Report of Lab9

Yuxiang Chen 5110309783

December 21, 2012

Contents

1 The Purpose of This Experiment

In this experiment, our main purpose is to use a hash table to eliminate the time when we try to match a certain picture. Usually, we use the method referred in the pdf files of establishing a twelve-dimension vector for each picture we're to search, and then compare them with each other, getting the one that is the most similar. However, this may lead to a waste of time when we just compare these pictures one by one. So in order to save time, we may just get a part of their characters and set up a hash table using which we can promptly find the certain group that has the same feature with the target. Thus we can spare quite a lot time. So in this experiment. we need to use the LSH method to accomplish our goal to save time and make it easy when matching pictures.

2 The Main Part of the Experiment

Since we have to compare the effect of those two method, I write two programs. One for the normal NN or KNN method, and the other for LSH. And at last I print out the matched result and the time used in the searching program.

2.1 The method of KNN

This method is the so-called 'violent' method, since we just compare the twelve-dimension vector of them and if they're the same, we can judge that the two pictures are similar. Although this is a correct method, it really wastes time

since it will also compare those graphs which is definitely has no similarity with the target.

In this part, I calculate the percentage of green, red and blue in the four parts of the picture, and they consist the twelve-dimension vector we need. Then I change it into the '0 1 2' form we need. And following are my codes:

```
1  import sys
2  import string
3  import time
4  from opencv.cv import *
5  from opencv.highgui import *
6
7  def get_data(filename):
8      twelve=list()
9      p=list()
10     image = cvLoadImage (filename,1)
11     width=image.width
12     height=image.height
13     semi_w=width/2
14     semi_h=height/2
15     red=0
16     green=0
17     blue=0
18     for i in range(0,semi_w):
19         for j in range(0,semi_h):
20             red+=image[j,i][2]
21             blue+=image[j,i][0]
22             green+=image[j,i][1]
23     total=red+green+blue
24     p.append(float(blue)/total)
25     p.append(float(green)/total)
26     p.append(float(red)/total)
27     red=0
28     green=0
29     blue=0
30     for i in range(0,semi_w):
31         for j in range(semi_h,height):
32             red+=image[j,i][2]
33             blue+=image[j,i][0]
34             green+=image[j,i][1]
35     total=red+green+blue
36     p.append(float(blue)/total)
37     p.append(float(green)/total)
38     p.append(float(red)/total)
39     red=0
40     green=0
```

```
41      blue=0
42      for i in range(semi_w,width):
43          for j in range(0,semi_h):
44              red+=image[j,i][2]
45              blue+=image[j,i][0]
46              green+=image[j,i][1]
47      total=red+green+blue
48      p.append(float(blue)/total)
49      p.append(float(green)/total)
50      p.append(float(red)/total)
51      red=0
52      green=0
53      blue=0
54      for i in range(semi_w,width):
55          for j in range(semi_h,height):
56              red+=image[j,i][2]
57              blue+=image[j,i][0]
58              green+=image[j,i][1]
59      total=red+green+blue
60      p.append(float(blue)/total)
61      p.append(float(green)/total)
62      p.append(float(red)/total)
63      for k in range(0,12):
64          if p[k]>=0 and p[k]<0.3:
65              twelve.append(0)
66          if p[k]>=0.3 and p[k]<0.6:
67              twelve.append(1)
68          if p[k]>=0.6:
69              twelve.append(2)
70      return twelve
71
72  def get_all_twelve():
73      total_vector=list()
74      for t in range(1,41):
75          filename=str(t)+".jpg"
76          twelve=get_data(filename)
77          total_vector.append(twelve)
78      return total_vector
79
80  def get_all_target():
81      new_twelve=get_data("target.jpg")
82      return new_twelve
83
84  total_vector=get_all_twelve()
85  new_twelve=get_all_target()
86  start=time.clock()
```

```
for i in range(0,40):
    if total_vector[i]==new_twelve:
        print "The picture is "+str(i+1)+".jpg"
end=time.clock()
delta=end-start
print "time:",delta
```

And after this program, I get the result like this picture. But we get two result, the program thinks that both 12.jpg and 38.jpg is similar to the target. And if we see it carefully, 12.jpg, although not the exact graph, really resembles our target in some way. Whatever, as the method itself is to find the similar result, not the exact one, so we can think it attain the expectation we want.

```
>>>
The picture is 12.jpg
The picture is 38.jpg
time: 0.0138216831083
>>>
```

Figure 1: The result of KNN

2.2 The method of LSH

In this part, we have to make an hash table which contains some feature of the pictures, say, maybe some digits of its vector. And then if we use the hash function we establish and get a key, then search the key to get a certain bin in the data base. And in this way, we find the bin which the graph most similar to our target in. So we can just use the KNN method to search this bin, but with less samples to traverse. Thus we can save much time.
And in this section, I first make a hash table of the data base given, then just find which bin the target is in. And then the program is just the same as the first one, finding the result by comparing the twelve-dimension vector. I set the projection to be 1,3,7,8 by default, you can modify it directly in the program if you want, and my program will get the new hamming code automatically. Here are my codes:

```
import sys
import string
import time
from opencv.cv import *
from opencv.highgui import *
#           C=2  d=12,d'=24
I=[1,3,7,8]#              I                                   hamming
c=2
projection=dict()
for i in range(1,13):
    for num in I:
        if (i-1)*c+1<=num and num<=i*c:
```

```
13              if projection.has_key(str(i)):
14                  projection[str(i)].append(num)
15              else:
16                  projection[str(i)]=[num]
17  hashkeys=projection.keys()
18
19  def get_data(filename):
20      twelve=list()
21      p=list()
22      image = cvLoadImage (filename,1)
23      width=image.width
24      height=image.height
25      semi_w=width/2
26      semi_h=height/2
27      red=0
28      green=0
29      blue=0
30      for i in range(0,semi_w):
31          for j in range(0,semi_h):
32              red+=image[j,i][2]
33              blue+=image[j,i][0]
34              green+=image[j,i][1]
35      total=red+green+blue
36      p.append(float(blue)/total)
37      p.append(float(green)/total)
38      p.append(float(red)/total)
39      red=0
40      green=0
41      blue=0
42      for i in range(0,semi_w):
43          for j in range(semi_h,height):
44              red+=image[j,i][2]
45              blue+=image[j,i][0]
46              green+=image[j,i][1]
47      total=red+green+blue
48      p.append(float(blue)/total)
49      p.append(float(green)/total)
50      p.append(float(red)/total)
51      red=0
52      green=0
53      blue=0
54      for i in range(semi_w,width):
55          for j in range(0,semi_h):
56              red+=image[j,i][2]
57              blue+=image[j,i][0]
58              green+=image[j,i][1]
```

```
59      total=red+green+blue
60      p.append(float(blue)/total)
61      p.append(float(green)/total)
62      p.append(float(red)/total)
63      red=0
64      green=0
65      blue=0
66      for i in range(semi_w,width):
67          for j in range(semi_h,height):
68              red+=image[j,i][2]
69              blue+=image[j,i][0]
70              green+=image[j,i][1]
71      total=red+green+blue
72      p.append(float(blue)/total)
73      p.append(float(green)/total)
74      p.append(float(red)/total)
75      for k in range(0,12):
76          if p[k]>=0 and p[k]<0.3:
77              twelve.append(0)
78          if p[k]>=0.3 and p[k]<0.6:
79              twelve.append(1)
80          if p[k]>=0.6:
81              twelve.append(2)
82      return twelve
83
84  def get_all_twelve():
85      total_vector=list()
86      for t in range(1,41):
87          filename=str(t)+".jpg"
88          twelve=get_data(filename)
89          total_vector.append(twelve)
90      return total_vector
91
92  def get_all_target():
93      new_twelve=get_data("target.jpg")
94      return new_twelve
95
96  def get_hash_key(vector,hashkeys):
97      key=list()
98      for j in range(0,len(hashkeys)):
99          x=vector[int(hashkeys[j])-1]
100         for num in projection[hashkeys[j]]:
101             if num-c*(int(hashkeys[j])-1)<=x:
102                 key.append(1)
103             else:
104                 key.append(0)
```

```
105     return key
106
107 def get_hash(total_vector,projection):
108     hash_dict={}
109     for i in range(0,40):
110         key=get_hash_key(total_vector[i],hashkeys)
111         if hash_dict.has_key(str(key)):
112             hash_dict[str(key)].append(i)
113         else:
114             hash_dict[str(key)]=[i]
115     return hash_dict
116
117 def get_target_hash(newtwelve):
118     new_key=get_hash_key(newtwelve,hashkeys)
119     return new_key
120
121 if __name__ == '__main__':
122     total_vector=get_all_twelve()
123     newtwelve=get_all_target()
124     hash_dict=get_hash(total_vector,projection)
125     newkey=get_target_hash(newtwelve)
126     start=time.clock()
127     result_set=hash_dict[str(newkey)]
128     for num in result_set:
129         if newtwelve==total_vector[num]:
130             print "The picture is "+str(num+1)+".jpg"
131     end=time.clock()
132     delta=end-start
133     print "time:",delta
```

And in this part I get the same result as the KNN method – two result. But there's a significant time-drop by using this method. So we can just believe this method can save us lots of time when the data base is big enough. And then I change the projection to be 2,3,7,10 and still get the same result. Here are the screenshots.

```
>>>
The picture is 12.jpg
The picture is 38.jpg
time: 0.00730016902844
>>>
```

Figure 2: The result of 1,3,7,8

```
>>>
The picture is 12.jpg
The picture is 38.jpg
time: 0.022180733622
>>>
```

Figure 3: The result of 2,3,7,10

3 The Problems I Met and My Thoughts

As the last program in this semester, I feel happy to accomplished it in time all by myself, as all of the former codes. So after finishing this program, what I realise first is the importance of make a structure like a directory when searching files. Since in this way, we can largely eliminate the time used in some unrelated objects. And since the experiment isn't very difficult, so I didn't meet much trouble at first. After understanding what the question means, I directly write it. And at last, I feel very thankful for TA's generous help when I emailed him for some questions. Of course, although the course is over, I will keep on learning about image processing, since I think there's a lot fun in it. Thank you.