プログラムの実行方法

プログラムは github 上に上げられているが、それを clone するだけでは動作しない。sbt をインストールした上で、opencv を使用できるようにせねばならない。

Ubuntu であれば、sbt は以下のコマンドでインストールできる。

ソースコード 1　sbt

```
1 $ sudo apt-get install sbt
```

opencv は各自自力でコンパイルする必要がある。公式ドキュメントを参考にして、以下のコマンドを実行して.so ファイルを生成する。

ソースコード 2　opencv

```
1 $ git clone git://github.com/Itseez/opencv.
    git
2 $ cd opencv
3 $ mkdir build
4 $ cd build
5 $ cmake -DBUILD_SHARED_LIBS=OFF
6 $ make -j8
```

.so ファイルは opencv/build/lib に生成されるはずである。もし生成されていない場合は、cmake の出力結果を確認すること。To be built の項目に java とない場合は、java-8-oracle の環境変数を設定できていないので、

ソースコード 3　JAVAHOME

```
1 $ JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

を実行すること。

生成された.so ファイルを、clone してきた mahjongs ディレクトリの下の mahjongs/recognisser/lib に置けば sbt 環境下で opencv を使用することができるようになる。

3　画像認識のフロー

画像認識はパターンマッチングによって行われている。よって、最初に訓練データからテンプレートを生成し、その後テストデータとテンプレートを照合させて結果を求める。まずはテンプレートの生成手順を示す。

1. 雀牌を黒と白で二値化する
2. 雀牌の輪郭を判別し、雀牌だけを切り取る
3. 雀牌から牌ひとつあたりの縦幅と横幅を求める
4. 白と判別された誤差（牌の隅）を黒く塗りつ

ぶす
5. 牌の絵柄部分の輪郭情報を得る
6. 輪郭情報と雀牌のグレースケール画像からテンプレートを生成する

詳細な説明については後述するとして、次に照合の手順を示す。

1. 手牌の輪郭を判別し、手牌（純手牌＋鳴牌）だけを切り取る
2. テンプレートと同じサイズにリサイズする
3. 取得した輪郭の頂点数を求める（四角形なら純手牌である）
4. テンプレートから各牌を回転させたイメージを得る
5. 手牌とテンプレート（及び回転イメージ）をテンプレートマッチングで照合する
6. 照合結果に牌種のシーケンスを与え、結果の最大スコアを取り出す
7. 最大スコアを基に矩形を作成し、手牌を矩形領域で塗りつぶす
8. ５．６．７．の手順を再帰的に繰り返すと、手牌が黒く塗りつぶされてゆく。完全に塗りつぶされたら、再帰を終了する
9. 純手牌を第一要素、鳴牌を第二要素のタプルを作り、関数の結果として返す

こうして手牌を認識することができた。次章以降ではこれらの流れの詳細について述べる。

4　テンプレートの生成

テンプレートの生成にあたって、まずは元画像を用意する。以下にその要件を示す。

- 雀牌を 4*9 の矩形上に隙間なく並べる。矩形は平行になるようにする
- 牌の位置は固定である。一萬からはじまる SEQ が予め定められている
- 雀牌と背景以外のものを除外する
- 撮影時に、各牌が等しい高さと幅を持つよう注意する
- 二値化からの境界値探索を阻害しないよう背景を選ぶ

背景には麻雀マットなどを用いれば良い。また斜めから撮影された画像からは正しいテンプレートが得られないことに気を付ける。

元画像はプログラムへ Mat 型で渡される。Mat 型は opencv[*1]のライブラリが提供する行列の型であり、opencv ではこの型で画像データを扱う。Mat 型の内部構造は行列、画素値、画素のデータ型である。createTemplate の引数 mat が元画像のデータを示す変数である。

mat に対して最初に行われる処理は、定数 MaxResolution のサイズに近づくよう resize で画像サイズを修正する。大きい画像であると処理時間が増えてしまうため、これによって効率化する。

次に雀牌と背景を識別し、雀牌だけを切り取る crop を実行する。crop は元画像を引数に取り、Buffer 型のシーケンスで識別結果の画像を返す。（なお Buffer はタプルであり、画像だけでなく輪郭の二次元配列も返しているが、この値は createTemplate では参照しない。）

crop の実装は、まず findContours によって元画像を輪郭で分離し、minAreaRect で矩形を得て、warpAffine で矩形の傾きを底面と垂直になるよう修正し、getRectSubPix で矩形領域のピクセル値を得ることで、入力画像を輪郭毎に分離した上で補正をかけた画像を得るようになっている。

白と黒で二値化されたテンプレート画像は、牌の集合の縁は白、背景は黒と別の色で分けられているため、findCountours によって輪郭を得られるようになっている。得た輪郭 countour のメソッド toArray を用いれば、minAreaRect で矩形を引くべき座標を得られるのである。

こうして得られた分離後の画像のシーケンス（返り値は Buffer 型であることに注意）から headOption を使うことで先頭要素を取り出せる。findContours

[*1] https://github.com/bytedeco/javacv

では大きさ順に画像をソートする作用があるので、先頭要素には一番大きな画像が入る。一番大きな画像が雀牌であるという前提であるなら、この先頭要素は雀牌であるため、雀牌のテンプレートは引数 template に束縛される。

threshold によってテンプレートを黒と白で二値化し、width と height に牌ひとつあたりの幅と高さを代入する。このとき雀牌の並びが 4*9 でなかったとき、想定外の値を取ってしまうので注意すること。

こうして得られた二値化後の画像について見てみると、牌の模様が白、牌の背景が黒で表されていることが分かる。しかしそれと同時に、牌と牌の間なども白く判定されてしまっている。牌の模様以外が白と認識されてはパターンマッチに失敗するため、次はこの誤判定部分を黒く塗りつぶす。それには floodFill を使う。floodFill は座標を指定し、連結部分を指定した色で塗りつぶす関数である。これを牌と牌の間にすべてに対して実行することで、誤判定が塗りつぶされる。floodFill の補完として、dilate を使用する。dilate は膨張処理によってノイズを消す効果があり、ターゲットの周辺に 1 ピクセルでも白があれば、ターゲットを白に置き換える処理を行う。これによって牌の模様をはっきりとさせる。

そして牌の模様だけを抽出する。findContours によって再び輪郭ごとの分離をかければ、模様だけを抽出することができる。四筒のように模様が複数に分離されてしまうケース（二筒ふたつとして認識される）の場合は、convexHull によって結合する。convexHull は点群の凸包を得た後に外接する矩形で近似する。最後に、得られた模様の中で、もっとも面積の大きなものを size に代入する。マッチングを行う際にはこのサイズを基準にするためである。

ここまでで二値化を通して牌の模様の輪郭情報を抽出する処理を行った。今度はこれをグレースケール画像に適用し、牌一種類ごとのテンプレートを得る。

それには grid を用いる。grid は submat を実装に含み、元画像から矩形範囲を抽出できる関数である。grid で牌を 4*9 に区切り、zip で牌ごとに模様の輪郭の情報を付与し、それらに flatMap を挟んで getRectSubPix をかけることによって、輪郭だけが抽出されたすべての牌のテンプレートを得ることができる。

以上でテンプレートの生成が完了した。

5　テストデータとの照合

照合は関数 recognize によって実装されている。この引数 mat に手牌の画像データを、templates にテンプレートの画像データを渡す。

手牌のデータは crop によって背景から切り離される。しかしテンプレートを crop で切り離したときと違い、この場合は牌のデータが複数に分かれる。手牌には純手牌と鳴牌があり、鳴牌は純手牌と離して置くためだ。

手牌のデータは collect によって hand に束縛される。hand をテンプレートのサイズに合わせてリサイズし、頂点数を求め純手牌か否かの情報を edge に代入して、テンプレートに向きを回転させた牌を加えて tiles に代入すると、いよいよ go でパターンマッチを行う。

go の内部では、まず各テンプレートに対して opencv のテンプレートマッチング関数たる matchTemplate を呼ぶ。その結果が result へと代入されるので、minMaxLoc を通して最高スコアと最低スコアの情報を取り出し、使用したテンプレートとスコア情報のタプルを locs に加えてゆく。この手順をすべてのテンプレートに対して実行する。minMaxLoc の返り値はドキュメントによると [2] Core.MinMaxLocResult 型である。座標を返す maxLoc フィールドと値を返す maxVal フィールドを参照することで最高スコアの情報を得ることができる。

すべてのテンプレート（及びその回転）に対してのマッチングの結果 locs から、loc 値が最大となるものを maxBy によって取得し、変数 ((tile,loc),i) に代入する。スコアが最大となる位置は loc.maxLoc で参照でき、それにテンプレートの大きさ tile.size を組み合わせることで、マッチした手牌の輪郭 rect を得られる。rect の実装では、最大スコアの場所とサ

イズから、基準点と縦横のサイズ情報を持つ矩形を作成している。

ここで go の引数を見てみる。go の引数は rects であり、それは rect と、そのインデックス番号のタプルのシーケンスである。go は再帰を前提としており、このシーケンス rects は再帰のたびに認識結果を積み重ねてゆく。

rectangle によってマッチングした牌の輪郭部分に矩形を描き、矩形を領域を塗りつぶすと、go を再帰する。このときシーケンスである引数 rects にマッチした牌 rect を加えており、rects はすべての牌についてのマッチングが終了した時には関数 go の返り値となる。

矩形領域がすべて塗りつぶされたあとは、適当な地点が最大スコアとして返り、そこからそのループでの rect が得られるが、intersects によって現在の rect と過去の rect（rects に保存）の矩形領域が重なっているか否かの判別を行い、重なっていた場合にマッチング終了という判定を下す。

マッチングを終えたら結果を手牌の位置順にソートし、シーケンス情報だけを取り出し indices に代入する。最後に純手牌か否かの判別を改めて行い、その結果と indices のタプルを result に代入していく。

result は純手牌および鳴牌のまとまり毎に要素数がひとつずつ増える構造であり、最終的に、純手牌と鳴牌のシーケンスのタプルが recognize の返り値となる。

6　matchTemplate について

opencv におけるテンプレートマッチングを行うメソッド matchTemplate は、以下の引数を取る。

1. image, テンプレートの探索対象となる画像。8 ビットまたは 32 ビットの浮動小数点型
2. templ, 探索されるテンプレート。image と同じデータ型である必要がある
3. result, 比較結果のマップ
4. method, 比較方法の指定

プログラムでは method に TM_CCORR_NORMED を指定している。これは数式で表すと

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I'(x+x', y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}}$$

図 1　相互相関関数

であり、一般に相互相関関数と呼ばれる式である。

この式の意味するところは、テンプレートと対象画像の内積である。相互相関関数の式の左辺を cos θ に、右辺を内積とゲインの商だと見れば、まさしくベクトルの内積を示していることがわかる。相互相関関数の左辺は内積の角度 cos θ を示しており、cos θ が 1 に近づけば近づくほど、θ は小さく、テンプレートと対象画像の領域が近づいていることを意味する。

各ピクセルに対して内積を求める操作は行われ、その総和を以ってスコアとするので、スコアが大きければ大きいほど、テンプレートとマッチしているという結果になる。この点は最小誤差によってマッチングを行う methodSQDIFF とは逆の意味になっているので、関数 matchTemplate の結果 result には注意が必要である。

matchTemplate の結果である result については、スコアのマップ形式で返される。マップについて説明する。そもそもテンプレートマッチングとは、テンプレート画像と対象画像を重ね合わせて画素の差を比較するという行為を、各ピクセルに対して行うものである。つまり重ねあわせが行われる範囲でマッチングの結果が得られるのであって、それは対象画像のサイズとテンプレート画像のサイズの差の範囲である。この範囲の行列をマップと呼ぶ。（この呼び名は opencv のドキュメント [3][4] に即する）そしてマップの値にはテンプレートマッチングのスコア（相互相関関数の値）が入るのである。

この理論は実際に画像を用意して matchTemplate を行い、元画像・テンプレート画像のサイズとマップのサイズを比較すれば裏付けられる。マップのサイズは縦横ともに元画像とテンプレート画像の差に

等しい結果が得られる。そしてこの結果を画像形式に落とし込んだものが、吉田さんの梗概集の図 5 に示されたマッチング結果である。マップの値たる相互相関関数の結果は、0 から 1 の範囲で表されているため、そのまま opencv の出力関数を使うと一面真っ黒な画像（図 2）が得られてしまうが、255 を乗ずることで、opencv 標準の 0 から 255 までの範囲での画素の表現に対応するようになるため、吉田さんのマッチング結果のような、縞模様状の結果（図 3）を得ることができる。

図 2　相互相関関数出力結果 1

図 3　相互相関関数出力結果 2

参考文献

[1] Sanshiro Yoshida. mahjongs.
https://github.com/halcat0x15a/mahjongs

[2] OpenCV 2. 4. 2 Java API. Core. MinMaxLocResult.
http://docs.opencv.org/java/2.4.2/index.html?org/opencv/core/Core.html

[3] 物体検出− opencv2.2 documentation.
http://opencv.jp/opencv-2svn/cpp/object_detection.html

[4] Object Detection − opencv2.4.10.
http://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html

7　付録

7.1　画像認識

ソースコード 4　TemplateMatching.scala

```
1  package mahjongs.recognizer
2  import org.opencv.core._
3  import org.opencv.imgproc.Imgproc
4
5  object TemplateMatching {
6
7    val MaxResolution: Int = 1024 * 1024
8
9    def createTemplate(mat: Mat): Option[(IndexedSeq[Mat], Size)] = {
10     crop(resize(mat, MaxResolution)).headOption.map {
11       case (template, _) =>
12         val mask = threshold(template.clone, true)
13         val width = template.cols / 9
14         val height = template.rows / 4
15         floodFill(mask, (0 until 4).map(_ * height) :+ (mask.rows - 1), 0 until mask.cols)
16         floodFill(mask, 0 until mask.rows, (0 until 9).map(_ * width) :+ (mask.cols - 1))
17         Imgproc.dilate(mask, mask, new Mat)
18         val rects = for (tiles <- grid(mask, 4, 9)) yield {
19           for (tile <- tiles) yield {
20             val contours = findContours(tile.clone).flatMap(_.toArray)
21             if (contours.length > 0)
22               Some(convexHull(contours))
23             else
24               None
25           }
26         }
27         val size = rects.flatten.flatten.map(_.size).maxBy(_.area)
28         val tiles = grid(template, 4, 9).zip(rects).flatMap {
29           case (tiles, rects) =>
30             tiles.zip(rects).map {
31               case (tile, rect) =>
32                 Imgproc.getRectSubPix(tile, size, rect.fold(center(tile))(center), tile)
33                 tile
34             }
35         }.take(34)
36         (tiles, new Size(width, height))
37     }
38   }
39
40   def recognize(mat: Mat, templates: Seq[Mat], width: Int, height: Int): (Seq[Int], Seq[Seq[Int]]) = {
41     val result = crop(resize(mat, MaxResolution)).collect {
42       case (hand, contour) if hand.size.area > 0 =>
```

```
43          Imgproc.resize(hand, hand, new Size(hand.size.width * height / hand.size.height, height))
44          val edge = approxPoly(new MatOfPoint2f(contour.toArray: _*)).rows
45          val tiles = templates ++ templates.map(m => flip(m.t, 0)) ++ templates.map(flip(_, -1)) ++
                templates.map(m => flip(m.t, 1))
46          def go(rects: List[(Int, Rect)]): List[(Int, Rect)] = {
47            val locs = for (tile <- tiles) yield {
48              val result = new Mat
49              Imgproc.matchTemplate(hand, tile, result, Imgproc.TM_CCORR_NORMED)
50              (tile, Core.minMaxLoc(result))
51            }
52            val ((tile, loc), i) = locs.zipWithIndex.maxBy(__1._2.maxVal)
53            val rect = new Rect(loc.maxLoc, tile.size)
54            if (rects.forall(pair => !intersects(rect, pair._2))) {
55              Imgproc.rectangle(hand, rect.tl, rect.br, new Scalar(0), -1)
56              go((i % 34, rect) :: rects)
57            } else {
58              rects
59            }
60          }
61          val indices = go(Nil).sortBy(__2.x).map(__1)
62          (edge == 4 && indices.size != 4, indices)
63      }.groupBy(__1).mapValues(_.map(__2))
64      (result(true)(0), result.get(false).toList.flatten)
65    }
66
67  }
```

7.2 ユーザ定義関数

ソースコード 5　package.scala

```
1  package mahjongs
2  import scala.collection.JavaConverters._
3  import scala.collection.mutable.Buffer
4  import org.opencv.core._
5  import org.opencv.imgcodecs.Imgcodecs
6  import org.opencv.imgproc.Imgproc
7
8  package object recognizer {
9
10   for {
11     ext <- sys.props("os.name").toLowerCase match {
12       case name if name.contains("nix") => Some("so")
13       case name if name.contains("mac") => Some("dylib")
14       case _ => None
15     }
16   } System.load(getClass.getResource(s"/libopencv_java300.$ext").getPath)
17
```

```
18  def findContours(mat: Mat): Buffer[MatOfPoint] = {
19    val contours = Buffer.empty[MatOfPoint]
20    Imgproc.findContours(mat, contours.asJava, new Mat, Imgproc.RETR_EXTERNAL, Imgproc.
          CHAIN_APPROX_TC89_KCOS)
21    contours.sortBy(Imgproc.contourArea)(Ordering.Double.reverse)
22  }
23
24  def floodFill(mat: Mat, rows: Seq[Int], cols: Seq[Int]): Mat = {
25    val mask = new Mat
26    val color = new Scalar(0)
27    for (row <- rows; col <- cols if mat.get(row, col)(0) > 0)
28      Imgproc.floodFill(mat, mask, new Point(col, row), color)
29    mat
30  }
31
32  def grid(mat: Mat, rows: Int, cols: Int): IndexedSeq[IndexedSeq[Mat]] = {
33    val height = mat.rows / rows
34    val width = mat.cols / cols
35    for (row <- 0 until rows) yield {
36      val rowRange = new Range(row * height, (row + 1) * height)
37      for (col <- 0 until cols) yield
38        mat.submat(rowRange, new Range(col * width, (col + 1) * width))
39    }
40  }
41
42  def approxPoly(contour: MatOfPoint2f, epsilon: Double = 0.01): MatOfPoint2f = {
43    val curve = new MatOfPoint2f
44    Imgproc.approxPolyDP(contour, curve, Imgproc.arcLength(contour, true) * epsilon, true)
45    if (curve.rows % 2 == 0)
46      curve
47    else
48      approxPoly(contour, epsilon + 0.01)
49  }
50
51  def threshold(mat: Mat, inv: Boolean): Mat = {
52    val (tpe, op) = if (inv) (Imgproc.THRESH_BINARY_INV, Imgproc.MORPH_OPEN) else (Imgproc.
          THRESH_BINARY, Imgproc.MORPH_CLOSE)
53    Imgproc.threshold(mat, mat, 0, 255, tpe | Imgproc.THRESH_OTSU)
54    Imgproc.morphologyEx(mat, mat, op, new Mat)
55    mat
56  }
57
58  def crop(mat: Mat): Buffer[(Mat, MatOfPoint)] = {
59    for (contour <- findContours(threshold(mat.clone, false))) yield {
60      val patch = new Mat
61      val rect = Imgproc.minAreaRect(new MatOfPoint2f(contour.toArray: _*))
62      Imgproc.warpAffine(mat, patch, Imgproc.getRotationMatrix2D(rect.center, rect.angle, 1), mat.size)
63      Imgproc.getRectSubPix(patch, rect.size, rect.center, patch)
64      if (rect.angle <= -45) Core.flip(patch.t, patch, 0)
```

```
65        (patch, contour)
66      }
67    }
68
69    def convexHull(contours: Seq[Point]): Rect = {
70      val hull = new MatOfInt
71      Imgproc.convexHull(new MatOfPoint(contours: _*), hull, false)
72      Imgproc.boundingRect(new MatOfPoint(hull.toArray.map(contours): _*))
73    }
74
75    def flip(mat: Mat, code: Int): Mat = {
76      val m = new Mat
77      Core.flip(mat, m, code)
78      m
79    }
80
81    def resize(mat: Mat, max: Int): Mat = {
82      val r = math.sqrt(mat.rows * mat.cols / max.toDouble)
83      if (r > 1) Imgproc.resize(mat, mat, new Size(mat.size.width / r, mat.size.height / r))
84      mat
85    }
86
87    def center(mat: Mat): Point =
88      new Point(mat.size.width / 2, mat.size.height / 2)
89
90    def center(rect: Rect): Point =
91      new Point(rect.x + rect.width / 2, rect.y + rect.height / 2)
92
93    def toMat(bytes: Array[Byte], gray: Boolean): Mat =
94      Imgcodecs.imdecode(new MatOfByte(bytes: _*), if (gray) Imgcodecs.
            CV_LOAD_IMAGE_GRAYSCALE else Imgcodecs.CV_LOAD_IMAGE_COLOR)
95
96    def fromMat(mat: Mat): Array[Byte] = {
97      val buf = new MatOfByte
98      Imgcodecs.imencode(".png", mat, buf)
99      buf.toArray
100   }
101
102   def intersects(a: Rect, b: Rect): Boolean =
103     math.max(a.x, b.x) < math.min(a.x + a.width, b.x + b.width) && math.max(a.y, b.y) < math.min(
            a.y + a.height, b.y + b.height)
104
105   def read(filename: String, gray: Boolean): Mat =
106     Imgcodecs.imread(filename, if (gray) Imgcodecs.IMREAD_GRAYSCALE else Imgcodecs.
            IMREAD_COLOR)
107
108 }
```