An Example-based Image Retrieval System

Qichen Jing, He Zhang

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec 13, 2013

Technical Report No. ECE-2013-12

Contents

List of Figures

List of Tables

1 Introduction

Some owners of digital cameras with poor organization of photos will probably encounter a problem of finding a specific picture. They can clearly remember the objects and contents in that picture but do not remember when it was taken or which file it is in. Sometimes people want to search images that are similar to the one he or she already have. In those scenarios we are facing the same problem of finding similar photos. Since the photoset containing the desired images has a numerous volume, exploring every picture manually is very time consuming. We expect to have a program that can automatically output the required similar images from a database based on the input example image. Therefor in our project, we aim at investigating and establishing an image retrieval system that helps users pick photos in a database that is alike the input example image.

2 Literature review

For centuries, most of the images retrieval is text-based which means searching is based on those keyword and text generated by human's creation. The text-based image retrieval systems only concern about the text described by humans, instead of looking into the content of images.[5] And there are many problems about this text-based method. First, generated text is subjective and different people may use different on the same image. Second, when generating keyword, it is easy to make some mistakes such as misspelling. Also, annotation impreciseness may cause unrecoverable mismatches.

To overcome those drawbacks, content based image retrieval was introduced. Example-based image retrieval is a kind of CBIR. Feature extraction is the basic of image retrieval. The feature used in CBIR can be divided in several categories: color, texture, shape, color layout and so on.[7][4] In our project, we use covariance matrices and compare it to commonly used descriptor HSV space color histogram.

Furthermore, some classification technique is usually used in CBIR to perform well and easy retrieval. Some people use k-means and hierarchical clustering method in retrieval problem.[5][6] What's more, some people proposed a relevance feedback technique using some machine learning methods and achieved some effect in satisfying people’s expectation. And this inspires us try to use classification way in image retrieval problem in specific case.

3 Problem statements

3.1 General case

In this case, our goal is to find the most similar image in the database to our query one. And there is no restriction on our database.

The key issue in designing this system is to represent query image and images in our database and to measure their similarity. First, a suitable image descriptor that could greatly describe the characteristics of the image should be found. Secondly, a method is needed to be designed to match the selected descriptor.

By analyzing the function of the system, we proposed four sections to be implemented. First a database contains some categories of pictures with the same size and resolution need to be established. (Establishing this database with images from some categories makes it convenient to evaluate our system) Next, we need to design an algorithm module that handles the input image example and calculates the descriptors of images in database. Thirdly, there should be a matching module comparing query image with the rest of images in the dataset. The last part is an evaluation module which calculates the correct classify rate of this system.

3.2 Specific case

In this case, our goal is to find k (k is a number of outputs) pictures in our database that are most similar to query image and expect them in the same coherent set. There is one restriction about our database: that is the database could be classified. And we transform image retrieval problem into a problem about classification first and retrieving in one category.

After analyzing the function of our system in this specific case, we proposed sections to be implemented. Firstly, a database contains images from some different categories is established. Secondly, we need to find an appropriate descriptor to represent images. Thirdly, a classifier is needed to classify our database into coherent sets. Next, we decide which is the nearest coherent set to query image. Finally, we retrieve k nearest images in that coherent set. More details will be discussed in section 3.

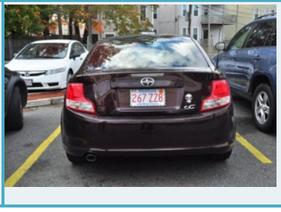
4 Implementations

4.1General case

4.1.1 Database description

Our database contains 190 different pictures (384 by 256 pixels) in total, 30 of them were taken by us and the rest are from the Internet. They are divided into 11 categories. In each category, every photo has similar main objects with others, but with different positions and sizes. Chart 1 gives the description of the database in detail.

Table1. Database1

Content	No.	Examples	Content	No.	Examples
Bus	**1-20**		**Human**	**101-120**	
Elephan t	**21-40**		**Seashore**	**121-140**	
Flower	**41-60**		**Building**	**141-160**	
Horse	**61-80**		**House**	**161-170**	
Foods	**81-100**		**Car (side view)**	**171-180**	
Car (back view)	**181-190**				

4.1.2 Flowchart

The low chart of our system is shown in Fig1.

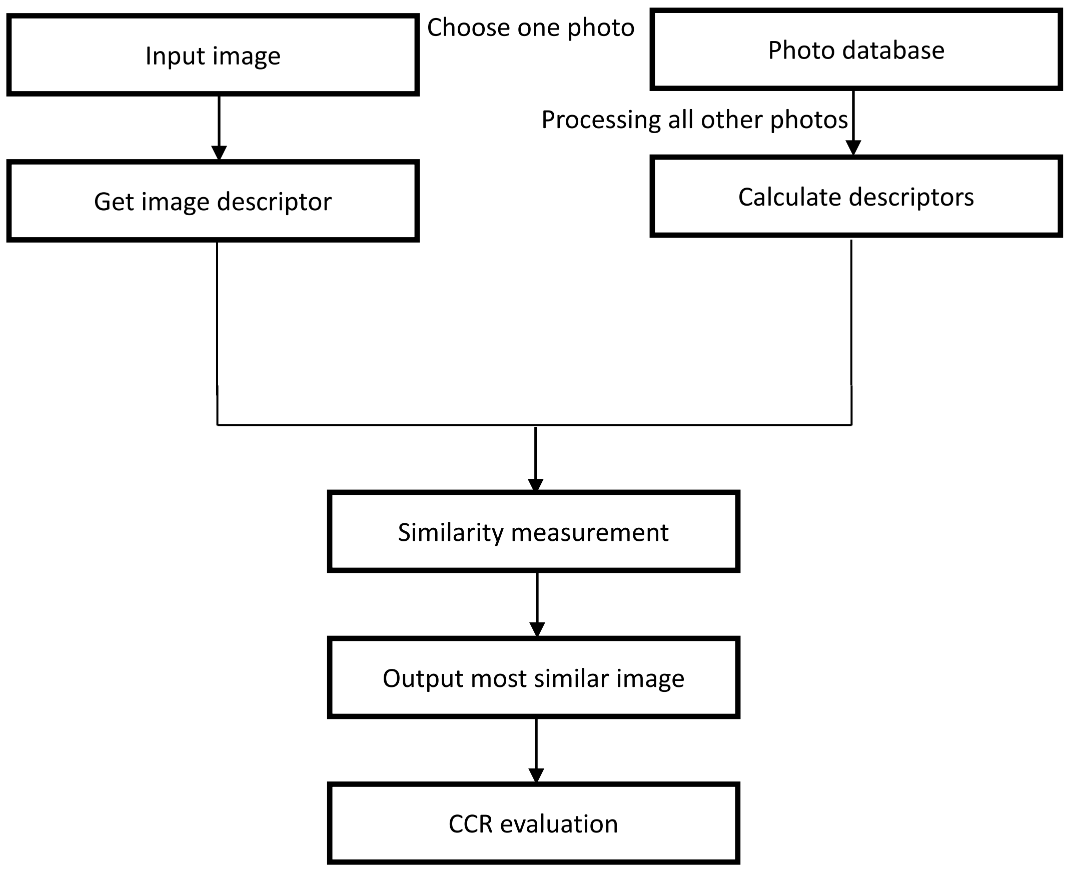


Fig1. System flowchart1

We use two descriptors here, one is overlap covariance matrices and the other is HSV color histogram. First of all, an arbitrary image is chosen from database and then calculating its descriptor. Next, calculation distances between descriptor of input image and other images left in database. Finally, using nearest neighbor classify method to find the nearest as output.

The descriptor we choose here are overlap covariance matrices and HSV color histogram.

4.1.3 Covariance matrix calculation

The feature space of pictures we choose is expressed as

$$F(x,y) = \begin{bmatrix} x,\ y,\ R(x,y),\ G(x,y),\ B(x,y), \\ \left|\frac{\partial I(x,y)}{\partial x}\right|, \left|\frac{\partial I(x,y)}{\partial y}\right|, \left|\frac{\partial^2 I(x,y)}{\partial x^2}\right|, \left|\frac{\partial^2 I(x,y)}{\partial y^2}\right|,\ atan2(\frac{\partial I(x,y)}{\partial y}, \frac{\partial I(x,y)}{\partial x}) \end{bmatrix}$$

Where x,y are the location of pixels, R(x,y), G(x,y), B(x,y) are the color values, $\frac{\partial^2 \mathrm{I(x,y)}}{\partial \mathrm{x}^2} \frac{\partial^2 \mathrm{I(x,y)}}{\partial \mathrm{y}^2}$ are the first order gradient of intensity, $\frac{\partial^2 I(x,y)}{\partial x^2} \frac{\partial^2 I(x,y)}{\partial y^2}$ are the second order gradient of Intensity, and $atan2(\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y})$ is the direction of first order gradient.

The process of calculating the covariance matrix in the region *R* is as follows:

$$\boldsymbol{C}_R = \frac{1}{n-1}\sum_{k=1}^{n}(\boldsymbol{Z}_k - \boldsymbol{\mu})(\boldsymbol{Z}_k - \boldsymbol{\mu})^T \text{ [1]}$$

Where $\{\boldsymbol{Z}_k\}_{k=1\ldots n}$ are the-dimensional feature points inside *R* and the (*i*, *j*)-th element of the covariance matrix is

$$C_R(i,j) = \frac{1}{n-1}\sum_{k=1}^{n}(Z_k(i) - \mu(i))\,(Z_k(j) - \mu(j))$$

$$= \frac{1}{n-1}\left[\sum_{k=1}^{n} Z_k(i)Z_k(j) - \frac{1}{n}\sum_{k=1}^{n} Z_k(i)\sum_{k=1}^{n} Z_k(j)\right]$$

To increase the speed to calculate the region sums, we found a fast computation method that summing up all the pixels inside the rectangle region of interest.

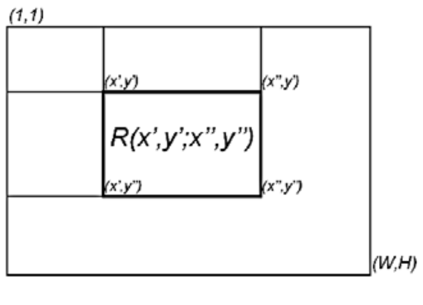


Fig2. Region sums

For an intensity image R, its integral image is defined as

Integral Image (*x'*, *y'*) = $\sum_{x<x',y<y'} R(x,y)$.

P be the *W* × *H* × *d* tensor of the integral images

$$P(x', y', i) = \sum_{x<x', y<y'} F(x, y, i)\,,\ i = 1 \ldots d$$

Q be the $W \times H \times d \times d$ tensor of the second order integral images

$$Q(x', y', i, j) = \sum_{x<x', y<y'} F(x, y, i) F(x, y, j)$$

$P_{x,y}$ is the d dimensional vector

$$P_{x,y} = [P(x, y, 1) \ldots P(x, y, d)]^T$$

$Q_{x,y}$ is the $d \times d$ dimensional matrix

$$Q_{x,y} = \begin{pmatrix} Q(x, y, 1,1) \ldots Q(x, y, 1, d) \\ \ldots \\ Q(x, y, d, 1) \ldots Q(x, y, d, d) \end{pmatrix}$$

$R(x', y', x'', y'')$ is the rectangular region shown in Fig3.

$$C_{R(x', y'; x'', y'')} = \frac{1}{n-1} \left[Q_{x'',y''} + Q_{x',y'} - Q_{x'',y'} - Q_{x',y''} \right.$$

$$\left. - \frac{1}{n} (P_{x'',y''} + P_{x',y'} - P_{x',y''} - P_{x'',y'}) (P_{x'',y''} + P_{x',y'} - P_{x',y''} - P_{x'',y'}) \right]$$

where n = $(x'' - x') \bullet (y'' - y')$.

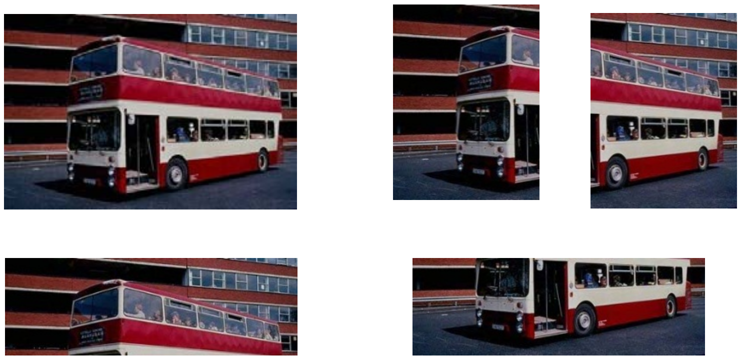
After we get the covariance matrices, we use the formula below to calculated their distance

$$\rho\left(\boldsymbol{C}_{1,}\ \boldsymbol{C}_2\right) = \sqrt{\sum_{i=1}^{n} ln^2\, \lambda_i\, (\boldsymbol{C}_{1\,,}\ \boldsymbol{C}_2)}\ [2]$$

Where $\{\lambda_i(C_1, C_2)\}_{i=1\ldots n}$ are the generalized eigenvalues of $C_{1\,,}\ C_2$ which satisfy

$\lambda_i C_1 x_i - C_2 x_i = 0 \quad i = 1 \ldots d$

4.1.4 Overlap Covariance Matrices



For each image, we use five covariance matrices to describe it. The first covariance matrix is for the whole image. And the second and the third one represent left half part and right half part. The fourth and the fifth ones describe upper and lower parts respectively.

Then we use the summation of these five distances as distance between two pictures.

$$\mathrm{Dis}(\mathrm{P1},\mathrm{P2}) = \rho\left(C_{11,}\ C_{21}\right) + \rho\left(C_{12,}\ C_{22}\right) + \rho\left(C_{13,}\ C_{23}\right) + \rho\left(C_{14,}\ C_{24}\right) + \rho\left(C_{15,}\ C_{25}\right)\ .$$

4.1.5 HSV color histogram

Color histogram is a common global descriptor to describe an image and was used in many image retrieval systems. HSV stands for the HSV color space(hue, saturation, value) which could better represent human vision than some other color spaces such as RGB.

We divided each direction into ten bins and form a three dimensional color histogram for image. And the distance measurement is Euclidian.

4.2 Specific case

4.2.1 Flow chart

cIn this case, we test a new database and try to find five nearest outputs of query image. The flow chart in this case is shown in Fig2.

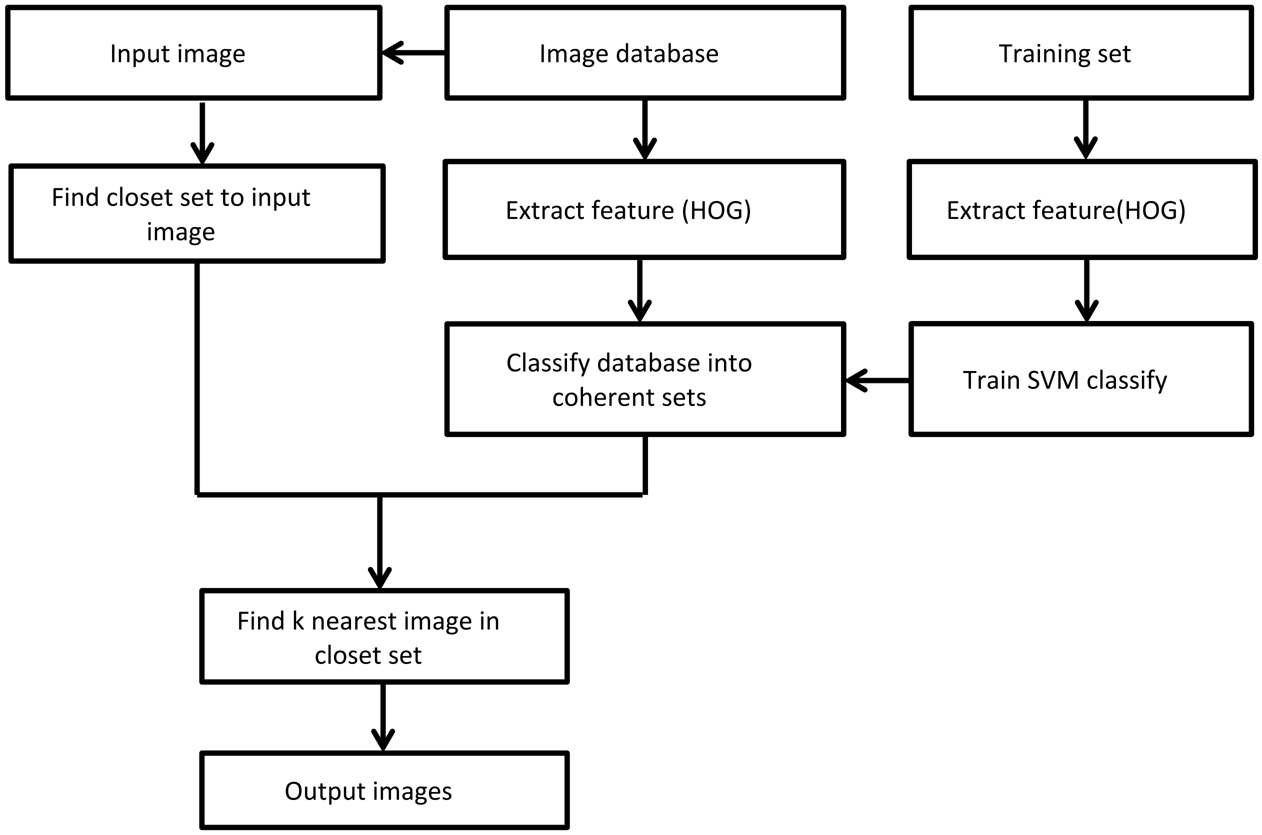


Fig2. System Flowchart2

4.2.2 Database description

The database here we form includes 100 images (360 by 240) in five categories. Examples are as follows:

Table2. Database2

Content	No.	Examples
Airplanes	**1-20**	
Motor cycles	**21-40**	
Cars	**41-60**	
Human faces	**61-80**	
Buildings	**81-100**	

Then we formed a training set consisting of 2000 images (400 in each category). And we extract HOG feature to train a SVM classify. [3][8]

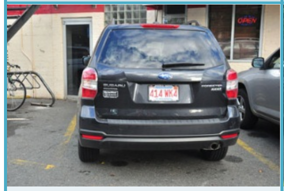
4.2.3 HOG feature

We extract the HOG features in the following steps. Firstly, calculate gradient at every pixel of picture. Then divided image into cells with size 24 by 24 and calculate histogram of gradients for each cell (9 bins in our project). Next, cascades four cells to get block descriptor. In addition, we use 50% overlap block. Finally a 4536 dimension descriptor for an image is obtained.

5 Experimental results

5.1 general case

In this case, we want to find the most similar image of a query image. Chart3 below shows examples using overlap covariance matrices.

Input images	Output images	Input images	Output images
			

To calculate the correct classify rate of our system, we proposed one strategy: if the output is in the same category with input. If they are in the same class, return “correct”. If not, return “false”.

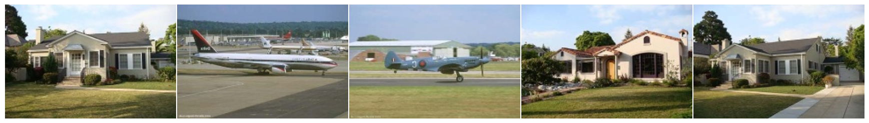
After testing 190 images in our database, we achieve CCR: 88.947(169 correct, 21 false), much higher than HSV color histogram (75.673%).

5.2 specific case

In this case, we want to get 5 most likely images in coherent sets. We test this using covariance matrices method and SVM method. Following is an example:

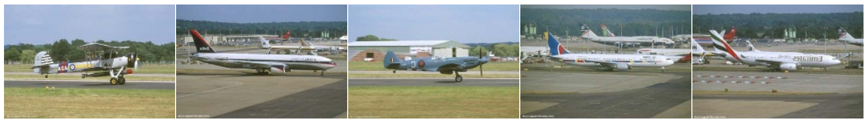
Covariance matrices method

Input image:

Five outputs: 

SVM method

Input images:

Five outputs: 

Here we use a new strategy to calculate CCR:

If all outputs are in the same category with input, it returns “correct”. If not, “false”.

CCR of covariance method: less than 93%.

CCR of SVM method: 97%.

6 Conclusions

Covariance matrix descriptor works greatly for image retrieval, we do not need to know anything about database. But it works not so well for K closest output situations (not coherent images)

The second method achieves better about K closest output problem in one specific case (if we know how many categories in photo database).

7 Future work

Our future work mainly focus on two aspects.

(1) To find a more robust descriptor or to try other features used in covariance matrix

(2) Add a feedback module in our system. For different people may expect different output images, they can choose what images are they want in the outputs of first retrieval and what are not. Then we could let our system to learn customer’s expectation and returns another outputs. He can choose again and retrieval again until he is satisfied with outputs.

References

[1] Oncel Tuzel, Fatih Porikli,and Peter Meer1,“*Region Covariance: A Fast Descriptor fo r Detection and Classification*,” Computer Science Department, Electrical and Computer Engineering Department, Rutgers University, Mitsubishi Electric Research Laboratories, Cambridge, 2006

[2] Förstner W, Moonen B. “*A metric for covariance matrices*”. Quo vadis geodesia, 1999: 113-128.

[3] Dalal N, Triggs B. “*Histograms of oriented gradients for human detection*,” Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005, 1: 886-893.

[4]Lin C H, Chen R T, Chan Y K. “*A smart content-based image retrieval system based on color and texture feature,*” Image and Vision Computing, 2009, 27(6): 658-665.

[5] Ho J M, Lin S Y, Fann C W, et al. “*A novel content based image retrieval system using K-means with feature extraction*” Systems and Informatics (ICSAI), 2012 International Conference on. IEEE, 2012: 785-790.

[6] Murthy V, Vamsidhar E, Kumar J S, et al. “*Content based image retrieval using Hierarchical and K-means clustering techniques*”. International Journal of Engineering Science and Technology, 2010, 2(3): 209-212.

[7] Y.Rui, T.S.Huang, S.F.Chang,” *Image Retrieval: Current Techniques, Promising Dir ections, and Open Issues*”, Department of ECE & Beckman Institute, University of Illinoi s at Urbana-Champaign, Urbana, Illinois, 1999

[8] N. Dalal and B.Triggs, “*Histograms of Oriented Gradients for Human Detection*” IN RIA Rh.one-Alps, 655 avenue de l'Europe, Montbonnot 38334, France, 2005

[9]http://wang.ist.psu.edu/docs/related/. Some pictures in our database are downloaded

Appendix

1. covariance matrices method

1.1 find one nearest image

```
clc;
clear;
nump=190;
%% calculate to get covariance matrices of database
nf=10;
C=zeros(nf,nf,nump);
C1=zeros(nf,nf,nump);
C2=zeros(nf,nf,nump);
C3=zeros(nf,nf,nump);
C4=zeros(nf,nf,nump);
%% imread the first image to calculate image size
imname1=strcat(num2str(1),'.jpg');
imname1=strcat('photos\',imname1);
picture1=imread(imname1);
[a,b,c]=size(picture1);
%% calculate covariance matrixes for all pictures
for i=1:nump
   imname=strcat(num2str(i),'.jpg');
   imname=strcat('photos\',imname);
   picture=imread(imname);
   C(:,:,i)=covmat(picture,1,1,b,a);
   C1(:,:,i)=covmat(picture,1,1,b/2,a);
   C2(:,:,i)=covmat(picture,1,1,b,a/2);
   C3(:,:,i)=covmat(picture,b/2,1,b,a);
   C4(:,:,i)=covmat(picture,1,a/2,b,a);
end
save('database.mat','C');
save('database1.mat','C1');
save('database2.mat','C2');
save('database3.mat','C3');
save('database4.mat','C4');
%% give a picture and then find the most likely picture

B=load('database.mat');
C=B.C;
B1=load('database1.mat');
C1=B1.C1;
B2=load('database2.mat');
C2=B2.C2;
B3=load('database3.mat');
C3=B3.C3;
B4=load('database4.mat');
C4=B4.C4;
```

```
j=1; %the picture you give to find similar one
relat=zeros(nump,1);
relat1=zeros(nump,1);
relat2=zeros(nump,1);
relat3=zeros(nump,1);
relat4=zeros(nump,1);
for i=1:nump
   relat(i)=relation(C(:,:,j),C(:,:,i));
   relat1(i)=relation(C1(:,:,j),C1(:,:,i));
   relat2(i)=relation(C2(:,:,j),C2(:,:,i));
   relat3(i)=relation(C3(:,:,j),C3(:,:,i));
   relat4(i)=relation(C4(:,:,j),C4(:,:,i));

end
relat=1/2.*relat+1/2.*relat1+1/2.*relat2+1/2.*relat3+1/2.*relat4;
relat(j)=[];
[la,ind]=sort(relat);
% [min,num]=min((relat));
for k=1:nump-1
  if ind(k)<j
    ind(k)=ind(k);
  else
    ind(k)=ind(k)+1;
  end
end
num=ind(1);
imggive=strcat(num2str(j),'.jpg');
imggive=strcat('photos\',imggive);
imshow(imggive);
figure();
title('picture j');
imgfind=strcat(num2str(num),'.jpg');
imgfind=strcat('photos\',imgfind);
imshow(imgfind);
% title('picture num');

%% calculate wrong nums
B=load('database.mat');
C=B.C;
B1=load('database1.mat');
C1=B1.C1;
B2=load('database2.mat');
C2=B2.C2;
B3=load('database3.mat');
C3=B3.C3;
B4=load('database4.mat');
C4=B4.C4;
wrongnum=0;
wrongind=zeros(1,nump);
relat=zeros(nump,1);
relat1=zeros(nump,1);
relat2=zeros(nump,1);
relat3=zeros(nump,1);
relat4=zeros(nump,1);
for j=1:190
```

```
      if j<=160
       m20=floor((j-1)/20);
       m10=0;
   else if j>160
           m20=8;
           m10=floor((j-160-1)/10);
       end
   end

for i=1:nump
    relat(i)=relation(C(:,:,j),C(:,:,i));
   relat1(i)=relation(C1(:,:,j),C1(:,:,i));
   relat2(i)=relation(C2(:,:,j),C2(:,:,i));
   relat3(i)=relation(C3(:,:,j),C3(:,:,i));
   relat4(i)=relation(C4(:,:,j),C4(:,:,i));
end
relat=(0.5).*relat+(0.5).*relat1+(0.5).*relat2+(0.5).*relat3+(0.5).*rel
at4;
relat(j)=[];
[la,ind]=sort(relat);
% [min,num]=min((relat));
for k=1:nump-1
  if ind(k)<j
    ind(k)=ind(k);
  else
    ind(k)=ind(k)+1;
  end
end
num=ind(1);
if m20<8
 numdown=m20*20;
 numup=(m20+1)*20;
else if m20==8
       numdown=m20*20+m10*10;
       numup=m20*20+(m10+1)*10;
   end
end


if num>numdown&&num<=numup
   wrongnum=wrongnum;
else
   wrongnum=wrongnum+1;
   wrongind(j)=j;
end
end
error=wrongnum/nump;
correct=1-error;
wrongind=find(wrongind>0)
disp(['Correct Retrieval Rate is ',num2str(correct)]);
```

1.2 extract 10 dimensional features and calculate covariance

```
function C=covmat(picture,x1,y1,x2,y2)
[H,W,R]=size(picture);
nf=10;
F=zeros(H,W,nf);
P=F;
Q=zeros(H,W,nf,nf);
[w,h]=meshgrid((1:W),(1:H));
%%  Extract feature space of an image(x,y,R,G,B,1-2D gradient)
F(:,:,1)=h;
F(:,:,2)=w;
F(:,:,3)=picture(:,:,1);
F(:,:,4)=picture(:,:,2);
F(:,:,5)=picture(:,:,3);
picgray=rgb2gray(picture);
w1=[-1 0 1];
w2=[-1 2 -1];
F(:,:,6)=imfilter(picgray,w1,'replicate');
F(:,:,7)=imfilter(picgray,w1','replicate');
F(:,:,8)=imfilter(picgray,w2,'replicate');
F(:,:,9)=imfilter(picgray,w2','replicate');
picgray=double(picgray);
F(:,:,10)=getderi(picgray);

%% normalization
F(:,:,1)=(F(:,:,1)-min(min(F(:,:,1))))/(max(max(F(:,:,1)))-min(min(F(:,
:,1))));
F(:,:,2)=(F(:,:,2)-min(min(F(:,:,2))))/(max(max(F(:,:,2)))-min(min(F(:,
:,2))));
F(:,:,3)=(F(:,:,3)-min(min(F(:,:,3))))/(max(max(F(:,:,3)))-min(min(F(:,
:,3))));
F(:,:,4)=(F(:,:,4)-min(min(F(:,:,4))))/(max(max(F(:,:,4)))-min(min(F(:,
:,4))));
F(:,:,5)=(F(:,:,5)-min(min(F(:,:,5))))/(max(max(F(:,:,5)))-min(min(F(:,
:,5))));
F(:,:,6)=(F(:,:,6)-min(min(F(:,:,6))))/(max(max(F(:,:,6)))-min(min(F(:,
:,6))));
F(:,:,7)=(F(:,:,7)-min(min(F(:,:,7))))/(max(max(F(:,:,7)))-min(min(F(:,
:,7))));
F(:,:,8)=(F(:,:,8)-min(min(F(:,:,8))))/(max(max(F(:,:,8)))-min(min(F(:,
:,8))));
F(:,:,9)=(F(:,:,9)-min(min(F(:,:,9))))/(max(max(F(:,:,9)))-min(min(F(:,
:,9))));
F(:,:,10)=(F(:,:,10)-min(min(F(:,:,10))))/(max(max(F(:,:,10)))-min(min(
F(:,:,10))));
%%  Calculate covariance of image using Integral
P=cumsum(cumsum(F,1),2);
for i=1:nf
    for j=1:nf
        Q(:,:,i,j)=cumsum(cumsum(F(:,:,i).*F(:,:,j),1),2);
    end
end
if x1==1&&y1==1&&x2==W&&y2==H
   P1=reshape(P(H,W,:),nf,1);
   Q1=reshape(Q(H,W,:),nf,nf);
   C=1/(W*H-1)*(Q1-1/(W*H)*(P1*P1'));
else
    P1=reshape(P(y1,x1,:),nf,1);
    P2=reshape(P(y2,x2,:),nf,1);
    P3=reshape(P(y2,x1,:),nf,1);
    P4=reshape(P(y1,x2,:),nf,1);
    Q1=reshape(Q(y1,x1,:),nf,nf);
    Q2=reshape(Q(y2,x2,:),nf,nf);
    Q3=reshape(Q(y2,x1,:),nf,nf);
    Q4=reshape(Q(y1,x2,:),nf,nf);

C=1/(abs((x2-x1)*(y2-y1))-1)*((Q2+Q1-Q3-Q4)-1/abs(((x2-x1)*(y2-y1)))*(P
2+P1-P3-P4)*(P2+P1-P3-P4)');
end
```

1.3 get atan2(dy,dx)

```
function [deriction]=getderi(Image)
[gx,gy]=smoothGradient(Image);
deriction=zeros(size(Image));
deriction=atan2(gy,gx);



function [GX, GY] = smoothGradient(I)

% Create an even-length 1-D separable Derivative of Gaussian filter

% Determine filter length
sigma=sqrt(2);
filterLength = 8*ceil(sigma);
n = (filterLength - 1)/2;
x = -n:n;

% Create 1-D Gaussian Kernel
c = 1/(sqrt(2*pi)*sigma);
gaussKernel = c * exp(-(x.^2)/(2*sigma^2));

% Normalize to ensure kernel sums to one
gaussKernel = gaussKernel/sum(gaussKernel);

% Create 1-D Derivative of Gaussian Kernel
derivGaussKernel = gradient(gaussKernel);

% Normalize to ensure kernel sums to zero
negVals = derivGaussKernel < 0;
posVals = derivGaussKernel > 0;
derivGaussKernel(posVals) =
derivGaussKernel(posVals)/sum(derivGaussKernel(posVals));
derivGaussKernel(negVals) =
derivGaussKernel(negVals)/abs(sum(derivGaussKernel(negVals)));

% Compute smoothed numerical gradient of image I along x (horizontal)
% direction. GX corresponds to dG/dx, where G is the Gaussian Smoothed
% version of image I.
GX = imfilter(I, gaussKernel', 'conv', 'replicate');
GX = imfilter(GX, derivGaussKernel, 'conv', 'replicate');

% Compute smoothed numerical gradient of image I along y (vertical)
% direction. GY corresponds to dG/dy, where G is the Gaussian Smoothed
% version of image I.
GY = imfilter(I, gaussKernel, 'conv', 'replicate');
GY  = imfilter(GY, derivGaussKernel', 'conv', 'replicate');
```

1.4 measure distance

```
function relat= relation(C1,C2)
%%
[V,D]=eig(C2,C1);
lamdam=diag(D);
m=size(lamdam);
relat=sqrt(sum(log((lamdam)).^2));
```

2 svm method

2.1 This part is to extract HOG feature for training set.

```
clc;
clear;
m=20;
hog=zeros(2000,4536);
for i=1:9
   string='000';
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat(string,imname1);
 imname1=strcat('photos\',imname1);
 picture1=imread(imname1);
 picture1=rgb2gray(picture1);
 hog(i,:)=getHOGDescriptor(picture1);
end

for i=10:99
   string='00';
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat(string,imname1);
 imname1=strcat('photos\',imname1);
 picture1=imread(imname1);
 picture1=rgb2gray(picture1);
 hog(i,:)=getHOGDescriptor(picture1);
end

for i=100:999
   string='0';
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat(string,imname1);
 imname1=strcat('photos\',imname1);
 picture1=imread(imname1);
 picture1=rgb2gray(picture1);
 hog(i,:)=getHOGDescriptor(picture1);
end

for i=1001:2000
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat('photos\',imname1);
 picture1=imread(imname1);
 picture1=rgb2gray(picture1);
 hog(i,:)=getHOGDescriptor(picture1);
end

save('Hog.mat','hog');
```

2.2 this part is to train SVM classifier

```
clc;
clear;
l=load('Hog.mat');
trainingset=l.hog;
label=zeros(1,2000);
label(1:400)=1;
label(401:800)=2;
label(801:1200)=3;
label(1201:1600)=4;
label(1601:2000)=5;
label=double(label');
model=svmtrain(label, trainingset,'-s 1 -t 3 -c 2 -g 0.01');
save('mod','model');
[a,b,c]= svmpredict(label,trainingset,model,'-b 0');
```

2.3 extract HOG feature for our database

```
clc;
clear;
m=20;
hogt=zeros(100,4536);
for i=1:100
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat('testphotos\',imname1);
 picture1=imread(imname1);
 picture1=rgb2gray(picture1);
 hogt(i,:)=getHOGDescriptor(picture1);
end
```

```
save('Hogtest.mat','hogt');
```

2.4 To find 5 nearest pictures

```
clc;
clear;
L=load('lab');
B=load('database.mat');
C=B.C;
label=L.prelabel;
 i=1;
 imname1=strcat(num2str(i),'.jpg');
 imname1=strcat('testphotos\',imname1);
 picture1=imread(imname1);
 imshow(picture1);
 ind=find(label==label(i));
 n=size(ind);
 relat=zeros(size(ind));
 for j=1:n
    relat(j)=relation(C(:,:,i),C(:,:,ind(j)));
 end
    [la,indd]=sort(relat);
    k=ind(indd(2));
    imname2=strcat(num2str(k),'.jpg');
    imname2=strcat('testphotos\',imname2);
    picture1=imread(imname2);
    figure();
    imshow(imname2);

    k=ind(indd(3));
    imname2=strcat(num2str(k),'.jpg');
    imname2=strcat('testphotos\',imname2);
    picture1=imread(imname2);
    figure();
    imshow(imname2);

     k=ind(indd(4));
    imname2=strcat(num2str(k),'.jpg');
    imname2=strcat('testphotos\',imname2);
    picture1=imread(imname2);
    figure();
    imshow(imname2);

     k=ind(indd(5));
    imname2=strcat(num2str(k),'.jpg');
    imname2=strcat('testphotos\',imname2);
    picture1=imread(imname2);
    figure();
    imshow(imname2);

     k=ind(indd(6));
    imname2=strcat(num2str(k),'.jpg');
    imname2=strcat('testphotos\',imname2);
    picture1=imread(imname2);
    figure();
    imshow(imname2);
```

2.5 Get hog descriptor

```
function H = getHOGDescriptor(img)


% The number of bins to use in the histograms.
numBins = 9;

% The cells are 24 x 24 pixels.
cellSize = 24;

% Empty vector to store computed descriptor.
H = [];




% Compute the number cells horizontally and vertically
numHorizCells = width / cellSize;
numVertCells = height / cellSize;

% ===============================
%    Compute Gradient Vectors
% ===============================
% Compute the gradient vector at every pixel in the image.

hx = [-1,0,1];
hy = -hx';

% Compute the derivative in the x and y direction for every pixel.
dx = imfilter(double(img), hx);
dy = imfilter(double(img), hy);

% Convert the gradient vectors to polar coordinates (angle and magnitude).
angles = atan2(dy, dx);
magnit = ((dy.^2) + (dx.^2)).^.5;

% =================================
%     Compute Cell Histograms
% =================================
% Compute the histogram for every cell in the image. We'll combine the cells
% into blocks and normalize them later.

% Create a three dimensional matrix to hold the histogram for each cell.
histograms = zeros(numVertCells, numHorizCells, numBins);

% Cast the image to floating point values.
img = double(img);
```

```
% For each cell in the y-direction...
for row = 0:(numVertCells - 1)

   % Compute the row number in the 'img' matrix corresponding to the top
   % of the cells in this row. Add 1 since the matrices are indexed from
1.
   rowOffset = (row * cellSize) + 1;

   % For each cell in the x-direction...
   for col = 0:(numHorizCells - 1)

      % Select the pixels for this cell.

      % Compute column number in the 'img' matrix corresponding to the left
      % of the current cell. Add 1 since the matrices are indexed from 1.
      colOffset = (col * cellSize) + 1;

      % Compute the indices of the pixels within this cell.
      rows = rowOffset : (rowOffset + cellSize - 1);
      cols = colOffset : (colOffset + cellSize - 1);

      % Select the angles and magnitudes for the pixels in this cell.
      cellAngles = angles(rows, cols);
      cellMagnitudes = magnit(rows, cols);

      % Compute the histogram for this cell.
      % Convert the cells to column vectors before passing them in.
      histograms(row + 1, col + 1, :) = getHistogram(cellMagnitudes(:),
cellAngles(:), numBins);
   end

end

% ===================================
%      Block Normalization
% ===================================

% Take 2 x 2 blocks of cells and normalize the histograms within the block.
% Normalization provides some invariance to changes in contrast, which can
% be thought of as multiplying every pixel in the block by some coefficient.

% For each cell in the y-direction...
for row = 1:(numVertCells - 1)
   % For each cell in the x-direction...
   for col = 1:(numHorizCells - 1)

      % Get the histograms for the cells in this block.
      blockHists = histograms(row : row + 1, col : col + 1, :);

      % Put all the histogram values into a single vector (nevermind the
      % order), and compute the magnitude.
      % Add a small amount to the magnitude to ensure that it's never 0.
      magnitude = norm(blockHists(:)) + 0.01;
```

```
        % Divide all of the histogram values by the magnitude to normalize
        % them.
        normalized = blockHists / magnitude;

        % Append the normalized histograms to our descriptor vector.
        H = [H; normalized(:)];
    end
end

end


function H = getHistogram(magnitudes, angles, numBins)

% Compute the bin size in radians. 360 degress = 2*pi.
binSize = (2 * pi) / numBins;

% The angle values will range from -pi to pi.
minAngle = -pi;

% The gradient angle for each pixel will fall between two bin centers.
% For each pixel, we split the bin contributions between the bin to the left
% and the bin to the right based on how far the angle is from the bin centers.

% For each pixel, compute the index of the bin to the left and to the right.
%
% The histogram needs to wrap around at the edges--pixels on the far edges
of
% the histogram (i.e., close to -pi or pi) will contribute partly to the
bin
% at that edge, and partly to the bin on the other end of the histogram.
% For pixels on the far left edge of the histogram, leftBinIndex will be
0.
% Likewise, for pixels on the far right edge, rightBinIndex will be 10.
leftBinIndex = round((angles - minAngle) / binSize);
rightBinIndex = leftBinIndex + 1;

% For each pixel, compute the center of the bin to the left.
leftBinCenter = ((leftBinIndex - 0.5) * binSize) - pi;

% For each pixel, compute the fraction of the magnitude
% to contribute to each bin.
rightPortions = angles - leftBinCenter;
leftPortions = binSize - rightPortions;
rightPortions = rightPortions / binSize;
leftPortions = leftPortions / binSize;

% Before using the bin indeces, we need to fix the '0' and '10' values.
% Recall that the histogram needs to wrap around at the edges--bin "0"
% contributions, for example, really belong in bin 9.
% Replace index 0 with 9 and index 10 with 1.
leftBinIndex(leftBinIndex == 0) = 9;
rightBinIndex(rightBinIndex == 10) = 1;
```

```
% Create an empty row vector for the histogram.
H = zeros(1, numBins);

% For each bin index...
for (i = 1:numBins)
   % Find the pixels with left bin == i
   pixels = (leftBinIndex == i);

   % For each of the selected pixels, add the gradient magnitude to bin 'i',
   % weighted by the 'leftPortion' for that pixel.
   H(1, i) =H(1,i)+ sum(leftPortions(pixels)' * magnitudes(pixels));

   % Find the pixels with right bin == i
   pixels = (rightBinIndex == i);

   % For each of the selected pixels, add the gradient magnitude to bin 'i',
   % weighted by the 'rightPortion' for that pixel.
   H(1, i) =H(1,i)+ sum(rightPortions(pixels)' * magnitudes(pixels));
end

end
```