HW5 report

Group 5

0516222 許芳瑀 0516209 呂淇安 0516326陳廷達

1. Introduction

   In this homework, we try to build a classifier to categorize images into one of 15 scene types. We tried different methods from task 1 to task 3 to get better results.

   In task 1, we implemented tiny images representation, then use brute force method to find k-nearest neighbor and decide the type of test image.

   In task 2, we try the combination of bag of SIFT representation and nearest neighbor classifier, the accuracy of classification rise to about 50%.

   In task 3, nearest neighbor classifier is replaced with linear SVM classifier.

2. Implementation procedure

   Task1:

   I. resize the picture as 16*16

```
42  def Resize_Normal(Img_list):
43      Result = []
44      for ImgClass in Img_list:
45          img, Class_ = ImgClass
46
47          Rsz = cv2.resize( img, dsize=(16, 16) )
48          temp = [ pixel for row in Rsz for pixel in row ]
49          #normalize
50          Nmlz = [ float(pixel) / sum(temp) for pixel in temp ]
51
52          Result.append( (Nmlz, Class_) )
53      return Result
```

   II. Use k nearest neighbor algorithm, it means that given a test instance i, find the k closest neighbors and their labels. Predict i's label as the majority of the labels of the k nearest neighbors

```
9   def euclidean_distance(row1, row2):
10      distance = 0.0
11      if len(row1) != len(row2):
12      print("not equal", len(row1), len(row2))
13      for i in range(len(row1)-1):
14          distance += (row1[i] - row2[i])**2
15      return math.sqrt(distance)
16
17
18  # Locate the most similar neighbors
19  def GetNeighbors(train, test_row, num_neighbors):
20      distances = list()
21      for (train_row, Class_) in train:
22          dist = euclidean_distance(row1 = test_row, row2 = train_row)
23          distances.append((Class_, dist))
24      distances.sort(key = lambda y: y[1])
25      neighbors = list()
26      for i in range(num_neighbors):
27          neighbors.append(distances[i][0])
28      return neighbors

68  ## NN
69  for k in range(1,11):   #testing on different k
70      iter_ = 0           #counter of image
71      right = 0           #counter of right prediction
72
73      for (test_row, Class_) in RN_test_list:
74          iter_ += 1
75          neighbors = GetNeighbors(train = RN_train_list, test_row = test_row, num_neighbors = k)
76          output_values = [row for row in neighbors]
77          prediction = max(set(output_values), key = output_values.count)
78          if prediction == Class_:
79              right += 1
80
81      Result.append((k, right / (float(iter_))))
82      print(k, iter_, right, right / float(iter_))
83
```

Task2:

I. find SIFT keypoints of all train pictures

```
# find kp
print("finding keypoints of training data")
train_kp = []
for i in train_list:
    img, img_class = i
    sift = cv2.xfeatures2d.SIFT_create()
    kp, descriptors = sift.detectAndCompute(img, None)
    train_kp.append(descriptors)
train_kp = np.concatenate(train_kp, axis=0)
```

II. function *k_means(data, k, threas)* build cluster of all train key points, we use k=34, threas = 1000 in task 2

it returns k centers of k clusters.

```
while(abs_error > threas):
    # cluster 為有k個空list的list
    cluster = []
    for i in range(k):
        cluster.append([])
    # classify pts
    for pt in data:
        classIdx = -1
        min_distance = float("inf")
        for idx, c in enumerate(center):
            #print(pt)
            #print(c)
            new_distance = distance(pt, c)
            if new_distance < min_distance:
                classIdx = idx
                min_distance = new_distance
        cluster[classIdx].append(pt)
    print('k_cluster, k=', k)
    for C in cluster:
        print(len(C))
    cluster = np.array(cluster)
```

while the error is bigger than threshold, we repeatedly classify all points into k clusters and compute new center of each cluster.

```
    # calculate new center & error
    new_error = 0
    for idx, C in enumerate(cluster):
        center[idx] = np.mean(C, axis=0)
        #print(idx, center[idx].shape)
        #print(center[idx])
        for point in C:
            new_error += distance(center[idx], point)
    abs_error = abs(error - new_error)
    error = new_error
    #print(abs_error)

return center
```

III. build histogram of train pictures

```
train_histogram = []
for i in train_list:
    img, img_class = i
    sift = cv2.xfeatures2d.SIFT_create()
    kp, descriptors = sift.detectAndCompute(img, None)
    train_histogram.append((build_histogram(descriptors, center), img_class))
```

```
def build_histogram(descriptor, center):
    '''
    input: descriptor of a picture, centers of k-cluster
    output: histogram of the picture
    '''
    histogram = np.zeros(center.shape[0])
    for kp in descriptor:
        label = -1
        min_distance = float("inf")
        for idx, c in enumerate(center):
            dis = distance(kp, c)
            if dis < min_distance:
                lable = idx
                min_distance = dis
            histogram[lable] += 1
    return histogram
```

IV. for test data, find SIFT key points and build the histogram

```
# testing
print("finding keypoints of test data")
test_kp = []
for i in test_list:
    img, img_class = i
    sift = cv2.xfeatures2d.SIFT_create()
    kp, descriptors = sift.detectAndCompute(img, None)
    test_kp.append(descriptors)
test_kp = np.concatenate(test_kp, axis=0)

print("generating histogram of testing data")
test_histogram = []
for i in test_list:
    img, img_class = i
    sift = cv2.xfeatures2d.SIFT_create()
    kp, descriptors = sift.detectAndCompute(img, None)
    test_histogram.append((build_histogram(descriptors, center), img_class))
```

V. use function *GetNeighbors(train, test_row, num_neighbors)* in task 1, find k nearest neighbors of each test picture and classify them.

Task3:

I. use the train_histogram got in step2 with support vector machine algorithm, The objective of the svm algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. In this step, we use libsvm package to simply the process. First we create the data in specified type.

```
# write train data
seq = np.arange(0, len(tr_histogram))
np.random.shuffle(seq)
data = open("tr.txt",'w+')
for i in seq:
    s = str(tr_histogram[i][1])+' 1:'+str(tr_histogram[i][0][0])+' 2:'+str(tr_histogram[i][0][1])+' 3:'+str(tr_histogram[i][0][2])
    print(s,file=data)
data.close()

# write test data
seq = np.arange(0, len(te_histogram))
np.random.shuffle(seq)
data = open("te.txt",'w+')
for i in seq:
    s = str(te_histogram[i][1])+' 1:'+str(te_histogram[i][0][0])+' 2:'+str(te_histogram[i][0][1])+' 3:'+str(te_histogram[i][0][2])
    print(s,file=data)
data.close()
```

II. Use grid.py in libsvm to find best parameters

```
C:\Users\user\anaconda3\envs\cv2_v3.4\Lib\libsvm\windows>python grid.py train_data
gnuplot executable not found
[local] 5 -7 47.2667 (best c=32.0, g=0.0078125, rate=47.2667)
[local] -1 -7 32.0667 (best c=32.0, g=0.0078125, rate=47.2667)
[local] 5 -1 45.8667 (best c=32.0, g=0.0078125, rate=47.2667)
[local] -1 -1 46.0 (best c=32.0, g=0.0078125, rate=47.2667)
```

III. Train model with best parameters

```
C:\Users\user\anaconda3\envs\cv2_v3.4\Lib\libsvm\windows>svm-train.exe -c 8.0 -g 0.03125 train100
*
optimization finished, #iter = 103
nu = 0.087297
obj = -74.153030, rho = 1.615177
nSV = 33, nBSV = 6
*.*
optimization finished, #iter = 202
nu = 0.262965
obj = -272.800453, rho = 0.943790
nSV = 75, nBSV = 37
*.*
optimization finished, #iter = 215
nu = 0.368263
obj = -418.702488, rho = 3.551816
nSV = 99, nBSV = 57
*
```

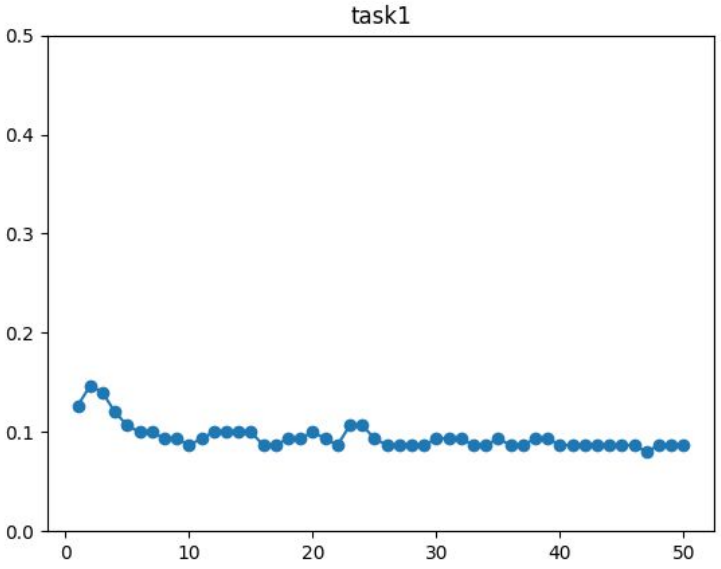
3. Experimental results

Task1:

k=1~10 accuracy

```
1 150 19 0.12666666666666668
2 150 22 0.14666666666666667
3 150 21 0.14
4 150 18 0.12
5 150 16 0.10666666666666667
6 150 15 0.1
7 150 15 0.1
8 150 14 0.09333333333333334
9 150 14 0.09333333333333334
10 150 13 0.08666666666666667
(env) apple@AppledeMacBook-Air lab5 %
```

line chart of different k, we get best accuracy when k = 2



Task2:

```
21 150 72 0.48
22 150 75 0.5
23 150 75 0.5
24 150 73 0.4866666666666667
25 150 71 0.47333333333333333
26 150 70 0.4666666666666667
27 150 72 0.48
28 150 72 0.48
29 150 69 0.46
30 150 72 0.48
31 150 70 0.4666666666666667
32 150 72 0.48
33 150 73 0.4866666666666667
34 150 76 0.5066666666666667
35 150 73 0.4866666666666667
36 150 72 0.48
37 150 71 0.47333333333333333
38 150 70 0.4666666666666667
39 150 72 0.48
40 150 71 0.47333333333333333
```

Task3:

The best result is 48%

4. Discussion

   1. Since the training data is much more bigger than any other homeworks before, it spends much time to train the model. So we store the histogram of Bag of SIFT representation of every pictures when we finished task 2. So we can just load the file in task 3, which saves much time.

   2. At first we only have 3 vocabularies in bags of SIFT, thus the result is bad. After adding more vocabularies ,the result get better.

   3. The best accuracy of part 3 is 48%, which doesn't hit the goal. We have tried many methods(add vocabularies, scale data……) but it still fail.

5. Conclusion

   After finishing this homework, we know more about the model in machine learning. There are a lot of theory in computer vision can be used in many applications. The algorithms about classifier to

categorize images and the experience of this homework will help us a lot in the future.

6. Work assignment plan between team members.

   陳廷達:33%
   許芳瑀:33%
   呂淇安:33%