パターン認識入門

1．カラー画像からグレースケール画像を作る方法

カメラに限らず、昨今コンピュータで取り扱われる画像は一般的にはカラー画像です。

パターン認識においてカラー画像をそのまま用いる場合もありますが、単にコンピュータに形を認識させたりあるいは今回のアプリケーションのように文字を認識させたりしたい場合には、カラー画像をそのまま使うよりもグレースケール画像を認識に用いた方がアルゴリズムが簡単になります。ここで言うグレースケール画像というのは、明るさの度合いのみによって構成された画像のことです。

この明るさ l は、赤、緑、青の各色成分の明るさをそれぞれ r 、 g 、 b とすると、

$$l = w_r r + w_g g + w_b b$$

で表せます。ここで w_r 、 w_g 、 w_b は各色成分の重みです。

実際に人間の目が感じる明るさは r 、 g 、 b の値に応じて w_r 、 w_g 、 w_b も異なるのですが、通常これらを定数とします。特に、

- $w_r = 0.299$
- $w_g = 0.587$
- $w_b = 0.114$

の値が用いられます。

この処理の結果は、具体的には次のようになります。

Before

After

2．背景と文字の区別

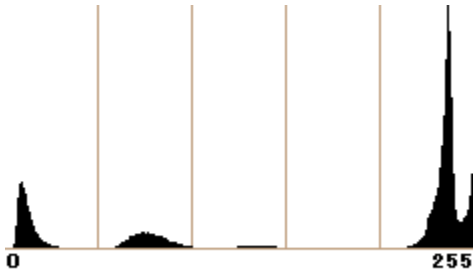
紙などに印刷された文字や図形をコンピュータに認識させるためには、認識させたい対象とその周りにある背景とをコンピュータに区別させなければなりません。これを 2 値化

と言います。つまり認識の対象を 1 で、背景を 0 と符号化するのです。この文字認識アプリケーションでは 2 値化に判別分析法と呼ばれる方法を採用しました。

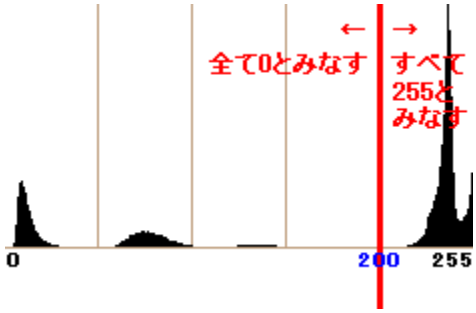
判別分析法は、2 つの山からなる明るさに関するヒストグラムから谷の部分を探し出す手法です。因みにヒストグラムとは横軸に明るさを、縦軸にその明るさを持つ画素の数をとった棒グラフのことです。

例えば、次の

という画像では、ヒストグラムは次のようになります。



このヒストグラムでは、左側が上の画像の黒い部分に、右側が白い部分に対応します。このヒストグラムで、明るさが 200 未満の点の明るさを全て 0 とし、逆に明るさが 200 以上の点の明るさを全て 255 としてみます。



すると、上の画像は、

となります。

この例の 200 という値、これを閾値と言いますが、この値を人が決めました。これをコンピュータが自分で判断するためのアルゴリズムの 1 つが、判別分析法なのです。そして、閾値で分けた後の 0 という値の代わりに 1 を、255 という値の代わりに 0 という値を用いれば、処理後の画像の黒い点が 1 で、白い点が 0 で表されることになり、結果的に文字と背景とが分離できるのです。

この判別分析法にはヒストグラムに谷が存在しなかったり、あるいは谷が複数存在しても多くの場合適切な閾値が得られるという特徴があります。

それでは判別分析法による閾値の求め方を説明します。

画像の明るさが 0 から N までの数値で表現されているとします。そして、明るさが t である画素の数を $f(t)$ で表すことにします。この $f(t)$ こそ、ヒストグラムそのものです。

まず明るさ t を境に明るさが t 未満のものをクラス 1、明るさが t 以上のものをクラス 2 としてヒストグラムを 2 つのクラスに分けたとします。

クラス 1 に含まれる画素の数 $n_1(t)$ は、

$$n_1(t) = \sum_{j=0}^{t-1} f(j)$$

です。またクラス 1 に含まれる画素が持つ明るさの平均値 $\mu_1(t)$ は、

$$\mu_1(t) = \frac{1}{n_1(t)} \sum_{j=0}^{t-1} jf(j)$$

です。これからクラス 1 の明るさに関する分散 $\sigma_1(t)^2$ は、

$$\sigma_1(t)^2 = \frac{1}{n_1(t)} \sum_{j=0}^{t-1} f(j)\left(j - \mu_1(t)\right)^2$$

と求まります。

同様にクラス 2 に含まれる画素の数 $n_2(t)$、明るさの平均値 $\mu_2(t)$、分散 $\sigma_2(t)^2$ はそれぞれ、

$$n_2(t) = \sum_{j=t}^{N} f(j)$$

$$\mu_2(t) = \frac{1}{n_2(t)} \sum_{j=t}^{N} jf(j)$$

$$\sigma_2(t)^2 = \frac{1}{n_2(t)} \sum_{j=t}^{N} f(j)\left(j - \mu_2(t)\right)^2$$

となります。

これらの値を使って、クラス内分散 $\sigma_W(t)^2$ とクラス間分散 $\sigma_B(t)^2$ とを次のように定義します。

$$\sigma_W(t)^2 = \frac{n_1(t)\sigma_1(t)^2 + n_2(t)\sigma_2(t)^2}{n_1(t) + n_2(t)}$$

$$\sigma_B(t)^2 = \frac{n_1(t)\left(\mu_1(t) - \mu\right)^2 + n_2(t)\left(\mu_2(t) - \mu\right)^2}{n_1(t) + n_2(t)}$$

ただし μ は画像全体の明るさの平均値で、

$$\mu = \frac{\sum_{j=0}^{N} jf(j)}{\sum_{k=0}^{N} f(k)}$$

です。

更に $W(t)$ を、

$$W(t) = \frac{\sigma_B(t)}{\sigma_W(t)}$$

とします。

0 から N までの全ての t に対して $W(t)$ を計算し、$W(t)$ が最大となる $t = t_{MAX}$ を見つけます。

この t_{MAX} を 2 値化の閾値と採用します。これが判別分析法です。

今回の文字認識アプリケーションでは文字は暗い色に、背景は明るい色になっていると想定して、各点の明るさがこの t_{MAX} 未満であれば文字の一部であり、そうでない場合は背景であると見なすようにしています。

3．ノイズ対策

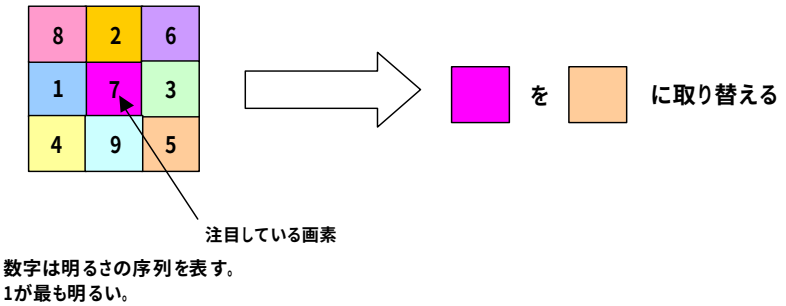
カメラやスキャナで取り込んだ画像は、一般的には多くのノイズを含んでいます。人間の目では分からないのですが、判別分析法による閾値がこのノイズに大きく左右されて背景と文字を上手く区別できないことがしばしばです。

ここでは簡単なノイズ対策に触れておきます。

3.1. メディアンフィルターによるノイズ除去

ノイズを除去する簡単な方法は、メディアンフィルターを掛けることです。

メディアンフィルターとはある点とその周辺の全 9 点の明るさを調べ、その中で 5 番目の明るさをその点の明るさとする画像処理のことです。



ここで 1 つ注意があります。それはこの変換では元の画像から新しい画像を作っていることです。決して 1 つの画像上で処理を行っているわけではありません。

このメディアンフィルターにより、パルスノイズをある程度除去可能です。

この処理結果の具体例は、次のようになります。

Before

After

3.2. 輪郭付近の画像に対して判別分析法を適用する

画像の中で意味を持つのはやはり輪郭です。多くの場合輪郭以外の部分は大した情報を含んでいません。無意味な部分に乗ったノイズの影響で有益な情報が取り出せなくなってしまっては困ります。そこで、先に挙げた判別分析法を適応する箇所を輪郭周辺に限定しましょう。

しかしここで新たな問題が発生します。どうやって輪郭を探すのか、という問題です。

輪郭を検出する手法にはいろいろありますが、今回はラプラスフィルターによって輪郭を検出することにしました。

説明の都合上ここでは画像を行列で表現することにします。行列 A に関してその (i, j) 成分を A_{ij} と書きます。画像を行列で表現しますので、点 (x, y) の画素が持つ明るさが A_{yx} のように添え字が逆に書かれることに注意してください。但し座標に合わせるため、行の添え字も列の添え字も共に 0 から始めることにします。

ラプラスフィルターもメディアンフィルターと同様に、注目している画素とその周辺あわせて 9 つの画素を見ます。

まず、次のような 3×3 行列 L を定義します。

$$\begin{cases} L_{ij} = 1 & ((i,j) \neq (1,1)) \\ L_{11} = -8 \end{cases}$$

図に描くとこんな感じです。

1	1	1
1	-8	1
1	1	1

元となる画像を A、ラプラスフィルターにより得られる画像を B とすると、

$$B_{ij} = \sum_{k=0}^{2} \sum_{l=0}^{2} L_{kl} A_{(i-k-1)(j-l-1)}$$

という関係になります。得られた画像 B の画素の中で明るいものが輪郭を表します。

具体的には、次のようになります。

Before

After

こうして得られた輪郭に対して先に説明した判別分析法を用いると、背景と文字を精度良く区別できます。

4. 輪郭線の検出

パターン認識に図形全体の情報を用いることもありますが、今回の文字認識アプリケーションでは対象となる文字の一番外側の輪郭線をなぞり、その形で文字を認識しています。この方法では輪郭線が 1 本の閉曲線になっていないと正しく認識できません。

3.2.でラプラスフィルターによる輪郭検出を説明しましたが、ラプラスフィルターで検出される輪郭は一般に幅を持っています。幅を持った輪郭線の幅をなくす処理も原理的には可能ですが、今回は簡単に、周囲にある文字を構成している点の数を画像中の各点について調べ、その数が 4〜6 であるものを輪郭であると見なすようにしました。2. の 2 値化

処理後の画像で言えば、値が 1 である全ての点についてその周囲 8 点にある値が 1 である点の個数を調べ、その数が 4～6 であれば、その点は輪郭であると判断するのです。

0	1	2	2	1	0	0
0	2	3	4	2	1	0
1	4	6	6	5	3	3
1	3	6	8	7	4	2
2	4	6	6	6	3	2
1	1	4	3	4	2	1
1	1	2	1	1	0	0

数字はその点の周囲8マスにある図形を構成する点の数を、
青色は図形を構成する点を、黄色は図形を構成する点のうち
輪郭と見なされる点を、ピンク色は図形の一部ではないが輪
郭と見なされる点を表す。

簡単ですね。

5. 複素 PARCOR(パーコル)係数法

輪郭も検出できたので、あとは図形の形を捉えることができればパターン認識が可能になります。今回はパターン認識に「複素 PARCOR 係数法」を用いました。

パターン認識の分野で複素 PARCOR 係数と言えば、ある図形についてその値を計算すると、その値がその図形を象徴する値となる値のことです。

この文字認識アプリケーションでは、あらかじめ見本となるアルファベットの大文字 A～Z について複素 PARCOR 係数を計算して保存しておきます。そして、カメラで撮影した画像から PARCOR 係数を計算して、見本の文字の中から最も近い PARCOR 係数を持つものを選択することで、文字認識を行っています。

この辺りの理論については専門書に譲ることにして、ここでは複素 PARCOR 係数の計算方法について簡単に説明します。

まず 4. で検出された輪郭について、認識に使う点を決めましょう。全部使うことも出来ますが、その場合図形の拡大や縮小の影響を受けますので、今回は輪郭を反時計回りに 1 周しながら、等間隔に 128 個取り出すようにしました。以後説明のため、取り出す輪郭上の点の数を N 個としておきます。

次にその N 個の点について周回方向を決め、0 から $N-1$ まで順に番号付けします。j 番目の点の座標を (x_j, y_j) と表すことにします。ただしこの座標はこの図形の重心からの相対座標です。具体的には、画像上の実際の座標が (x'_j, y'_j) であるとき重心の座標 $(\bar{x}, \bar{y})$ は

$$\bar{x} = \frac{1}{N}\sum_{j=1}^{N} x_j, \quad \bar{y} = \frac{1}{N}\sum_{j=1}^{N} y_j$$

であり、

$$x_j = x'_j - \bar{x}, \quad y_j = y'_j - \bar{y}$$

です。

各点を複素平面上の点と考えて、各点の座標を複素数$\{z_j\}$で表します。

$$z_j = x_j + iy_j$$

そして、複素相関係数$\{r_j\}_{j=0}^{M}$を次のように定めます。

$$r_j = \sum_{k=0}^{N-1} z_k z^*_{(j+k) \bmod N}$$

ここで ”$n \mod m$” は “nをmで割った余り” を表します。また、z^*のように文字の上にアスタリスクがついたものは、その共役複素数を表します。

最後に、行列Aと複素 PARCOR 係数$\{p_j\}_{j=1}^{M}$を以下の規則で計算して行きます(行列の添え字は 1 から始まります)。

- $j < k$ のとき $A_{jk} = 0$
- $p_1 = A_{11} = \dfrac{r_1}{r_0}$
- $j \geq 2$ のとき

$$p_j = A_{jj} = \frac{r_j - \sum_{k=1}^{j-1} r_{j-k} A_{(j-1)k}}{r_0 - \sum_{k=1}^{j-1} r_k A^*_{(j-1)k}}$$

$$A_{jk} = A_{(j-1)k} - p_j A^*_{(j-1)(j-k)} \quad (k < j)$$

こうして得られた複素 PARCOR 係数$\{p_j\}_{j=1}^{M}$はその図形を拡大・縮小をしても、回転させても、平行移動しても変わりません。

あとは識別したい文字や図形について予め計算しておいた複素 PARCOR 係数と比較すれば、認識作業は完了です。