Visual Basic による文字認識ソフトの作成

２年Ａ組　上田　樹
指導教諭　米田　隆恒

１．要約

私は看板などに読めない漢字を見つけたとき、その漢字の読みがわからないと意味を調べることも困難であると感じた。携帯などについている文字認識ソフトでは、文字の大きさや行の間隔が決められていてうまく認識することができない。そこで、写真に撮った文字を認識するソフトを、Visual Basic を用いて作成することにした。

キーワード　文字認識、Visual Basic、ビットマップ、ピクセル

２．研究の背景と目的

看板などを見て、読めない漢字や知らない言語の文字などを見つけたとき、そのたびに意味を調べるのは大変である。そこで、それらをすべて写真に撮るだけで文字を認識できればよいと感じた。写真に撮った文字を認識する場合、文字が写真のどの位置にどんな大きさであっても認識できるようにしなければならない。そこで、まず、文字の位置、大きさ、文字の判定を行うソフトを作成することにした。

３．研究内容

（１）文字の位置を特定する

携帯電話などについている文字認識ソフトは、図１のように読み取る文字の位置や大きさ、行の間隔が決められており、文字の行間がそろっていない手書き文字やすでにとった写真から認識することはできない。そこで、図２のように文字が写真のどこにあっても認識できるように、文字の位置を調べるプログラムを作成した。

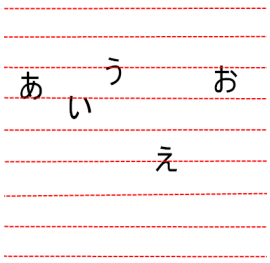

図１

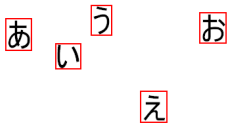

図２

文字の位置を特定するためには、上下左右の端の座標を取得する必要がある。

まず、文字の左端を特定するために、図３のように画像の左端から １ ピクセルずつ右にずらして見ていき、一番初めに見つかった、黒が含まれている列の x 座標を「左端」として保存する。同様にして、上の端も特定することができる。

次に、文字の右端を特定するために、図４のように先ほど特定した左端から １ ピクセルずつ右にずらして見ていき、一番初めに見つかった、黒が全く含まれていない列の x 座標を「右端」として保存する。同様に、下の端も特定することができる。

図３　文字の左端の特定方法

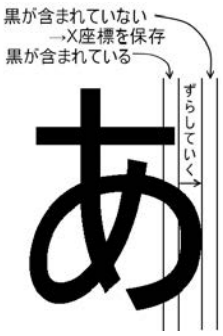

図４　文字の右端の特定方法

これにより、文字を四角く囲うことができる。しかし、このままでは、「い」などのように全体が繋がっていない文字を正確に囲うことができない。そこで、四角に囲ったとき、図５のように文字の幅が高さの半分以下なら、図６のようにもう一度左端を求める処理を行い、さらに右端を求める処理を行い、そこを右端とすることで、全体の繋がっていない文字も正確に囲うことができた。また、縦でも同じようにすることで、「う」のような縦に繋がっていない文字も読み取ることができた。

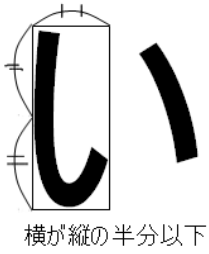

図５

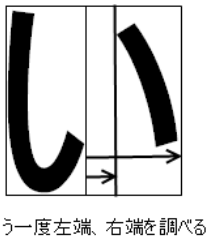

図６

次は、２文字目以降の読み取りである。２文字目以降は、左端の特定時に図７のように前の文字の右端からずらしていく。これを画像の右端まで繰り返すと、右端までの１行分の文字を切り出すことができる。

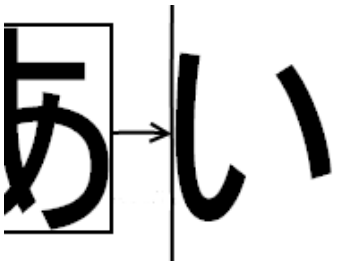

図７　２文字目の読み取り

さらに、２行目以降の読み取りが必要である。図８のように一つ上の行の下の端から２行目の上の端を探す場合、文字が揃っていれば可能だが、図９のように行間が開いていないと認識できない。そこで、上の端の特定方法は変更せず、文字の特定時にすでに切り出したものと重なっていなければ囲うという方法を使用した。

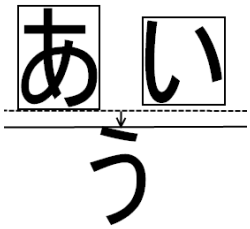

図８

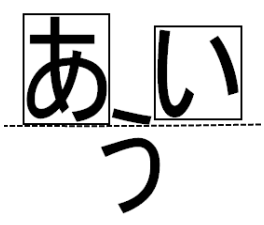

図９

ここまでの処理で、文字の位置、大きさが揃っていない場合でも文字を切り出すことができるようになった。

（２）画像を２値化する

ここまでの処理では、白と黒の文字は読み取れるが、色つきの文字は読み取ることができない。そこで、２値化(画像を白と黒の２色だけにする)処理を行った。

しかしこれだけでは、背景が紺など黒に近い色の場合、２値化時に背景も黒になってしまい、文字が認識できなくなってしまう。そこで、最初に画像全領域の RGB 値の平均値を求め、全体の平均より黒に近いピクセルを黒、それ以外を白とした。

また、画像の四隅のピクセルは字がないので、四隅の色の平均が全体の色の平均より濃い(黒に近い)場合、白黒反転処理を行うようにした。これにより、黒地に白の文字の場合も読み取れるようになった。

（３）文字を判定する

＜方法１＞　重ねる判定方法

この方法は、最初にパソコン内のフォントから判定用データベースを作る。画像から切り出した文字の画像を、先程作った判定用データベースと重ね、重なったピクセル数で判定するという方法である。

切り出した画像の大きさを判定用データに合わせ、また元の大きさが大きい場合、大きな文字の判定用データベースを別で作成し、文字の大きさがある程度大きい場合には大きい文字の判定用データベースに重ねて判定するようにしたところ、50%程度の精度で判定できるようになった。

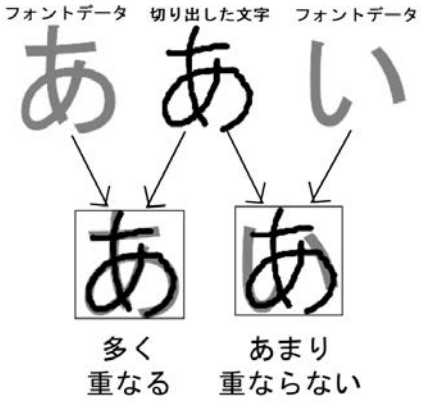

図 10　重ねる判定方法

＜方法２＞　白黒パターンでの判定方法

方法１では、処理にかなり時間がかかった。また、重ねているだけなので、この方法でこれ以上精度を上げるのは困難であると感じた。そこで、新たな方法を考案した。

図 11 のように白を 0、黒を 1 として、白と黒の入れ替わりが何回あるかを記録し、縦横それぞれで判定用データベースと比べ、最も近いものをその文字の読みとした。この方法では、かなりの速度で判定することができた。

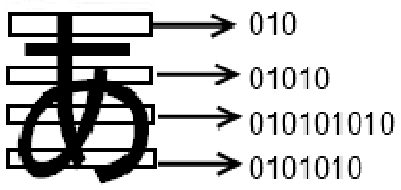

図 11　白黒パターンの作成

しかし、精度がかなり低く、原因を調べたところ、字の線の端が荒れており、結果に影響が出ているとわかった。そこで、切り出した画像を縮小し、線の荒れを小さくした。これにより、ある程度精度は上がったが、細かい部分がつぶれてしまい、読み

取れないことがある。そこで、線の荒れのみを消せる縮小の割合を判定する方法と線の荒れを修正できる方法を共に現在考案中である。

（４）背景を読み取る

写真などの場合、背景は単色ではない。影などもつき、単に背景の平均色を求めるだけではうまく判定できない。そこで、画像内の各ピクセルの濃さ(0～255, 0 に近いほど黒に近い)を調べ、色の濃さとその濃さのピクセル数をグラフにした。

そして、文字の色、背景の色はそれぞれ色の濃さが片寄っていると考え、その結果、図 12 のようにピークが得られた。これらのピークは、文字もしくは背景の主となっている色であると考え、その部分だけで平均色を求めた。これにより、ある程度写真からも読み取れるようになった。

また、このグラフのピークが文字と背景の２つ以外にあり、さらにその幅が狭い場合は、背景がグラデーションであると判断し、その幅の狭い複数個のピークの両端以外を平均色の判定には影響しないようにすることで読み込めるようにした。

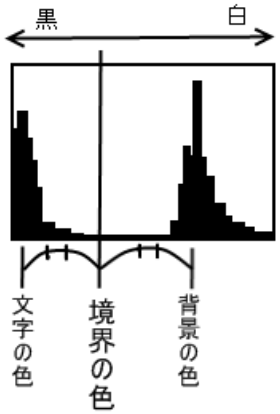

図 12　色の濃度の分布

（５）文字データの保存による高速化

フォントから方法２で作成した判定用データベースは、起動するたびに生成するため、起動に時間がかかってしまう。また、判定用データ側にバグがあった場合、判定用データベースはプログラムにより自動で生成しているため、手動での修正が不可能である。そこで、文字データを編集可能な状態で保存できるようにした。

判定用のデータは可変長配列に入れているため、保存形式は可変長配列を保存できる XML 形式を使用することにした。起動時に生成ではなく、読み込みにしたことで、起動が約半分の時間になった。

また、XML への保存にすることで、生成にかかる時間を考慮せずにすみ、判定用データの数を増やして精度を上げるということができるようになった。

４．今後の課題

今回の研究では、文字認識の大体の基礎ができた。だが、精度が低く、切り出す処理もグラデーションの背景や影付きの写真では読み込みに失敗する場合が多い。そこで、今後は背景のグラデーションや影、ノイズにも対応し、精度も判定用データをさらに増やすことで上げ、それでもできるだけ早く処理できるようにしようと思う。

そして、最終的な目標である翻訳機能も付けたいと考えている。

５．参考文献

[1]「Visual Basic 中学校」
http://homepage1.nifty.com/rucio/main/main.htm
[2]「DOBON.NET」

http://dobon.net/index.html
[3]「C#と VB.NET の入門サイト」
http://jeanne.wankuma.com/

６．謝辞

今回の研究にあたり指導してくださった顧問の米田先生、ありがとうございました。また、サイエンス研究会の先輩方にもご指導、ご協力していただきました。ありがとうございました。