Coursework3 Scene Recognition (For COMP6223)

Author: Runyao Zhou 30018714 rz1f18; ID: 30018714

Coding Environment: python (Jupyter)

Run1: KNN Classifier

Description of the implementation of the classifiers

1. Input all the images from 'training' and 'testing'.

2. Crop each into a square about the centre, and resize into 16*16.

3. Divide 'training' into 5 parts. Each part contains 20 images from each class. Named 'train_1', 'train_2'… 'train_5'.

4. Every image is packed into a vector by concatenating each image row. Then make them to have zero mean and unit length by subtracting the overall average for each pixel and make each pixel be divided by the standard deviation.

```
def iKNN(tr,te,k):
    te_original=copy.deepcopy(te) #for calculate accurary
    te_new=copy.deepcopy(te)
    num_train=len(tr)
    num_test=len(te)
    for i in range(num_test):
        distances=[]
        for j in range(num_train):
            distance=np.sqrt(np.sum(np.square(te[i][0]-tr[j][0])))
            distances.append(distance)
        dis_in_k_label_train=np.array(tr)[np.argsort(distances)[:k]][:,1]
        u,cishu=np.unique(dis_in_k_label_train,return_inverse=True) #cishu
        te_new[i][1]=u[np.argmax(np.bincount(cishu))] #!!!I used to use 't
    ##accuracy
    acc=np.mean(np.array(te_new)[:,1]==np.array(te_original)[:,1])
    return k,acc
```

5. Calculate the Euclidean distance of a 'testing' image to all 'training' images, and find out the training images whose distances are less than k. Among these images, the most common label is the prediction label of the 'testing' image.

6. Cross validation. Make each 'train_( )' to be 'test'. Calculate k from 1 to 30 each time. So in the 30*5=150 'k's, find one who have the highest accuracy. The accuracy is how many labels in the prediction is right compared with the labels given.

I checked all the 'distances' in 'variable explorer' , most of them are between 15-25, so I choose k from 1 to 30.

Result

The best k is 11, and the highest accuracy is 26.33%.

Run2: simple Bag-of-visual-words feature, K-Means vocabulary, LinearSVC

Description of the implementation of the classifiers

1. Input all the images from ‘training’ and ‘testing’.

2. Crop each into a square about the centre, and resize into 40*40.

3. Sample 8x8 patches, sampled every 4 pixels in the x and y directions. Every patch has been flattened into a vector. Then mean-centring and normalising each patch by subtracting the overall average for each pixel and make each pixel be divided by the standard deviation.

4. Split ‘training’ into ‘train’ and ‘test’ in which ‘test’ contains 20%-30% for accuracy prediction. All the ‘training’ will be used for final prediction printing (.txt).

5. Use KMeans to learn a vocabulary of 500 clusters.

```
#KMeans
clusters=500
kmeans=KMeans(n_clusters=clusters)
kmeans.fit(X_train_2d)
labels_train=kmeans.labels_
labels_test=kmeans.predict(X_test_2d)
```

6. Define a ‘bow_descriptortor’ to build bag-of-visual-words features. Separate each image’s information from label values learned by KMeans. Count numbers in each cluster (from 0 to 499). Then we get the bag-of-visual-words features.

7. Use linear SVC to predict ‘testing’ labels and calculate accuracy. Prediction labels will be printed out at last when using full ‘training’.

```
#SVC,linear
svc=svm.SVC(C=10,kernel='linear',gamma=10)
svc.fit(bow_descriptor_train,y_train)
#accuracy
y_pre=svc.predict(bow_descriptor_test)
print(classification_report(y_test,y_pre))
```

Result

Picture on the right shows the best result in run2.

	precision	recall	f1-score	support
Coast	0.45	0.62	0.53	16
Forest	0.55	0.50	0.52	22
Highway	0.70	0.58	0.64	24
Insidecity	0.39	0.53	0.45	17
Mountain	0.58	0.65	0.61	17
Office	0.31	0.22	0.26	23
OpenCountry	0.50	0.21	0.29	29
Street	0.73	0.48	0.58	23
Suburb	0.52	0.68	0.59	19
TallBuilding	0.68	0.75	0.71	20
bedroom	0.31	0.25	0.28	20
industrial	0.29	0.35	0.32	17
kitchen	0.16	0.33	0.22	12
livingroom	0.31	0.19	0.23	27
store	0.21	0.43	0.29	14
avg / total	0.46	0.44	0.43	300

And we find out parameters that influence the result:

i. Size of image resize.
Bigger size makes the accuracy better, but the amount of data will be large. If we run a 80 size, 7 hours will not be enough. 60 size takes 4 hours.

ii. Size of patch.
Big patches make worse result. If we run a 16*16 patch instead of 8*8, the accuracy will be down to 23%.

iii. Length of move step of patch
Big steps make worse result. If we run in step of 8 instead of 4 , the accuracy will be down to 23%.

iv. Cluster in KMeans
Small cluster will be bad, but big cluster only works better when we have bigger size of image (e.g. 60).

v. The percentage taken as a test set from the training set (little influence when the percentage is between 20%-30%).

Run3: SIFT and Dense SIFT, K-Means, LinearSVC

Description of the implementation of the classifiers

1. Input all the images from 'training' and 'testing'.

2. Write code to resize the images. We use it for some experiments, but finally we set the resize ratio to 100% to get the original image.

3. Get features.

3.1 Get SIFT features. We tried to use SIFT in OpenCV. The number of features are limited at 100, 200 or the amount of data will be too large (About 600,000*128 SIFT features, >5 hours running).

```
sift=cv2.xfeatures2d.SIFT_create(nfeatures=100)
kp_temp,des_temp=sift.detectAndCompute(images[i],None)
```

We also tried SIFT in VLFeat as figure shows below. A version for python, called cyvlfeat.

```
from cyvlfeat.sift import sift

kp_temp,des_temp=sift(images[i],edge_thresh=2,compute_descriptor=True)
```

Default for edge_thresh is 10, small edge_thresh limit the amount of data.

3.2 Get Dense SIFT features. D-SIFT in OpenCV has been removed from the version 3. Then we choose VLFeat to run D-SIFT.

```
from cyvlfeat.sift import dsift

kp_temp,des_temp=dsift(images[i], step=16, size=20, fast=True)
```

'size' is size of the SIFT descriptor, 'step' is the step SIFT descriptor moves. We choose size=20 and step=16 to limit the amount of data to about 200,000 features.

4. Use KMeans to learn a vocabulary of 1200 clusters.

5. Use the same function to build bag-of-visual-words features.

6. Use linear SVC to predict 'testing' labels and calculate accuracy. Prediction labels will be printed out at last when using full 'training'.

```
from sklearn import svm

svc=svm.SVC(C=10,kernel='linear',gamma=10)
```

We tried non-linear SVM as 'poly' and 'rbf', but the result is bad.

Result

The best results are showed as below. The figure on the left is from SIFT, and the right one is from D-SIFT.

	precision	recall	f1-score	support
Coast	0.57	0.57	0.57	37
Forest	0.50	0.50	0.50	28
Highway	0.58	0.58	0.58	31
Insidecity	0.39	0.26	0.31	35
Mountain	0.29	0.38	0.33	26
Office	0.36	0.36	0.36	25
OpenCountry	0.41	0.23	0.30	39
Street	0.32	0.37	0.34	30
Suburb	0.46	0.52	0.48	31
TallBuilding	0.33	0.41	0.37	29
bedroom	0.12	0.16	0.14	25
industrial	0.19	0.19	0.19	26
kitchen	0.32	0.35	0.34	31
livingroom	0.17	0.21	0.19	24
store	0.30	0.18	0.23	33
avg / total	0.37	0.36	0.35	450

	precision	recall	f1-score	support
Coast	0.85	0.68	0.76	25
Forest	0.78	0.88	0.82	16
Highway	0.86	0.72	0.78	25
Insidecity	0.52	0.52	0.52	21
Mountain	0.70	0.88	0.78	16
Office	0.58	0.79	0.67	19
OpenCountry	0.64	0.70	0.67	20
Street	0.81	0.72	0.76	18
Suburb	0.89	0.89	0.89	27
TallBuilding	0.57	0.81	0.67	16
bedroom	0.50	0.55	0.52	20
industrial	0.50	0.47	0.48	17
kitchen	0.56	0.43	0.49	21
livingroom	0.41	0.33	0.37	21
store	0.47	0.39	0.42	18
micro avg	0.65	0.65	0.65	300
macro avg	0.64	0.65	0.64	300
weighted avg	0.65	0.65	0.65	300

And we find out parameters that influence the result:

i. For SIFT
The amount data is most important. When the number features we limited increased from 100 to 200, the accuracy increased from 26%-27% to 35%-37%.

ii. For D-SIFT
Small 'size 'and 'step' provide more data and increase the accuracy. But when data increase from 100,000 to 200,000 features the accuracy stays almost the same. Overall, D-SITF is better than SIFT under the same amount of data.