>> 100行で書く画像処理最先端

>> 画像の特徴抽出：

>> Histogram of Oriented Gradients (HoG) |

>> 正会員 山崎俊彦[†]

キーワード：Histogram of Oriented Gradients (HoG)

1. ま え が き

近年、Histogram of Oriented Gradients (HoG)¹⁾やScale-Invariant Feature Transform (SIFT)²⁾³⁾といった、画像の輝度勾配をヒストグラム化する画像特徴抽出法が流行しています。元々HoGは画像からの人物領域抽出に、SIFTは画像からの特徴点抽出と画像同士のマッチングのために開発された技術でした。しかし、特徴抽出の汎用性と性能の高さからPyramid HoG⁴⁾やPCA-SIFT⁵⁾、Bag of Features⁶⁾など、さまざまな拡張がなされました。応用範囲は広く、一般物体認識、シーン分類、物体追跡、画像合成、3次元処理など画像・映像処理のありとあらゆる分野に利用されています。また、HoGは第2回で紹介したSeam Carvingの画像エネルギー算出方法の一つとしても利用されています。画像処理系のどの学会にいてもいまやHoG、SIFTという言葉を開かない日はないといっても過言ではないでしょう。

今回は、HoGを実装し、Support Vector Machine (SVM)を用いた人物検出を行ってみたいと思います。HoG、SIFTについては、すでに中部大学の藤吉先生がすばらしい解説文を多数執筆されています⁷⁾。興味があればそれらを参照して、理解を深めていただければと思います。

2. Histogram of Oriented Gradients (HoG) の実装

2.1 アルゴリズム

画像の中から人物領域を検出する処理は、服の色や体の向き、姿勢などさまざまな不確定要素があるため、解決が難しい問題の一つです。HoGは、輝度勾配のヒストグラムを用いて形状の特徴ベクトルを表現するアルゴリズムで、これにより効率的に人物の有無の判定や物体認識などを可能にします。

文献1)を読むと、画像フィルタリングや色正規化の有無、

[†] 東京大学 大学院情報理工学系研究科 電子情報学専攻
"Histogram of Oriented Gradients (HoG)" by Toshihiko Yamasaki
(Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo)

画像勾配の計算方法、角度の分割方法、1セルあたりの画素サイズや1ブロックあたりのセル数、ブロック形状の取り方、ベクトルの正規化の方法に至るまでさまざまな検討がなされていることがわかります。もちろん、対象とする画像の大きさや、判別対象物体の種類によって最適値は変わりますが、ここでは文献1)で性能が高い、とされているパラメータを使うこととします。

入力画像は64×128のグレイスケール画像とします。まずはすべての画素において、縦・横方向の勾配強度と勾配方向を下記の式によって算出します。

$$f_u(u, v) = I(u+1, v) - I(u-1, v)$$

$$f_v(u, v) = I(u, v+1) - I(u, v-1)$$

$$m(u, v) = \sqrt{f_u^2 + f_v^2}$$

$$\theta(u, v) = \arctan \frac{f_v}{f_u}$$

なお、ここで I は入力画像を、 (u, v) は画素の位置を表すものとします。

次に、6×6画素大の、互いに重ならないように敷き詰めた「セル」を図1に示すように考えます。得られた方向0°～180°を9方向(20°ずつ)に分割します。それぞれのセル内に

図1 HoGの計算の流れ

において、各画素の勾配方向の対応するビンにその画素の勾配強度 m の値を加算することで、9次元の特徴ベクトルを生成します。

さらに 3×3 の大きさのセルを一つの「ブロック」とし、1セルずつずらしながら多数のブロックを形成していきます。セルのときとは違い、ブロックは互いに重なっていることに注意が必要です。それぞれのブロックにおいて、ブロック内の特徴ベクトルを連結して81次元($9 \times 3 \times 3$)の特徴ベクトルとし、さらにベクトルの大きさが1となるように正規化します。

最後に、正規化後の81次元のベクトルをすべて連結して最終的な特徴ベクトルを得ます。64×128の画像サイズは6で割り切ることはできませんが、右端・下端の中途半端な大きさのセルも考慮に入れて計算すると縦22、横11のセルが生成できます。ブロック数は $20 \times 9 = 180$ 、最終的に得られるベクトルの次元は $81 \times 180 = 14,580$ 次元となります。

2.2 MATLABによる実装

実装例をプログラム1に示します。ここでは、後に使いやすいように画像を引数として、生成したベクトルを返す関数として実装します。

6～8行目ではセルのサイズ、回転の分割数、ブロックのサイズを決定しています。必要に応じてこの値を変更するとよいでしょう。

11～20行目では u 、 v 方向について微分をし、輝度勾配の強度と方向を計算しています。16～19行目で端の値を0にしていますが、これは画像の端ではうまく f_u 、 f_v を計算することができず、無意味な値がでるためにそれを無視するための処理です。また、20行目で0.01という値を足すのは、 $\arctan$ の計算が無限大に発散するのを防ぐためのものです。

23～33行目では、各セルの特徴ベクトルを生成しています。for文の中でceilやminなどの関数を用いています。これは画像サイズがセルサイズで割りきれなかったときに右端・下端をうまく処理するためのものです。ここでは、右端・下端の中途半端な部分についてもとにかく9次元の特徴ベクトルを生成するようにしました。

36～49行目までは正規化と最終的な14,580次元の特徴ベクトル生成を行っています。40～44行目でブロックごとの(正規化前の)特徴ベクトルを生成しています。45行目で正規化を行っています。46行目は文献1)にも記述されていますが、各要素の上限値を0.2に制限する処理です。47行目で正規化が終わったベクトルを順番にどんどん連結していきます。

なお、毎度のことではありますが、プログラムの読みやすさを優先して記述してありますので必ずしも処理速度やメモリの利用効率は最適ではありません。ご注意ください。プログラムは、コメントや空行を除けば40行足らずで実装できてしまいます。

2.3 C言語による実装

実装例をプログラム2に示します。生成したベクトルは

標準出力に書き出します。

19～22行目までは変数を宣言し、24～27行目で画像を読み込みます。

30～45行目で輝度勾配の強度と方向を計算しています。ここでは、たたみ込み関数を用いずに f_u 、 f_v を計算しています。また、OpenCVでは、画像はunsigned char形式となっているので、計算に使う前に型変換が必要です。

48～73行目で各セルのヒストグラムを生成します。MATLABではcellを使いましたが、ここでは3次元の動的配列を取得しています。あとでfree関数により正しくメモリーを解放することを忘れないようにしてください。難しそうに見えますが、そのほとんどが動的メモリーの取得とその初期化です。

76～96行目で各ブロックのベクトル生成と正規化、さらに、(順番に表示することで等価的に)ベクトルの連結を行っています。ここは素直なコーディングなので順を追って確認して下さい。

最後に、取得したメモリーをすべて解放して終わりです。読みやすさのための改行を除くと、100行程度で実装できています。

3. SVMを用いた人物検出への応用

3.1 準備

文献1)と同様に、MIT Pedestrian Database⁸⁾とSVMを用いた人物検出の実験を行います。libsvm⁹⁾というフリーソフトウェアを利用します。今回はすべてのファイルがCドライブの直下のite_100/hogというディレクトリにあるものとして話を進めます。

まずは実験用の画像を準備しましょう。文献8)から、上記のディレクトリにpedestrians128x64.tar.gzという人物画像データベースをダウンロードします。第1回でインストールしたcygwinを用いて

```
> cd /cygdrive/c/ite_100/hog
```

```
> tar zxvf pedestrians128x64.tar.gz
```

としてください。これで、pedestrians128x64というディレクトリができ、多数のppmファイルが展開されるはずですが、非人物画像は、文献10)にある標準画像サイトから9枚の人物が写っていない画像を任意に選び(4.1.05, 4.1.06, 4.1.07, 4.1.08, 4.2.01, 4.2.03, 4.2.05, 4.2.06, 4.2.07), miscという名前のディレクトリに保存します。それぞれ画像形式がppm, tiffのカラー画像となっていますが、今回はグレースケール画像を仮定しているので色空間の変換を行うと同時に画像形式も変換してしまいましょう。CygwinのGraphics→ImageMagickをインストールした後、プログラム3に記したimg2bmp_gray.plを下記の通り実行してください。

```
> perl img2bmp_gray.pl
```

それぞれのディレクトリにグレースケールのbmp画像があることを確認して下さい。

3.2 MATLABによる特徴ベクトルの抽出

プログラム4に記したgen_ped.mを走らせて下さい。このプログラムは、bmpファイルを順番に読み出し、特徴ベクトルを抽出して、pedestrian.txtというファイルにlibsvmで規定されたフォーマットにて書き出します。次に、プログラム5に記したgen_nonped.mを走らせて下さい。与えられた非人物のbmpファイルに対し、64×128のサイズに適宜切り取りながら特徴ベクトルを抽出して、non-pedestrian.txtというファイルに同じように書き出します (pedestrian.txtとデータ数をほぼ同じにするために、画像の右端・下端は利用していません)。生成されるファイルは、両者ともサイズが大きく、Windowsに付属のメモ帳などでは開くことができません。適宜お好みのテキストエディタやcygwinからインストールできるEditors→vim, Text→lessなどを用いて閲覧してください。

3.3 C言語による特徴ベクトルの抽出

C言語で作成したプログラムを利用して特徴量を抽出する場合、プログラムの呼び出し、その他は第1回と同じようにperl言語を用いて行います。

```
> perl gen_ped.pl
> perl gen_nonped.pl
```

としてください。なお、ソースコードはそれぞれプログラム6、プログラム7を参照して下さい。両者ともhog.exeが出力した特徴ベクトルを、libsvmのフォーマットに変換して、テキストファイルに順番に保存しています。また、gen_nonped.plでは画像の一部を切り取って、64×128のサイズにするためにImageMagickの機能を利用しています。

3.4 SVMによる人物検出

それでは準備が整いましたので、SVMを用いた人物検出の実験を行います。トレーニング用のファイルとテスト用のファイルを準備しましょう。pedestrian.txt, non-pedestrian.txtから、それぞれ冒頭の900行をトレーニングデータとして使用します。また、それぞれ残りの24行、36行をテストデータとして使用します。

```
> head -900 pedestrian.txt > training.txt
> head -900 non-pedestrian.txt >> training.txt
> tail -24 pedestrian.txt > test.txt
> tail -36 non-pedestrian.txt >> test.txt
```

SVMのトレーニングを行いましょう。libsvmにはさまざまなオプションがありますが、今回はすべてデフォルト値を使います。

```
> svm-train.exe training.txt
```

すると、training.txt.modelというモデルファイルができていると思います。次に、実際に判別を行ってみましょう。

```
> svm-predict.exe test.txt training.txt.model out.txt
```

すると、out.txtというファイルにそれぞれのテストデータの判別結果が+1 (人物)、もしくは-1 (非人物) のいずれかで出力されています。また、画面上には

Accuracy = 96.6667% (58/60) (classification)

と表示され、60個のテストデータのうち、実に58個で正解していることがわかります。HoGのように簡単なアルゴリズムと、デフォルトのパラメータのままのSVMで、これだけの検出率が出せるのは驚きです。

実際の利用では、例えば、監視カメラからの入力画像に対して、64×128のウィンドウをずらしながら特徴ベクトルを抽出し、そのエリアに人物が含まれるか、といった判定を行って行くことになります。

4. む す び

今回はMATLABで40行、C言語で100行程度でHoGの実装を行いました。また、SVMを用いることで、人物検出が高精度に行えることを確認しました。今回の実装では、入力画像サイズ、セルのサイズ、ブロックのサイズ、角度の解像度などの変更に対応できるようになっています。パラメータを変化させて実験してみるのも面白いかもしれません。また、libsvmについてもさまざまなオプションや最適化ツールなどがあるので、添付のドキュメントを読んでみることをおすすめします。

また、本稿で紹介したのソースファイルは、すべて文献11) からダウンロードできます。 (2010年1月5日受付)

〔文 献〕

- 1) N. Dalal and B. Triggs: "Histograms of Oriented Gradients for Human Detection", Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.886-893 (2005)
- 2) D.G. Lowe: "Object Recognition from Local Scale-invariant Features", Proc. IEEE International Conference on Computer Vision (ICCV), pp.1150-1157 (1999)
- 3) D. Lowe: "Distinctive Image Features from Scale-invariant Keypoints", Proc. International Journal of Computer Vision (IJCV), 60, 2, pp.91-110 (2004)
- 4) A. Bosch, A. Zisserman and X. Munoz: "Representing Shape with a Spatial Pyramid Kernel", Proc. International Conference on Image and Video Retrieval (CIVR), pp.401-408, 2007
- 5) Y. Ke and R. Sukthankar: "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors", Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.511-517 (2004)
- 6) J. Sivic and A. Zisserman: "Video Google: A Text Retrieval Approach to Object Matching in Videos", Proc. The International Conference on Computer Vision (ICCV), pp.1470-1477 (2003)
- 7) <http://www.vision.cs.chubu.ac.jp/SIFT/>
- 8) <http://cbcl.mit.edu/software-datasets/PedestrianData.html>
- 9) <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- 10) <http://sipi.usc.edu/database/database.cgi?volume=misc>
- 11) <http://www.hal.tu-tokyo.ac.jp/~yamasaki/ite/>

やまざき としひこ
山崎 俊彦 1999年、東京大学工学部電子工学科卒業。2004年、東京大学大学院博士課程修了。現在、東京大学大学院情報理工学系研究科電子情報学専攻准教授。主として、実写3次元映像や3次元CG、デジタル・セキュリティなどを中心とした画像・映像処理に関する研究およびVLSIの研究に従事。博士(工学)。正会員。

```

1 % HoG.m
2 % (C) Toshihiko YAMASAKI
3 % 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 % 各自の責任において使用してください。
5 function vec=HoG(im)
6 cell_size=6;
7 rot_res=9;
8 block_size=3;
9
10 % magnitude and direction
11 kernel_x=[1, 0, -1];
12 kernel_y=[1; 0; -1];
13 dx=conv2(double(im), double(kernel_x), 'same');
14 dy=conv2(double(im), double(kernel_y), 'same');
15 mag=sqrt(dx.^2+dy.^2);
16 mag(1,:)=0;
17 mag(end,:)=0;
18 mag(:,1)=0;
19 mag(:,end)=0;
20 theta=atan(dy./(dx+0.01));
21
22 % histogram generation for each cell
23 hist_array=cell(ceil(size(im,1)/cell_size), ceil(size(im,2)/cell_size));
24 for i=1:ceil(size(im,1)/cell_size)
25     for j=1:ceil(size(im,2)/cell_size)
26         hist_array{i,j}=zeros(rot_res,1);
27         for m=1+(i-1)*cell_size:min(i*cell_size, size(im,1))
28             for n=1+(j-1)*cell_size:min(j*cell_size, size(im,2))
29                 hist_array{i,j}( floor((theta(m,n)+pi/2)*rot_res/pi)+1 , 1)=hist_array{i,j}( floor((theta(m,n)+pi/2)*rot_res/pi)+1 , 1) + mag(m,n);
30             end
31         end
32     end
33 end
34
35 % normalization for each block & generate vector
36 vec=[];
37 for i=1:ceil(size(im,1)/cell_size)-(block_size-1)
38     for j=1:ceil(size(im,2)/cell_size)-(block_size-1)
39         vec_tmp=[];
40         for m=i+block_size-1
41             for n=j+block_size-1
42                 vec_tmp=[vec_tmp; hist_array{m,n}];
43             end
44         end
45         vec_tmp=vec_tmp/sqrt(sum(vec_tmp.^2)+1);
46         vec_tmp(vec_tmp>0.2)=0.2;
47         vec=[vec; vec_tmp];
48     end
49 end

```

プログラム1 HoG.m

```

1 // hog.cpp
2 // (C) Toshihiko YAMASAKI
3 // 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 // 各自の責任において使用してください。
5 #include <stdio.h>
6 #include <math.h>
7 #include <cv.h>
8 #include <highgui.h>
9
10 int minimum(int a, int b){

```

```

11     if(a<b){
12         return a;
13     }
14     return b;
15 }
16
17 // usage: hog.exe [filename]
18 int main(int argc, char** argv){
19     int x, y, i, j, k, m, n, count, cell_size=6, rot_res=9, block_size=3;
20     float dx, dy, ***hist, *vec_tmp, norm;
21     IplImage *im = NULL;
22     CvMat *mag = NULL, *theta = NULL;
23
24     if( (im = cvLoadImage( argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ){
25         fprintf(stderr, "No such file %s", argv[1]);
26         exit(0);
27     }
28
29     // magnitude and direction
30     mag = cvCreateMat(im->height, im->width, CV_32F);
31     theta = cvCreateMat(im->height, im->width, CV_32F);
32     for(y=0; y<im->height; y++){
33         for(x=0; x<im->width; x++){
34             if(x==0 || x== im->width-1 || y==0 || y==im->height-1){
35                 cvmSet(mag, y, x, 0.0);
36                 cvmSet(theta, y, x, 0.0);
37             }
38             else{
39                 dx=double((uchar)im->imageData[y*im->widthStep + x+1]) - double((uchar)im->imageData[y*
im->widthStep + x-1]);
40                 dy=double((uchar)im->imageData[(y+1)*im->widthStep + x]) - double((uchar)im->imageData
[(y-1)*im->widthStep + x]);
41                 cvmSet(mag, y, x, sqrt(dx*dx + dy*dy));
42                 cvmSet(theta, y, x, atan(dy/(dx+0.01)));
43             }
44         }
45     }
46
47     // histogram generation for each cell
48     hist = (float***)malloc(sizeof(float**) * (int)ceil((float)im->height/(float)cell_size));
49     if (hist==NULL) exit(1);
50     for(i=0; i<(int)ceil((float)im->height/(float)cell_size); i++){
51         hist[i] = (float**)malloc(sizeof(float*)*(int)ceil((float)im->width/(float)cell_size));
52         if(hist[i]==NULL) exit(1);
53         for(j=0; j<(int)ceil((float)im->width/(float)cell_size); j++){
54             hist[i][j] = (float *)malloc(sizeof(float)*rot_res);
55             if(hist[i][j]==NULL) exit(1);
56         }
57     }
58     for(i=0; i<(int)ceil((float)im->height/(float)cell_size); i++){
59         for(j=0; j<(int)ceil((float)im->width/(float)cell_size); j++){
60             for(k=0; k<rot_res; k++){
61                 hist[i][j][k]=0.0;
62             }
63         }
64     }
65     for(i=0; i<(int)ceil((float)im->height/(float)cell_size); i++){
66         for(j=0; j<(int)ceil((float)im->width/(float)cell_size); j++){
67             for(m=i*cell_size; m<minimum((i+1)*cell_size, im->height); m++){
68                 for(n=j*cell_size; n<minimum((j+1)*cell_size, im->width); n++){
69                     hist[i][j][(int)floor((cvmGet(theta,m,n)+CV_PI/2)*rot_res/CV_PI)]+=cvmGet(mag,m,n);
70                 }
71             }
72         }
73     }

```

```

74
75 // normalization for each block & generate vector
76 vec_tmp=(float *)malloc(sizeof(float)*block_size*block_size*rot_res);
77 for(i=0; i<(int)ceil((float)im->height/(float)cell_size) - (block_size-1); i++){
78     for(j=0; j<(int)ceil((float)im->width/(float)cell_size) - (block_size-1); j++){
79         count=0;
80         norm=0.0;
81         for(m=i; m<i+block_size; m++){
82             for(n=j; n<j+block_size; n++){
83                 for(k=0; k<rot_res; k++){
84                     vec_tmp[count++]=hist[m][n][k];
85                     norm+=hist[m][n][k]*hist[m][n][k];
86                 }
87             }
88         }
89         for(count=0; count<block_size*block_size*rot_res; count++){
90             vec_tmp[count]=vec_tmp[count]/(sqrt(norm+1));
91             if(vec_tmp[count]>0.2) vec_tmp[count]=0.2;
92             printf("%.4f ", vec_tmp[count]);
93         }
94     }
95 }
96 printf("\n");
97
98 for (i=0; i<(int)ceil((float)im->height/(float)cell_size); i++) {
99     for (j=0; j <(int)ceil((float)im->width/(float)cell_size); j++) {
100         free(hist[i][j]);
101     }
102     free(hist[i]);
103 }
104 free(hist);
105 free(vec_tmp);
106 cvReleaseImage(&im);
107 cvReleaseMat(&mag);
108 cvReleaseMat(&theta);
109 return 0;
110 }

```

プログラム2 : HoG.cpp

```

1 # img2bmp_gray.pl
2 # (C) Toshihiko YAMASAKI
3 # 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 # 各自の責任において使用してください。
5
6 chdir("pedestrians128x64");
7 @ppm=<*.ppm>;
8 foreach $im (@ppm){
9     print $im, "\n";
10    $bmp=$im;
11    $bmp=~s/ppm$/bmp/;
12    system("convert $im -type Grayscale -colors 256 $bmp");
13 }
14 chdir("..");
15 chdir("misc");
16 @tiff=<*.tiff>;
17 foreach $im (@tiff){
18     print $im, "\n";
19     $bmp=$im;
20     $bmp=~s/tiff$/bmp/;
21     system("convert $im -type Grayscale -colors 256 $bmp");
22 }

```

プログラム3 img2bmp_gray.pl

```

1 % gen_ped.m
2 % (C) Toshihiko YAMASAKI
3 % 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 % 各自の責任において使用してください。
5 bmp=dir('pedestrians128x64¥¥*.bmp');
6
7 fid=fopen('pedestrian.txt','w');
8 for i=1:size(bmp,1)
9     fprintf('%s¥n',bmp(i).name);
10    im=imread(sprintf('pedestrians128x64¥¥%s',bmp(i).name));
11    vec=HoG(im);
12    fprintf(fid,'1 ');
13    for element=1:size(vec,1)
14        fprintf(fid,'%d:%.4f ',element, vec(element));
15    end
16    fprintf(fid,'¥n');
17 end
18 fclose(fid);

```

プログラム4 gen_ped.m

```

1 % gen_nonped.m
2 % (C) Toshihiko YAMASAKI
3 % 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 % 各自の責任において使用してください。
5 W=64;
6 H=128;
7 bmp=dir('misc¥¥*.bmp');
8
9 fid=fopen('non-pedestrian.txt','w');
10 for i=1:size(bmp,1)
11     fprintf('%s¥n',bmp(i).name);
12     im=imread(sprintf('misc¥¥%s',bmp(i).name));
13     for j=1:32:size(im,1)-H
14         for k=1:32:size(im,2)-W
15             vec=HoG(im(j:j+H-1, k:k+W-1));
16             fprintf(fid,'-1 ');
17             for element=1:size(vec,1)
18                 fprintf(fid,'%d:%.4f ',element, vec(element));
19             end
20             fprintf(fid,'¥n');
21         end
22     end
23 end
24 fclose(fid);

```

プログラム5 gen_nonped.m

```

1 # gen_ped.pl
2 # (C) Toshihiko YAMASAKI
3 # 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 # 各自の責任において使用してください。
5 $dir="pedestrians128x64";
6 open(OUT,">pedestrian.txt");
7
8 @bmp=<$dir/*.bmp>;
9 foreach $im (@bmp){
10     $count=1;
11     print $im, "\n";
12     $result=`./hog.exe $im`;
13     $result=~s/ [^ ]*$//;
14     @array=split(/ /,$result);
15     print OUT "1 ";
16     foreach $element (@array){
17         print OUT $count++, ":$element ";
18     }
19     print OUT "\n";
20 }
21 close(OUT);

```

プログラム6 gen_ped.pl

```

1 # gen_nonped.pl
2 # (C) Toshihiko YAMASAKI
3 # 本プログラムを使用したことによるいかなる障害、損害も責任はとりません
4 # 各自の責任において使用してください。$dir="misc";
5 open(OUT,">non-pedestrian.txt");
6 $step=32;
7 $W=64;
8 $H=128;
9
10 @bmp=<$dir/*.bmp>;
11 foreach $im (@bmp){
12     $info=`identify $im`;
13     $info=~ / (¥d+)x(¥d+) /;
14     $width=$1;
15     $height=$2;
16     print "$im\n";
17
18     for($i=0;$i<$height-$H;$i=$i+$step){
19         for($j=0;$j<$width-$W;$j=$j+$step){
20             system("convert $im -crop ${W}x${H}+$j+$i tmp.bmp");
21             $result=`./hog.exe tmp.bmp`;
22             $result=~s/ [^ ]*$//;
23             @array=split(/ /,$result);
24             print OUT "-1 ";
25             $count=1;
26             foreach $element (@array){
27                 print OUT $count++, ":$element ";
28             }
29             print OUT "\n";
30         }
31     }
32 }
33 close(OUT);
34 system('rm tmp.bmp');

```

プログラム7 gen_nonped.pl