基本的な OCR の方法

本書ではOCRの効率的な学習方法およびその理論的背景について詳しく説明します。a~z、A~Z および 0~9 の 62 文字を認識するまでの手順を順おって説明します。OCR を初めて行う人、また OCR の認識率でお悩みの方はこの文章が何らかのヒントになる事でしょう。

1. 学習用紙

まず本書で認識する文字を定義します。本書では以下の文字を認識する事を目的とします。

a ~ z（26 文字） A ~ Z（26 文字） 0 ~ 9（10 文字） → 合計 62 文字

フォント arial と time

どちらのフォントであっても 62 文字を認識する事を目的とします。

まず OCR の学習を行わなければいけません。OCR には認識する文字の学習が不可欠です。またこの学習精度が認識精度を大きく左右します。そこで精度の良い学習を行う為に以下の用紙を用意して下さい。

① 適当なワープロソフトを使用して、縦方向に a~9 の 62 文字を並べ横方向にそれぞれの文字を 20 文字並べます。用紙サイズとフォントサイズに指定はありませんが、学習プログラムを簡潔にする為に 1 枚の用紙に 62 文字が収まるようにして下さい。可能ならあまり行間を詰めない方が後の文字の分割が楽になるかもしれません。

② 上記の用紙を 5 枚程度プリンターで印刷して下さい。

③ ②の用紙をコピー機でコピーして下さい。この時コピーした用紙を次のコピーに使用して下さい。またコピー機の濃度を変化させて下さい。これを 15 枚程度作成します。

④ 以上の用紙を arial と time のフォントについて作成して下さい。それぞれ 20 枚で合計 40 枚の学習用紙が最終的に用意されます。

『良い学習用紙を作成するために』

良い学習を行うためには状態の異なる（良くない）文字も含めて学習する事が大切です。②と③で文字状態の異なる用紙を作成します。②は微妙に違うプリンターの印刷を考慮します。プリンターの解像度の関係から同じ文字でも微妙に変化があるはずです。③は正に文字を劣化させます。コピー機でコピーを作成する事は必ず文字の劣化を生じます。コピーした用紙を次のコピーに使用する事でノイズを積算します。コピー濃度を変化させる事で文字の濃淡が変化します。③でノイズや文字の切断が発生しますがこれらも含めて学習を行います。

2. 画像データ

スキャナーで学習用紙の画像データを撮影します。以下の条件で撮影を行って下さい。

① 1 字の高さが 20～40pixel として下さい。この理由については後で説明を行います。

② 256 グレイ値の画像として撮影を行って下さい。グレイ値の特徴量はバイト画像を基準としています。

③ 保存は bmp か tiff で行って下さい。HALCON で読み込めるファイル形式です。

今回は A4 の用紙に対して解像度 300dpi で撮影し bmp 形式で保存しました。この場合 1 画像が 8.7MB とかなり大きくなります。

『実際の検査システムにおける画像の取得』

実際の検査システムではその環境で撮影された文字を含む画像を使用します。今回のように 1 枚の画像中に文字がきちんと整列している事は少ないと考えられます。良い学習を行う為に実際の検査環境で多くの画像を取得する事が重要です。

3. 文字の分割(セグメンテーション)

次に画像処理に入ります。ここで OCR に対して 1 つの重要な認識が必要です。OCR は文字の分割と判定の 2 段階で構成されると言うことです。文字の分割の成功なくしては OCR の成功はありえません。一般に OCR と言う言葉から、文字の高い判定率ばかりを想像されるかもしれませんが、判定は文字を正しく分割した後での処理だと言うことを忘れてはいけません。OCR を行う場合判定以前に文字分割で悩まされる事も多々あります。（本書では文字分割の方法については詳しく述べません。なぜなら文字分割についてはその対象画像によってエッジ処、輝度むら補正、モフォロジーなど多くのテクニックを使い分ける必要があるからです。）

今回の例題で使用する画像は白地に黒の文字なので、その認識は比較的簡単です。簡単な 2 値化やノイズ除

去を行う事で文字を抽出する事ができます。唯一注意が必要な文字は小文字の i です。これは 2 つの領域がありますがこれを 1 つとして認識する必要があります。領域の膨張などを利用して 1 つの領域として認識しています。詳しくはサンプルプログラムを参照して下さい。

4. 文字ファイル作成

文字が分割できました。それぞれの文字の特徴を学します。学習の方法として 2 通りが考えられます。

① クラスファイルを直接作成する方法
② 一度文字ファイルを作成しその後でクラスファイルを作成する方法

今回は②の方法で学習を行います。①の方法は標準サンプルプログラムとして紹介されている方法です。この方法は文字の分割と教示そして学習(クラス分け)を直接行います。②の方法はいったん文字の教示と画像そして領域を文字ファイルに保存し、後でそのファイルから学習を行います。①の方法では再度学習を行う場合(例えば判定のための特徴量を変更する)に画像解析から行う必要がありますが、②の方法では既にある文字ファイルから学習を行う事ができます。また文字やフォントごとにファイルを細かく作成する事で、文字を限定したクラスファイルを作成する事もできます。判定する文字の種類が既知の場合、文字を限定したクラスファイルを指定する事は判定率の向上が期待できます。(例えば arial だけ、time だけ、アルファベットだけ、数字だけなど)

ここで誤解の無いように語句の定義を行います。

- 文字ファイル
  文字ファイルは、その文字が何であるか(例えば a や b… etc.)の教示データ、文字の画像データそして領域データを含みます。文字を判定する為のクラス分けのデータは含みません。
- クラスファイル
  クラス分けされた文字判定に使用するデータを持ちます。それぞれの文字がもつ特徴量がベクトルとして保存されています。このベクトルはクラス分けで指定した特徴量の数だけ存在するため一般に超空間のn ベクトルです。

話がそれましたが、本書では②の方法でクラス分けを行う前段階として文字ファイルを作成します。文字ファイルは最小単位の各文字で作成します。つまり 62×2=124 の学習ファイルを作成します。学習ファイルの作成には append_ocr_trainf を使用します。

append_ocr_trainf (Character, Image : : Class, FileName :)

Character	文字の領域を指定します。
Image	画像を指定します。
Class	教示データを指定します。
FileName	保存するファイル名を指定します。

Character と Class は tuple の形で渡す事も可能です。このオペレータは文字データを追加保存が可能で、同じファイル名が存在する場合にはそのファイルに情報が追加されます。これは文字データを順次蓄えるには非常に便利ですが、全く新しくデータを初めから作成する場合には前のファイルを削除する事に注意が必要です。文字ファイルを作成するプログラムの最初で system_call を使用して文字ファイルを全て削除すると良いかも知れません。繰り返し追加するプログラムを作成し全ての文字を学習ファイルへ保存して下さい。詳細なプログラムの方法はサンプルプログラムを参照して下さい。
最終的に 1 つの文字ファイルには 20 文字×20 枚=400 文字の情報が蓄えられます。今回のプログラムは文字の分割がきちんと行われている事が前提となります。もしそうでないと文字の順番が狂ってしまい間違った文字を保存することになります。文字ファイルを作成する前に文字分割の確認を行った方が良いでしょう。
文字ファイルを作成後にそれが正しい文字を含むかどうか read_ocr_trainf で確認します。

read_ocr_trainf (: Characters : TrainFileNames : CharacterNames)

Characters	蓄えた画像が返されます。
TrainFileNames	読み込む文字ファイルを指定します。
CharacterNames	蓄えた教示データが返されます。

画像と教示データは tuple の形で帰されます。画像を順次表示して教示データと一致しているか一応確認して

下さい。

5. クラス分けの概念

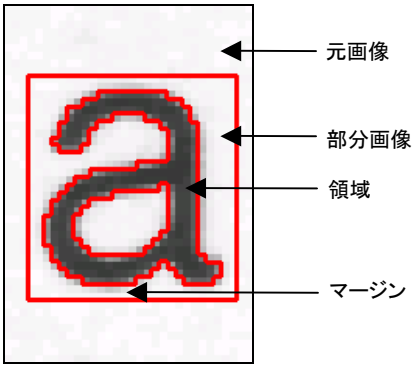
この段階でクラス分けに必要な文字データを取得できました。クラス分けの具体的なプログラム方法を説明する前にクラス分けの概念を説明します。プログラムの流れから言えばこの章はわき道にそれます。しかし文字判定の概念を理解する事は困難な文字認識に直面した場合に役に立つでしょう。もし手順だけを知りたい方はこの章は読み飛ばして下さい。

クラス分けの概念を理解するためには以下のプロセスを理解する必要があります。

① 文字の画像と領域の取り込み
② 特徴量の性質
③ クラス分けと信頼値(Confidence)

① 文字の画像と領域の取り込み

クラス分けを行う時、必ず画像と分割された文字領域が学習オペレータに渡されます。画像は通常大きな 1 画像で渡されますが内部では、文字の領域に 1pixel のマージンを追加した部分画像がその文字の画像として扱われます。



② 特徴量の性質

特徴量は全部で 26 種類あります。これらの特徴量の物理的な概念については create_ocr_class_box のオンラインヘルプに記述されています。これとは別に解析的な立場から以下の 3 つの性質を定義することができます。

A) グレイ値判定と領域判定
B) 正規化(zooming)に対して可変と不変
C) 情報量の大きさ

特徴量にはグレイ値（画像 pixel）判定と領域判定があります。グレイ値判定は pixel レベルで判定するために情報量が多く一般に領域判定より精度の向上が期待できます。グレイ値判定は正規化された画像に対して行われます。正規化は create_ocr_class_box の WidthPattern と HeightPattern で指定します。それぞれの特徴量で情報量が異なると言う事の認識が重要です。以下に特徴量の特性を表で示します。

特徴量	Gray ， Region	Zooming	情報量
ratio	Region	NoZoom	1
width	Region	NoZoom	1
height	Region	NoZoom	1
foreground	Region	NoZoom	1
foreground_grid_9	Region	NoZoom	9
foreground_grid_16	Region	NoZoom	16
anisometry	Region	NoZoom	1
compactness	Region	NoZoom	1
convexity	Region	NoZoom	1
moments_region_2nd_invar	Region	NoZoom	3
moments_region_2nd_rel_invar	Region	NoZoom	2
moments_region_3rd_invar	Region	NoZoom	4

moments_central	Region	NoZoom	4
phi	Region	NoZoom	1
num_connect	Region	NoZoom	1
num_holes	Region	NoZoom	1
projection_horizontal	Gray	Zoom	Height
projection_vertical	Gray	Zoom	Width
projection_horizontal_invar	Gray	Zoom	Height
projection_vertical_invar	Gray	Zoom	Width
chord_histo	Region	Zoom	Height
num_runs	Region	NoZoom	1
pixel	Gray	Zoom	Width×Height
pixel_invar	Gray	Zoom	Width×Height
cooc	Gray	NoZoom	12
moments_gray_plane	Gray	NoZoom	4

＊Width と Height は create_ocr_class_box で指定する正規化の WidthPattern と HeightPattern です。

③　クラス分けと信頼値(Confidence)

クラス分けとは 1 つの文字を多次元ベクトルで表現し、それを全ての文字に対して行う事です。これを図で表現すると多次元ベクトルが超空間で表され図示する事は難しくなります。ベクトル次元は create_ocr_class_box で指定する特徴量で決定されます。ベクトル次元は指定した特長量の数ではなく、指定した特徴量の情報量の合計で決定される事に注意が必要です。クラス分けの理論式を例で説明します。

- クラス分け

  クラス分けの特徴量と正規化を create_ocr_class_box で指定します。例として'height'と'pixel'の特徴量を、正規化として高さ 10、幅 8 を指定します。

  create_ocr_class_box(10, 8, … ['height', 'pixel'] …)

  さて特徴量'height'と'pixel'を指定しましたが、このベクトル次元(自由度)は幾つになるのでしょうか。'height'が 1 で'pixel'が 10×8=80 の情報量を持つのでベクトル次元は 81 です。2 でないことに注意して下さい。つまり'height'と'pixel'の 1pixel の情報量は等価です。グレイ値判定が如何に大きい情報量で文字を判定するかを理解できます。

  クラス分けを数式で説明します。ある 1 文字'a'のクラス分けを考えます。この場合文字'a'は 81 次元ベクトルで表現され式 1 で表現されます。

$$\begin{pmatrix} a_1^1 \\ \cdot \\ \cdot \\ a_n^1 \end{pmatrix} \qquad \text{式 1}$$

n はベクトル次元を示しここでは 81 まであります。今回は文字データを(1 枚 20 文字)×(1 フォント 20 枚) = 400 文字取得しました。従って式 1 は 400 文字通りの組み合わせがあり式 2 で表現されます。

$$\begin{pmatrix} a_1^1 \\ \cdot \\ \cdot \\ a_n^1 \end{pmatrix} \cdots \begin{pmatrix} a_1^m \\ \cdot \\ \cdot \\ a_n^m \end{pmatrix} \qquad \text{式 2}$$

ここで m は文字数を示します。今回は m=400 です。対応するベクトルの平均を計算します。平均ベクトルは

式 3 で表現されます。

$$\begin{pmatrix} \overline{a}_1 \\ \cdot \\ \cdot \\ \overline{a}_n \end{pmatrix}$$ 式 3

対応するベクトルのばらつき程度(分散)を計算します。分散ベクトルは式 4 で表現されます。

$$\begin{pmatrix} \hat{a}_1 \\ \cdot \\ \cdot \\ \hat{a}_n \end{pmatrix}$$ 式 4

これである 1 文字'a'のクラス分けが終了しました。この計算を全ての文字に対して行います。クラス分けとは平均ベクトルと分散ベクトルを全ての文字に対して求める事です。

- 信頼値(Confidence)

文字の判定は信頼値(Confidence)で判断されます。信頼値の計算方法について説明します。判断される1文字を'c'と仮定します。1文字'c'は式 1 と同様に式 5 で表現されます。

$$\begin{pmatrix} c_1 \\ \cdot \\ \cdot \\ c_n \end{pmatrix}$$ 式 5

式 5 とクラス分けで得られた式 3 の平均ベクトルと式 4 の分散ベクトルの間で式 6 の計算を行います。

$$\begin{pmatrix} \left|c_1 - \overline{a}_1\right| / \hat{a}_1 \\ \cdot \\ \cdot \\ \left|c_n - \overline{a}_n\right| / \hat{a}_n \end{pmatrix}$$ 式 6

式 6 の計算結果から式 7 を計算し信頼値を求めます。

$$Confidence = 1 - \frac{\sum \begin{pmatrix} \left|c_1 - \overline{a}_1\right| / \hat{a}_1 \\ \cdot \\ \cdot \\ \left|c_n - \overline{a}_n\right| / \hat{a}_n \end{pmatrix}}{n}$$ 式 7

式 6 ではそれぞれのベクトルを分散で割っています。これは信頼値の計算に特徴量データのばらつきを考慮しています。もしある特徴量が大きくばらついていた場合、すなわち信頼性が低い場合にはそのデータの影響は小さくなります。式 7 から信頼値は常に 1 以下の値になります。

これを全てのクラス分けされた文字に対して計算しそれぞれの信頼値を求めます。最終的に最も信頼値の高い文字が文字'c'の判定文字と判断されます。

6. 特徴量の選択

さて話をプログラムに戻します。文字の判定率を上げるにはそれらの文字を良く区別する特徴量で文字をクラス分けする必要があります。HALCON は実に 26 個の特徴量がそれらの組み合わせも自由に選択する事ができます。一見して非常に柔軟で高精度な特徴量を指定できるように思えますが、この自由度がプログラム設計者を悩ませます。いったい何を基準に特徴量を選択したらよいのでしょうか。認識する文字を見ただけでは何が良い特徴量なのか判断できません。多くの場合は設計者の勘と試行錯誤に頼る部分が多いのが実情でした。もっとシステマティックに数値で判断できないものでしょうか。ではその解法を以下に示します。

① それぞれ 1 つの特徴量だけでクラス分けを行い、そのエラー文字数を数えます。
② 判定率の高い特徴量を組み合わせます。

実にシンプルな方法で確実な方法です。①は 1 つの特徴量だけでクラス分けを行います。決してこの段階で複数の特徴量を使用しない事が重要です。次にそのクラス分けで全ての文字の判定を行い、それぞれの特徴量でエラー数を比較します。これによりどの特徴量が有効か判断できるでしょう。最後に良い結果の得られた特徴量を組み合わせ、最終的な特徴量を決定します。
では有効な特徴量を選択するために使用するオペレータについて流れに沿って説明します。まず create_ocr_class_box で正規化量と特徴量を指定し OCR のハンドルを取得します。

create_ocr_class_box (: : WidthPattern, HeightPattern, Interpolation,
Features, Character : OcrHandle)

WidthPattern	ピクセルレベルの特徴量に対して正規化する高さを指定します。
HeightPattern	ピクセルレベルの特徴量に対して正規化する幅を指定します。
Interpolation	正規化する補間モードを指定します。
Features	特徴量を指定します。
Character	全ての教示データを指定します。
OcrHandle	OCR のハンドルを指定します。

このオペレータはクラス分けの定義を行うだけです。これ以降で実際のクラス分けの学習が行われます。正規化について説明します。特徴量の幾つかは正規化されたグレイ値(画像)で計算を行います。取得した文字の大きさは異なるために文字をある大きさに正規化(zooming)します。一般に正規化を行うと情報量は減少方向へ向かいます。正規化の高さと幅の pixel サイズを WidthPattern と HeightPattern で指定します。標準で 10 と 8 が使われます。取り込む画像の大きさは最低この数値より大きいサイズである必要があります。もし指定した値よりも文字のピクセルサイズが小さい場合には正規化により画像が引き伸ばされ情報として意味が無くなりなす。ここで指定する正規化のサイズの 2 倍から 3 倍の解像度で文字を取得して下さい。この値は何を基準に決定するのでしょうか。これは認識する対象の細かさで決定されるべきです。例えば今回のような英数字だけのように文字が複雑でない場合には標準の 10 と 8 で良いでしょう。しかし非常に細かい漢字やマークを認識する場合にはこの値を大きくしまた撮影する解像度も大きくする必要があるでしょう。一つの目安として文字の凹凸数が参考になります。例えば縦方向に凹凸が 15 あるような難しい漢字を高さ 10 の正規化では認識できません。最低で 15、実際にはそれ以上の正規化と解像度が必要です。正規化の値を大きくした場合判定精度が向上するかもしれませんが学習と判定には時間がかかります。

次に trainf_ocr_class_box でクラス分けを行います。
trainf_ocr_class_box (: : OcrHandle, FileName : AvgConfidence)

OcrHandle	OCR のハンドルを指定します。
FileName	文字ファイルを指定します。
AvgConfidence	平均信頼値が返されます。

このオペレータは文字ファイルを読み込んでクラス分けを行いその平均信頼度を返します。
create_ocr_class_box の Character で指定した教示データと文字ファイルを作成する時に指定した教示データが対応していなければいけません。またこの組み合わせを変更することでアルファベットだけまたは数値だけのク

ラス分けデータを作成する事もできます。文字ファイルを渡す方法として 3 通り考えられます。

① tuple でまとめて渡す方法
② 1 つづつループでまわして渡す方法
③ concat_ocr_trainf で文字ファイルを結合から渡す方法

どれでもクラス分けの結果は同じです。①の方法はファイル名の tuple を作成しそれを FileName に渡します。この場合にはオペレータ内部的にまとめてクラス分けを行う為に、応答が返るまでに時間がかかる事があります。②の方法は 1 つづつ文字情報をクラス分けデータに追加編集します。③の方法はファイルレベルで文字ファイルをまとめてからクラス分けを同じように行います。

次に do_ocr_single で文字の判定を行います。

do_ocr_single (Character, Image : : OcrHandle : Classes, Confidences)

Character	判定する文字の領域を指定します。
Image	判定する文字の領域が含まれる画像を指定します。
OcrHandle	OCR のハンドルを指定します。
Classes	判定結果の第 1 文字と第 2 文字が返されます。
Confidences	判定結果の第 1 文字と第 2 文字の信頼値が返されます。

1 つだけの特徴量で学習したクラス分けを用いて文字用紙 1 枚の全部の文字を判定しエラーの数を数えます。エラーは第一候補の文字と正しい文字を比較して判断します。用紙はそれぞれのフォントで 20 枚ありますが適当な 1 枚について行うだけで良いでしょう。

以上の処理を全ての特徴量 26 個について行い、それぞれのエラー数を比較します。

- エラー数と平均信頼値の違い

trainf_ocr_class_box で平均信頼値が返されます。この平均信頼度が大きい方ほどクラス分けが良好に行われた事を意味しています。しかし今回良好な特徴量を決定する為に実際に文字判定を行いそのエラー数で決定しました。平均信頼値とエラー数は一致しません。最終的な特徴量の決定はエラー数を基準に決定されるべきです。平均信頼値は全ての文字の信頼値が平均されています。よって局所的なエラーの評価にはなりません。それに対してエラー数での判断は実際の検査でも同程度のエラー頻度を期待できます。

- エラー文字の確認

判定エラーが発生した場合にはそれがどの文字で発生したか認識しておく必要があります。また必要に応じて判定エラーの発生した時の第 2 候補にも注目して下さい。今回の場合では圧倒的に小文字の l と大文字の I がエラーを引き起こしました。またその第 2 候補はそれぞれがお互いを指し示していました。このような情報はエラーが発生した場合の修正として利用できます。

以上の実験結果を以下に示します。

	arial	time
ratio	1006	1139
width	1155	1127
height	1220	1220
foreground	1139	1181
foreground_grid_9	34	47
foreground_grid_16	27	26
anisometry	965	994
compactness	1058	1158
convexity	1152	1177
moments_region_2nd_invar	440	466
moments_region_2nd_rel_invar	812	825
moments_region_3rd_invar	235	265
moments_central	751	669
phi	1137	1144
num_connect	1200	1200

num_holes	1180	1180
projection_horizontal	269	391
projection_vertical	482	318
projection_horizontal_invar	100	109
projection_vertical_invar	136	106
chord_histo	344	307
num_runs	1220	1220
pixel	81	66
pixel_invar	28	10
cooc	348	445
moments_gray_plane	689	568

以上の結果から良好な特徴量として上位 5 つの
　foreground_grid_9,　foreground_grid_16,　projection_horizontal_invar,　pixel,　pixel_invar
を最終的な特徴量として選択しました。必ずしも 5 つが良いというわけではありません。もっと極端に
　foreground_grid_16,　pixel_invar
だけの選択が良い結果であるかもしれません。ここではどちらが最適化という問題については議論しません。大切なのはこのようにしてここの特徴量を検査しそれを組み合わせるという事です。上記の 2 つの組み合わせのどちらが最適化については同じようにエラー数を数える事で検討できます。

7.　最終的なクラス分け

さて、望ましい特徴量がシステマティックに計算できました。次はこの複数の特徴量でクラスファイルを作成します。クラスファイル作成の方法は 6 章と同様です。

```
create_ocr_class_box(10,8, … ['foreground_grid_9','foreground_grid_16',
                                        'projection_horizontal_invar','pixel','pixel_invar']  … )
trainf_ocr_class_box ( … [AllCharacterFile] … )
```

trainf_ocr_class_box で指定する文字ファイルを変える事で arial だけ、time だけなどの学習ファイルを作成する事ができます。そして最後にクラス分けしたファイルを write_ocr で保存します。

write_ocr (: : OcrHandle, FileName :)

OcrHandle	OCR のハンドルを指定します。
FileName	保存するファイル名を指定します。

ここで保存されるファイルはクラス分けの学習情報を含んでいます。

8.　文字認識

さてこれで最終的なクラス分けが終了しました。標準サンプルプログラムの letter.dev のようにマウスをクリックした文字を認識してみて下さい。

9.　更に精度を向上させるために

今回はかなり良い条件で学習を行いました。それでもまだなお 100%の判定率には至っていません。小文字の l と大文字の I を除けば 97%程度の認識率です。ではもう残りの 3%はどのように除去するのでしょうか。幾つかの提案を以下に示します。

①　文字の場所によって複数の学習ファイルを用意する

例えばはがきを思い浮かべて下さい。郵便番号には数字しかありません。このように文字の位置情報で既にその種類が限定される場合にはその教示データだけをもつ学習ファイルを使用します。

②　複数の文字を 1 文字として学習する

例えば車のナンバープレートを思い浮かべて下さい。地名のように文字がつながってある一つの意味をなす場合にはその文字を 1 つの文字として認識します。

③　データベースを利用する

英語では文章の初めは大文字で以降は小文字が使用されます。もしエラーが発生した場合その位置にふさわしい形式の文字を選択します。今回のlとIはこの方法で区別できるかもしれません。幸いにしてエラー時の第 2 候補はお互いを示しています。また住所のように地名が漢字で表され最後に番地が数字で表されるような場合にも文字列の順番の情報が使用できます。

以上は条件文で文字判定を制御する事を意味しています。