Feedback for yz13u18_zw1y18

Marks Breakdown

Successful Completion	6	/ 6
Excellent Implementation	4	/ 4
Excellent Reporting	3	/ 4
Approach	5	/ 6
Total	**18**	**/ 20**

Comments

Excellent overall - well done.

Excellent implementation.

Good report. Would have benefitted from more discussion on experimental findings.

Very good range of experimentation for run 3, with sound approaches used.

(See also any additional comments in following pages)

COMP3204/6223 Coursework 3 Report

Submission ID	yz13u18	
Member 1	yz13u18	Yigang Zhou
Member 2	zw1y18	Ziyang Wei

Run 1

Overall Results

Total instances	2985
Total correct	550
Total incorrect	2435
Accuracy	**0.184**
Error Rate	0.816
Average Class Accuracy	0.176
Average Class Error Rate	0.824

bit low

Per Class Results

Class	Class Accuracy	Class Error Rate	Actual Count	Predicted Count
bedroom	0.112	0.888	116	76
industrial	0.028	0.972	211	53
insidecity	0.082	0.918	208	102
livingroom	0.095	0.905	189	96
tallbuilding	0.086	0.914	256	114
forest	0.154	0.846	228	173
store	0.005	0.995	215	15
opencountry	0.313	0.687	310	566
coast	0.388	0.612	260	713
street	0.339	0.661	192	140
office	0.087	0.913	115	72
suburb	0.149	0.851	141	29
kitchen	0.027	0.973	110	49
mountain	0.142	0.858	274	189
highway	0.638	0.363	160	598

Run 2

Overall Results

Total instances	2985
Total correct	2093
Total incorrect	892
Accuracy	**0.701**
Error Rate	0.299
Average Class Accuracy	0.693
Average Class Error Rate	0.307

Per Class Results

Class	Class Accuracy	Class Error Rate	Actual Count	Predicted Count
bedroom	0.595	0.405	116	187
industrial	0.517	0.483	211	202
insidecity	0.784	0.216	208	238
livingroom	0.381	0.619	189	151
tallbuilding	0.688	0.313	256	201
forest	0.925	0.075	228	251
store	0.609	0.391	215	208
opencountry	0.648	0.352	310	279
coast	0.808	0.192	260	268
street	0.740	0.260	192	187
office	0.817	0.183	115	134
suburb	0.929	0.071	141	145
kitchen	0.391	0.609	110	102
mountain	0.788	0.212	274	255
highway	0.781	0.219	160	177

Run 3

Overall Results

Total instances	2985
Total correct	2093
Total incorrect	892
Accuracy	**0.701**
Error Rate	0.299
Average Class Accuracy	0.693
Average Class Error Rate	0.307

Per Class Results

Class	Class Accuracy	Class Error Rate	Actual Count	Predicted Count
bedroom	0.595	0.405	116	187
industrial	0.517	0.483	211	202
insidecity	0.784	0.216	208	238
livingroom	0.381	0.619	189	151
tallbuilding	0.688	0.313	256	201
forest	0.925	0.075	228	251
store	0.609	0.391	215	208
opencountry	0.648	0.352	310	279
coast	0.808	0.192	260	268
street	0.740	0.260	192	187
office	0.817	0.183	115	134
suburb	0.929	0.071	141	145
kitchen	0.391	0.609	110	102
mountain	0.788	0.212	274	255
highway	0.781	0.219	160	177

UNIVERSITY OF Southampton

Scene Recognition

COMP6223 Coursework 3

Yigang Zhou 30041996
Ziyang Wei 30305276

COMP6223 Coursework 3: Scene Recognition

1 Introduction

Scene classification application is an essential field in computer vision. The high accuracy and performance are desired in many practical applications such as unmanned vehicle and augmented reality. Compared with traditional image classification application, scene classification has a wide range of data and more complex information.

In this coursework we investigated multiple approaches for scene classification, 1500 images from 15 scenes are provided. Our objective is to train the best classifier as we can to produce the best test result.

2 Method Statement

There are 1500 images in the training set, which includes 15 scenes, 100 images for each scene.

We had 3 runs for this task. For run 1, we developed a K-nearest neighbor classifier. For run 2, we used bag of words model with k-means clustering to train our one vs rest classifier. The feature exacted from each image is fixed size patches. For run 3 we developed multiple classifiers for different features, more details will be discussed in following sections.

2500 images are provided without any tags, to test the performance of our classifiers, we hand tagged 300 images with 20 images for each of the 15 scenes. Predict accuracy of each scene will be examined and discussed.

3 Technical Approach for run 1

The main steps of run 1 are:

1. Resize the origin image to smaller size, for example 16 x 16 or 8 x 8. And flatten to a one-dimension vector. This vector will be the 'Tiny image' feature of the image.
2. Each vector is normalized to its zero-mean vector by using the following equation

$$\text{normalized feature} = \frac{feature - mean(feature)}{\max(feature) - \min(feature)}$$

   When denominator is equal to zero, it will be considered as a background image. This feature will be normalized to an all zero feature.
3. Then the classified tag of an image will be calculated by applying k-nearest neighbour algorithm. The Euclidean distance of target image to all training image will be calculated and the maximum occurrence tag in the k nearest image with minimum distances will be the predicted tag.

We examined this approach with different parameters with different size of the tiny image and k values. The following figure shows the performance benchmark, it can be seen that the total predict accuracy is maximum with 8x8 tiny image size and k = 1.

Does this hold under cross-validation. I'd guess not

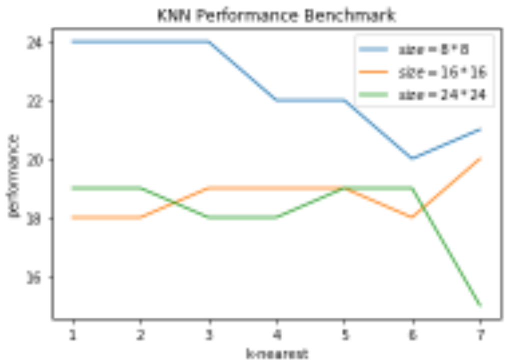

Figure 1. KNN performance benchmark for different parameters

The following table shows the prediction accuracy for each scene, it can be concluded that KNN has a better performance on outdoor scene, especially for Highway and Coast. For indoor scene like store and living room, KNN approach failed to predict all of the testing samples.

Scene	Accuracy	Scene	Accuracy
Forest	10%	Bedroom	20%
Office	20%	Highway	70%
Coast	50%	Inside city	0%
Tall Building	20%	Industrial	20%
Street	20%	Livingroom	0%
Suburb	30%	Mountain	40%
Kitchen	27%	Open country	40%
Store	0%	**Total**	**24.5%**

Table 1. KNN performance for 64-length tiny feature, k =1

For run 1 approach, the 'tiny image' feature is too simple for classifying scenes, it cannot express the transformation of a scene. Additionally, even if the illumination or exposure of a same scene is changed, using tiny image feature with KNN can still failed to classify.

4 Technical Approach for run 2

In run 2, we trained the classifier using bag-of-visual-words features exacted from each image.

The main steps of run 2 are:

1. Each image is sampled with fixed step from length and width, and a patch sized is given. After the process, each image is then divided into patches with fixed size.

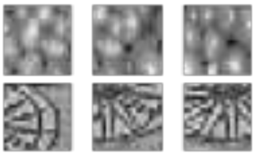

Figure 2. Patches for different image, with image resized to 128x128 patch size 16

2. Each patch is then be flattened to a one-dimensional vector, this vector is then be normalized by using the same method discussed in run 1.
3. All patches from images are then clustered into k categories by using k-means algorithm. Value k we examined is from 100 to 1000. Since the k value is

Only cluster a Sample!

relatively large while the vector to perform clustering is of length 64 to 256. It takes a long time to perform k-means clustering using the traditional method. We then come with a solution to use mini-batch k-means as an alternative method. Minibatch k-means is a variant of k-means algorithm, it uses mini batches of the input data for clustering. Minibatch is subsets of input data, randomly selected through reach iteration epoch. With this approach, computation time is reduced. We examined the performance of k-means and minibatch k-means. The result from minibatch k-means is only slightly worse than k-means, the test accuracy is reduced by 1%, but a lot of times is saved.

4. After clustering all patches into different k categories. A histogram can then be generated to express each image, it is a vector with length of k, each element represents the frequency of the corresponding category. The following figure shows the histogram of a coast image and an office image. This vector will be the feature vector of an image.

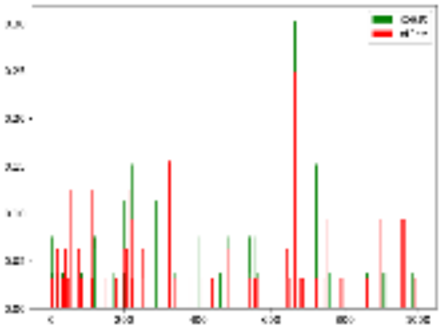

Figure 3. Histogram of coast and office image

5. After the process, each image will be expressed to a k length vector. These vectors are then be used to train the classifier, in this run, we selected linear SVM classifier to train our data. For predicting image, the histogram for input image is required to be generated for feeding classifier.

The result is much more improved compared with run 1. We benchmarked the classifier with different k value and patch size. With these parameters, we also tested the classifier with different image size to see whether sample image size will influence the result. The following image shows the benchmark result.

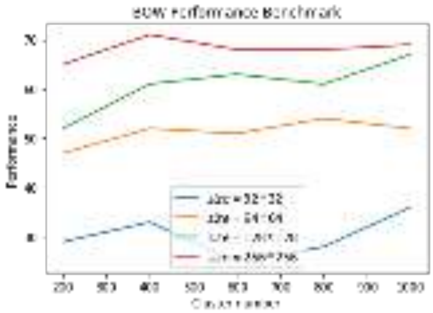

Figure 4. Dense sampled feature performance

From the benchmark, it can be seen that, the classifier has a better performance with bigger image size. The number of clusters k did not influence much on predict accuracy.

Best result is achieved when non-resized image is sampled with patch size 8x8 and step of 4, k value of k-means clustering is 800. The following table shows the test result for each scene category.

Scene	Accuracy	Scene	Accuracy
Forest	90%	Bedroom	50%
Office	85%	Highway	85%
Coast	100%	Inside city	50%
Tall Building	90%	Industrial	55%
Street	70%	Livingroom	50%
Suburb	85%	Mountain	85%
Kitchen	50%	Open country	55%
Store	85%	**Total**	**72.3%**

Figure 5. Dense patch 8x8 step=4, k = 800

It can be seen that, the test result is significantly improved with this approach, the classifier still maintains a high classify accuracy on outdoor scene, the accuracy is pretty high on coast, forest, mountain and highway. However, more time will be consumed with this approach, about 4000 patches are to be generated for an image with resolution of 300x250. Since the histogram is still desired to predict an image, the patches still need to be extracted from the image to be predicted.

Resizing images to lower resolution will increase the speed for patch sampling but the accuracy will also be influenced. So, this approach is not practical for real time applications.

5 Technical Approach for run 3

5.1 SIFT feature classifier

This approach is an improvement of run 2, instead of sampling dense patches, we extract the SIFT features from images, then we followed the same pattern in run 2, using bag of words model, classifier is trained based on the new histogram from each image. [1]

SIFT feature has the following advantages:

- It is invariant to the rotation, scale and brightness of an image. It also has a high dependence on illumination changes.
- Even if these is limited objects in the images, many SIFT feature vectors can still be generated.
- Solving the SIFT features on an image is really fast, it can be used in real time detection.

Each SIFT feature in an image can be expressed by a 128-length vector, also called SIFT descriptor.

To eliminate the influence of illumination, the feature vector is then be normalized using the equation:

$$feature_{normalized} = \frac{feature}{\sqrt{\sum_{j=1}^{128}(feature_j)^2}}$$

- After the normalization, a threshold of 0.2 is applied to the vector to eliminate the larger gradient of the direction and non-linear illumination.

We also used linear SVM classifiers to train and predict. We had multiple trials with normalized SIFT feature vectors and unnormalized SIFT feature vectors, the test accuracy when using unnormalized SIFT vector is always better in this case.

The following figure shows the benchmark of SIFT approach, all images are not resized. It can be seen that the cluster number k does not affect predict accuracy much.

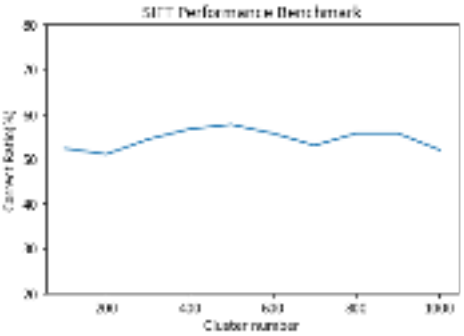

Figure 6. Benchmark on SIFT without image resize

Using SIFT features instead of dense sampling features significantly increase the speed for scene classification, but the predict accuracy is not satisfying.

5.2 Convolutional neural network

CNN is another popular approach for scene classification. We first worked out a network with 2 convolutional layers and 2 fully connected layers. The input image is resized to 32x32. For each epoch of training, the accuracy of model is calculated using our hand tagger test set, the accuracy was stabled to 46% which is not a satisfying result.

While VGGNet is a neural network that performed very well and got a high score on LLSVRC 2014. So, we determined to use VGG16 network for classification [2]
.

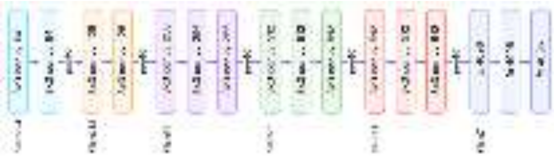

Figure 7. VGG16 layer summary

The main steps of this approach are:

1. The Output layer of origin VGG16 network is modified to 15 for our 15 categories classification.
2. Each image in the training set is resized to 224x224 with duplicated 3 channels.
3. The label for each image is a 15-length vector in which the label index element is 0 and all zeros for others.
4. The learning rate is set to 0.0001 and losses are calculated by using cross-entropy.
5. The images were feed into the model without any normalization.
6. The model was trained for 100 epochs.

The test result is not satisfying, during the training, the accuracy calculated by using a batch of training set kept increasing to 100%. After the training process, we tested the model with our hand tagged testing set, the VGG16 network can only achieve an accuracy of 57%.

We normalized the image by dividing the original image by 255 to make each pixel to range of 0 to 1 and retrained the model again, but the accuracy is not improved. The following figure shows a more detailed result.

Scene	Accuracy	Scene	Accuracy
Forest	75%	Bedroom	40%
Office	75%	Highway	90%
Coast	70%	Inside city	25%
Tall Building	60%	Industrial	40%
Street	80%	Livingroom	20%
Suburb	70%	Mountain	40%
Kitchen	30%	Open country	75%
Store	75%	**Total**	**57.6%**

There are significant drawbacks for training the network with grey-scaled image, the color information is lost for training set, the following figure shows the image of coast and open country. In our testing, classifier has a trend to predict the coast as open country, it is because from the grey scale image the two scenes look similar in overall layout.

Coast

Open country

Another reason for the low performance in VGG16 is that the number of training set is limited, the network don't have enough training set. If more training set is given, the performance of neural network would improve.

5.3 GIST Feature Classifier

Since the testing result for using SIFT features and CNN is not satisfying compared with the result achieved in run 2. We then investigate the performance for using GIST feature to classify images.

The idea of GIST descriptor is to develop a low dimension representation of the scene. [1] The GIST feature of an image is 960-length vector, which describes the following aspects of that image:

1. Degree of naturalness: if an image contains horizontal or vertical line, it shows a potential that the scene is artificial processed.
2. Degree of openness, scenes of forest, mountain, inside city are close fields, while scenes of coast, highway are open fields.
3. Degree of roughness, the particle size in the image
4. Degree of Expansion, buildings have a low degree expansion while long street has higher
5. Degree of Ruggedness, natural scene will have a higher degree of ruggedness.

The main steps of this approach are:

1. Extract GIST feature from each image as the feature vector of that image, the feature vector is of 960 length.
2. Linear SVM classifier is trained by using GIST features.
3. When predict, the GIST feature for target image should be exacted and feed to trained SVM classifier.

We trained the classifier and test the accuracy with different resized images. The following figure shows the predict accuracy. It can be seen that the maximum accuracy is achieved when each image is resized to 160 x 160.

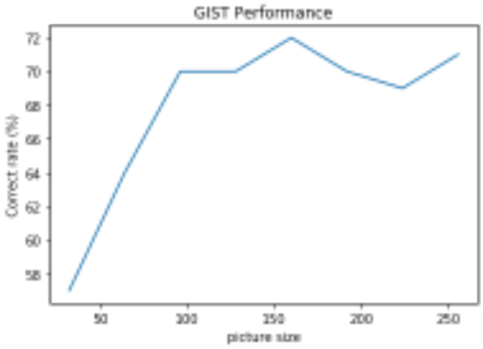

Figure 8. GIST Performance benchmark

The following table shows a more detailed test result:

Scene	Accuracy	Scene	Accuracy
Forest	90%	Bedroom	45%
Office	80%	Highway	90%
Coast	80%	Inside city	65%
Tall Building	80%	Industrial	50%
Street	75%	Livingroom	70%
Suburb	85%	Mountain	60%
Kitchen	50%	Open country	60%
Store	90%	**Total**	**71.3%**

Figure 9. Test result using GIST feature

A relatively high predict accuracy is achieved by using GIST feature classifier. This solution maintained a high accuracy and high performance. It only takes about an average of 0.2sec to extract GIST feature from an image. And is not required to perform clustering for generate bag of words histograms. So that this solution is practical for real time application.

5.4 <u>Other approaches</u>

In all the approaches discussed above we used linear SVM for classification task. We also tried Naïve Bayes classifiers, but liner SVM classifier still outperformed Bayes classifiers. The table below shows a more detailed result.

Classifier	Predict accuracy
Linear SVM	57%
Gaussian Naïve Bayes	27%
Multinomial Naive Bayes	52%
Bernoulli Naive Bayes	46%

Figure 10. Performance for different classifier, k=800, without image resize

6 Conclusion

In this report, we investigated multiple approaches for scene classifications.

For run 1, we used tiny image features with k-nearest neighbor to perform classification, this approach is of high predict performance but very low predict accuracy, not practical enough.

For run 2, we dense sampled each image to generate a bag of word. We achieved a relatively high predict accuracy of 72%. But the drawback of this approach is significant, it takes a long time to dense sample each image. So, it is not practical for real time application.

For run3, we extracted SIFT features and GIST features from images to perform classification. SIFT approach is of high performance but lower accuracy than. GIST feature classifier is the best solution in this case, it achieved high predict accuracy while maintaining a high performance, practical for real time applications. However, we do not recommend the CNN approach, this approach requires more training samples to improve accuracy.

7 Contribution Statement

We implemented each solution separately. Yigang Zhou build the prototype of each solution and examined the feasibility. Ziyang Wei rewrite all the solutions with a benchmark implemented. Ziyang Wei benchmarked all the solution and produced figure used in this report.

For run 3, we all implanted SIFT feature classifier, Yigang Zhou implemented and benchmarked VGG16 classifier and Ziyang Wei implemented and benchmarked GIST feature classifier.

Yigang Zhou wrote this report.

8 Reference

[1] C. R. D. L. F. J. W. C. B. Gabriella Csurka, "Visual Categorization with Bags of Keypoints," in ECCV International Workshop on Statistical Learning in Computer Vision, 2004.

[2] K. a. Z. A. Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in ICLR, 2015.

[3] M. J. H. S. H. A. L. &. S. C. Douze, "Evaluation of gist descriptors for web-scale image search," in Proceedings of the ACM International Conference on Image and Video Retrieval, ACM, 2009, p. 19.

```python
# COMP6223_CW3
# k_neighbor
# Created by Yigang Zhou on 2018/12/3.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.
# run 1 k_neighbor

from os import listdir
from os.path import join
from PIL import Image
import numpy as np
import util


data, labels, paths, label_names = util.load_tagged_set(img_size=16, flatten=True)
data = data/255


# Euclidean distance
def get_distance(img1_flat, img2_flat):
    '''
    calculate Euclidean distance between two image vectors
    :param img1_flat: tiny image feature 1
    :param img2_flat: tiny image feature 2
    :return:
    '''
    img1_flat = np.asmatrix(img1_flat)
    img2_flat = np.asmatrix(img2_flat)
    sub = img1_flat - img2_flat
    dis = (sub * sub.T)
    return float(dis)

# k neighbour
def k_neighbor(img, k=5):
    '''
    :param img: tiny image vector
    :param k: number of neighbors
    :return:
    '''
    distances = []

    for each_training_data in data:
        distance = get_distance(each_training_data, img)
        distances.append(distance)

    idxs = np.asanyarray(distances).argsort()[:k]

    k_nearest_label = []
    for each in idxs:
        k_nearest_label.append(labels[each])

    label_index = max(k_nearest_label, key=k_nearest_label.count)

    return label_index
```

```
# COMP6223_CW3
# sift.py
# Created by Yigang Zhou on 2018/12/6.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.


import cv2 as cv


def get_sift_features(image):
    sift = cv.xfeatures2d.SIFT_create()
    kp, des = sift.detectAndCompute(image, None)
    return des
```

```
# COMP6223_CW3
# CNN_test3
# Created by Yigang Zhou on 2018/12/2.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.


import tensorflow as tf
from os import listdir
from os.path import join, isdir
from PIL import Image
import numpy as np


def load_training_set():
    path_training = "data/training"

    data = []
    labels = []
    paths = []
    label_names = []

    # list training set images
    label_num = 0
    for each_label in sorted(listdir(path_training)):
        if isdir(each_label):
            print("Loading", each_label)
            label_names.append(each_label)
            label_path = join(path_training, each_label)
            for each_image in sorted(listdir(label_path)):
                if each_image.endswith(".jpg"):
                    # print(label_name, each_image)
                    path = join(path_training, each_label, each_image)
                    img = Image.open(path)
                    img = img.resize((64, 64), Image.ANTIALIAS)
                    img = np.array(img)
                    img_flat = img.flatten().reshape((4096))/255
                    data.append(img_flat)
                    paths.append(path)

                    temp = np.zeros(15)
                    temp[label_num] = 1

                    labels.append(temp)
            label_num += 1
    data = np.asanyarray(data)
    labels = np.asanyarray(labels)
    return data,labels,paths,label_names


data,labels,paths,label_names = load_training_set()
num_classes = 15

print(label_names)


def compute_accuracy(v_xs, v_ys):
    global prediction
    y_pre = sess.run(prediction, feed_dict={xs: v_xs, keep_prob: 1})
    correct_prediction = tf.equal(tf.argmax(y_pre,1), tf.argmax(v_ys,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    result = sess.run(accuracy, feed_dict={xs: v_xs, ys: v_ys, keep_prob: 1})
    return result


def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


def conv2d(x, W):
    # stride [1, x_movement, y_movement, 1]
    # Must have strides[0] = strides[3] = 1
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x2(x):
    # stride [1, x_movement, y_movement, 1]
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')


# define placeholder for inputs to network
xs = tf.placeholder(tf.float32, [None, 4096])    # 28x28
ys = tf.placeholder(tf.float32, [None, 15])
keep_prob = tf.placeholder(tf.float32)
x_image = tf.reshape(xs, [-1, 64, 64, 1])
# print(x_image.shape)  # [n_samples, 28,28,1]


# conv1 layer
W_conv1 = weight_variable([5,5,1,32]) # patch 5x5, in size 1, out size 32
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # output size 28x28x32
h_pool1 = max_pool_2x2(h_conv1)                                         # output size 14x
14x32


# conv2 layer
W_conv2 = weight_variable([5,5,32,64]) # patch 5x5, in size 32, out size 64
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # output size 14x14x64
h_pool2 = max_pool_2x2(h_conv2)                                         # output size 7x7
x64

# full connect 1 layer
W_fc1 = weight_variable([16*16*64, 1024])
b_fc1 = bias_variable([1024])
# [n_samples, 7, 7, 64] ->> [n_samples, 7*7*64]
h_pool2_flat = tf.reshape(h_pool2, [-1, 16*16*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# full connect 2 layer
W_fc2 = weight_variable([1024, 15])
b_fc2 = bias_variable([15])

prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)


# the error between prediction and real data
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                              reduction_indices=[1]))       # loss
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

sess = tf.Session()

init = tf.global_variables_initializer()
sess.run(init)


def test_accurancy():
    # path_test = "data/testing"

    data_test = []
    labels = []
    label_names_test = []
    test_paths = []

    path_testing = "data/testing_tagged"

    # list training set images
    label_num = 0
    for each_label in sorted(listdir(path_testing)):
        if '.DS_Store' not in each_label:
            label_names_test.append(each_label)
            label_path = join(path_testing, each_label)
            for each_image in sorted(listdir(label_path)):
                if each_image.endswith(".jpg"):
                    # print(label_name, each_image)
                    path = join(path_testing, each_label, each_image)
                    img = Image.open(path)
                    img = img.resize((64, 64), Image.ANTIALIAS)
                    img = np.array(img)
                    img_flat = img.flatten().reshape((4096)) / 255
                    data_test.append(img_flat)
                    test_paths.append(path)
                    labels.append(label_num)
            label_num += 1

    data = np.asanyarray(data_test)
    labels = np.asanyarray(labels)

    data_test = np.asanyarray(data_test)

    y_pre = sess.run(prediction, feed_dict={xs: data_test, keep_prob: 1})

    count = 0
    pass_num = 0
    for each in y_pre:
        index_predict = np.argmax(each)
        index_actual = labels[count]
        predicted_label = label_names[index_predict]
        actual_label = label_names_test[index_actual]

        if predicted_label == actual_label:
            pass_num += 1

        count += 1

    return pass_num / count


for i in range(1000):

    sess.run(train_step, feed_dict={xs: data, ys: labels, keep_prob: 1})

    print("step = {}".format(i), "accurancy", test_accurancy())
    print()

model_path = "model3/"
saver = tf.train.Saver()
saver.save(sess, model_path)
```

```
# COMP6223_CW3
# lbp_classifier.py
# Created by Yigang Zhou on 2018/12/9.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.

# import the necessary packages
from skimage import feature
import numpy as np
import util

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import NuSVC
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB


class LocalBinaryPatterns:
    def __init__(self, numPoints, radius):
        # store the number of points and radius
        self.numPoints = numPoints
        self.radius = radius

    def describe(self, image, eps=1e-7):
        # compute the Local Binary Pattern representation
        # of the image, and then use the LBP representation
        # to build the histogram of patterns
        lbp = feature.local_binary_pattern(image, self.numPoints,
                                           self.radius, method="uniform")
        (hist, _) = np.histogram(lbp.ravel(),
                                 bins=np.arange(0, self.numPoints + 3),
                                 range=(0, self.numPoints + 2))

        # normalize the histogram
        hist = hist.astype("float")
        hist /= (hist.sum() + eps)

        # return the histogram of Local Binary Patterns
        return hist


def build_histograms(images, lbp):
    histograms = []
    for i, image in enumerate(images):
        histograms.append(lbp.describe(image))
    return np.asarray(histograms)

#
# data, labels, paths, label_names = util.load_tagged_set(path_training="data/training",
img_size=None)
#
# lbp = LocalBinaryPatterns(128, 8)
# histograms = build_histograms(data, lbp)
#
# svm_classifier = NuSVC(gamma='scale')
# svm_classifier.fit(histograms, labels)
#
# data_test, labels_test, paths_test, label_names_test = util.load_tagged_set(path_traini
ng="data/testing_tagged", img_size=None)
# histograms_test = build_histograms(data_test, lbp)
# print("test shape", histograms_test.shape)
# predict_result = svm_classifier.predict(histograms_test)
# #
#
# count = 0
# pass_num = 0
#
# for each in predict_result:
#     label_actual = label_names_test[labels_test[count]]
#     label_predict = label_names[each]
#     print(paths_test[count], label_actual, label_predict)
#     if label_actual == label_predict:
#         pass_num += 1
#     count += 1
# print(pass_num/count)
# COMP6223_CW3
# CNN_test.py
# Created by Yigang Zhou on 2018/12/3.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.


import tensorflow as tf
from os import listdir
from os.path import join
from PIL import Image
import numpy as np

label_names = ['Coast', 'Forest', 'Highway', 'Insidecity', 'Mountain', 'Office', 'OpenCou
ntry', 'Street', 'Suburb', 'TallBuilding', 'bedroom', 'industrial', 'kitchen', 'livingroo
m', 'store']


def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


def conv2d(x, W):
    # stride [1, x_movement, y_movement, 1]
    # Must have strides[0] = strides[3] = 1
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x2(x):
    # stride [1, x_movement, y_movement, 1]
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

# define placeholder for inputs to network
xs = tf.placeholder(tf.float32, [None, 1024])   # 28x28
ys = tf.placeholder(tf.float32, [None, 15])
keep_prob = tf.placeholder(tf.float32)
x_image = tf.reshape(xs, [-1, 32, 32, 1])
# print(x_image.shape)  # [n_samples, 28,28,1]

## conv1 layer ##
W_conv1 = weight_variable([5,5,1,32]) # patch 5x5, in size 1, out size 32
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # output size 28x28x32
h_pool1 = max_pool_2x2(h_conv1)                                         # output size 14x
14x32

## conv2 layer ##
W_conv2 = weight_variable([5,5,32,64]) # patch 5x5, in size 32, out size 64
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # output size 14x14x64
h_pool2 = max_pool_2x2(h_conv2)                                         # output size 7x7
x64

## fc1 layer ##
W_fc1 = weight_variable([8*8*64, 1024])
b_fc1 = bias_variable([1024])
# [n_samples, 7, 7, 64] ->> [n_samples, 7*7*64]
h_pool2_flat = tf.reshape(h_pool2, [-1, 8*8*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

## fc2 layer ##
W_fc2 = weight_variable([1024, 15])
b_fc2 = bias_variable([15])
```

```
prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)


# the error between prediction and real data
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                              reduction_indices=[1]))       # loss
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)


sess = tf.Session()

init = tf.global_variables_initializer()
sess.run(init)

model_path = "model1/"
saver = tf.train.Saver()
saver.restore(sess, model_path)





print("Begin test")
# path_test = "data/testing"


data_test = []
labels = []
label_names_test = []

test_paths = []

path_testing = "data/testing_tagged"

# list training set images
label_num = 0
for each_label in sorted(listdir(path_testing)):
    if '.DS_Store' not in each_label:
        print("Loading", each_label)
        label_names_test.append(each_label)
        label_path = join(path_testing, each_label)
        for each_image in sorted(listdir(label_path)):
            if each_image.endswith(".jpg"):
                # print(label_name, each_image)
                path = join(path_testing, each_label, each_image)
                img = Image.open(path)
                img = img.resize((32, 32), Image.ANTIALIAS)
                img = np.array(img)
                img_flat = img.flatten().reshape((1024))/255
                data_test.append(img_flat)
                test_paths.append(path)
                labels.append(label_num)
        label_num += 1

data = np.asanyarray(data_test)
labels = np.asanyarray(labels)

data_test = np.asanyarray(data_test)
print(data_test.shape)

y_pre = sess.run(prediction, feed_dict={xs: data_test, keep_prob: 1})

count = 0
pass_num = 0
for each in y_pre:
    index_predict = np.argmax(each)
    index_actual = labels[count]
    predicted_label = label_names[index_predict]
    actual_label = label_names_test[index_actual]
    if predicted_label == actual_label:
        pass_num += 1

    print(test_paths[count], predicted_label, actual_label)
    count += 1

print(pass_num/count)
```

```
# COMP6223_CW3
# util.py
# Created by Yigang Zhou on 2018/12/6.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.

from os import listdir
from os.path import join, isdir
from PIL import Image
import numpy as np


def load_tagged_set(path_dataset="data/training", img_size=256, flatten=False):
    '''
    :param path_dataset:
    :param img_size: resize image, no resize when None
    :param flatten: whether to convert 2d arr to a 1d vector
    :return: data, image arrays
             labels, label of the image, a number
             paths, path of the image
             label_names, actual label name
    '''
    data = []
    labels = []
    paths = []
    label_names = []

    # list training set images
    label_num = 0
    for each_label in sorted(listdir(path_dataset)):
        if isdir(join(path_dataset, each_label)):
            print("Load data set", each_label)
            label_names.append(each_label)
            label_path = join(path_dataset, each_label)
            for each_image in sorted(listdir(label_path)):
                if each_image.endswith(".jpg"):
                    # print(label_name, each_image)
                    path = join(path_dataset, each_label, each_image)
                    img = Image.open(path)
                    if img_size is not None:
                        img = img.resize((img_size, img_size), Image.ANTIALIAS)
                    img = np.array(img)

                    if flatten:
                        img = img.flatten().reshape((img_size*img_size))

                    if img.shape == (img_size, img_size, 3):
                        img = img[:, :, 0]

                    data.append(img)
                    paths.append(path)

                    labels.append(label_num)
            label_num += 1

    data = np.asanyarray(data)
    labels = np.asanyarray(labels)
    return data, labels, paths, label_names



def evaluate_model(paths, actual_labels, predicted_labels, logger=None):
    '''
    evaluate model performance
    :param paths: paths of the test images
    :param actual_labels: actual label name (not label index!)
    :param predicted_labels: predicted label name (not label index!)
    :return:
    '''
    label_counter = {} #counts the total label num of testing set
    label_correct_counter = {} #counts the total correct label num of testing set

    for i, path in enumerate(paths):
        predicted_label = predicted_labels[i]
        actual_label = actual_labels[i]

        # prevent label_correct_counter dont have that key if a all labels are failed to
predict
        if actual_label not in label_correct_counter:
            label_correct_counter[actual_label] = 0

        if actual_label not in label_counter:
            label_counter[actual_label] = 1
        else:
            label_counter[actual_label] += 1

        if predicted_label == actual_label:
            if predicted_label not in label_correct_counter:
                label_correct_counter[predicted_label] = 1
            else:
                label_correct_counter[predicted_label] += 1

    total = 0
    hit = 0
    for each_label in label_counter:
        accuracy = label_correct_counter[each_label] / label_counter[each_label]
        total += label_counter[each_label]
        hit += label_correct_counter[each_label]
        logger.log(each_label,":", label_correct_counter[each_label],"out of",label_count
er[each_label], accuracy)
    total_accuracy = hit/total
    print(total_accuracy)
    logger.log("Total accuracy", total_accuracy)
    logger.print()
    logger.save("log/")
```

```
# COMP6223_CW3
# __init__.py
# Created by Yigang Zhou on 2018/12/10.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.
```

```python
# COMP6223_CW3
# logger.py
# Created by Yigang Zhou on 2018-12-12.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.


class Logger(object):
    '''
    a class to log outputs, and save to file
    '''

    def __init__(self, name):
        self.logs = []
        self.name = name

    def log(self, *argv):
        '''
        add new log to logger
        :param argv:
        :return:
        '''
        log = ""
        for each_log in argv:
            log += str(each_log) + " "
        self.logs.append(log)

    def print(self):
        print("Log:",self.name)
        for each in self.logs:
            print(each)

    def save(self, path):
        '''
        save log to local file
        :param path:
        :return:
        '''
        file_path = path + self.name + ".txt"
        file_path = file_path.replace(" ","_")
        f = open(file_path, "w")
        f.write(self.name+"\n")
        for each in self.logs:
            f.write(each+"\n")
        f.close()
        print("Log file saved to", file_path)
```

```python
# COMP6223_CW3
# run2
# Created by Yigang Zhou on 2018/12/3.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.
# Dense patch sample + SVM classifier


import numpy as np
from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
import pickle
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
import util
from logger import Logger

def normalize(vector):
    '''
    zero mean of a vector
    :param vector: input vector
    :return:
    '''
    zero_vector = np.zeros(shape=vector.shape)
    max = np.max(vector)
    min = np.min(vector)
    if max - min == 0:
        return zero_vector
    else:
        zero_mean_vector = (vector - np.mean(vector))/(max - min)
        return zero_mean_vector


def get_patches(img, patch_size, step):
    '''
    generate patches of a imag
    :param img: input image
    :param patch_size: size of each square patch
    :param step: step size in x and y direction
    :return: a array with flattened patched
    '''

    patches = []

    h, w = img.shape

    # pre calculate the origin of each patch
    Xs = np.arange(0, w, step)
    Ys = np.arange(0, h, step)

    for x in Xs:
        for y in Ys:
            if x+patch_size <= w and y+patch_size <= h: #prevent over flow
                patch = img[y: y + patch_size, x: x + patch_size]
                patch_flat = np.asarray(patch).flatten()
                #zero means normalize
                patch_normalized = normalize(patch_flat)
                patches.append(patch_normalized)

    return np.asanyarray(patches)


def stack_patches(images, patch_size, step):
    '''
    get all patches from all training image
    :param images: images with one channel
    :param patch_size:  patch size
    :param step: step
    :return: an array with all flatten patches
    '''
    all_patches = []
    print("Generating patches from all images")
    for each_image in images:
        for each_patch in get_patches(each_image, patch_size, step):
            all_patches.append(each_patch)


    return np.asanyarray(all_patches)


def train_kmeans(n_clusters, patches, save=False):
    '''

    :param n_clusters: k
    :param patches: patches extracted from all images
    :param save: whether to save model
    :return:
    '''
    print("Training Kmeans...")
    # kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(all_patches)
    predictor = MiniBatchKMeans(n_clusters=n_clusters, init_size=3*n_clusters).fit(patche
s)
    if save:
        filename = 'kmeans_model_run2_.sav'
        pickle.dump(predictor, open(filename, 'wb'))

    # kmeans = pickle.load(open(filename, 'rb'))
    return predictor


def build_histograms(images,k, kmeans_predictor, patch_size, step):
    '''
    :param images: image set
    :param k: cluster number
    :param kmeans_predictor: predictor
    :param patch_size: patch size
    :param step: step
    :return:
    '''
    histograms = []

    print("building histograms")
    for each_image in images:
        histogram = np.zeros(k)
        patches = get_patches(each_image, patch_size, step)

        index_array = kmeans_predictor.predict(patches)

        for each in index_array:
            histogram[each] += 1

        histograms.append(histogram)

    return np.asarray(histograms)


def train_and_evaluate_model(n_clusters=800, patch_size=8, step=4, image_size=None):

    data, labels, paths, label_names = util.load_tagged_set(img_size=image_size)

    patches = stack_patches(images=data, patch_size=patch_size, step=step)
    print(patches.shape)
    predictor = train_kmeans(n_clusters=n_clusters, patches=patches)

    histograms = build_histograms(images=data,
                                  k=n_clusters,
                                  kmeans_predictor=predictor,
                                  patch_size=patch_size,
                                  step=step)
    classifier = SVC(kernel='linear').fit(histograms, labels)

    # load tagged training set
    data_test, labels_test, paths_test, label_names_test = util.load_tagged_set("data/testing_tagged", img_size=image_size)

    # histograms of training set
    histograms_test = build_histograms(images=data_test,
                                       k=n_clusters,
                                       kmeans_predictor=predictor,
                                       patch_size=patch_size,
                                       step=step)

    labels_predicted = classifier.predict(histograms_test)

    # get actual predicted label name: 1 -> Bedroom. 2->Coast....
    predicted_label_names = []
    for each in labels_predicted:
        predicted_label_names.append(label_names[each])
    predicted_label_names = np.asanyarray(predicted_label_names)

    actual_label_names = []
    for each in labels_test:
        actual_label_names.append(label_names_test[each])
    actual_label_names = np.asanyarray(actual_label_names)

    # log
    log_name = 'run2 k={} patch_size={} step={} image_size={}'.format(n_clusters, patch_size, step, image_size)
    logger = Logger(log_name)

    util.evaluate_model(paths_test, actual_label_names, predicted_label_names, logger)


if __name__ == '__main__':
    train_and_evaluate_model(patch_size=8, step=4)
```

```
# COMP6223_CW3
# vgg16.py
# Created by Yigang Zhou on 2018/12/7.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.
# VGG16 CNN
import keras
from keras.models import Sequential
from keras.models import Model
from keras.models import load_model
from keras.layers import Dense
import keras.applications.vgg16 as VGG
from keras.optimizers import Adam
import util
import numpy as np
from logger import Logger


def concentrate_gray2rgb(images_gray):
    '''
    reshape image to (N, 224,224,3)
    :param images_gray:
    :return:
    '''

    rgbs = []

    for each_image in images_gray:
        rgbs.append([each_image, each_image,each_image])
    return np.asarray(rgbs).reshape(images_gray.shape[0], images_gray.shape[1], images_gr
ay.shape[2], 3)


def convert_label(labels_train):
    '''
    convert label to 15 lenghth vector
    1 -> [1,0,0,0,0,0,0,0.....]
    2 -> [0,1,0,0,0,0,0,0....]
    :param labels_train:
    :return:
    '''
    labels = []
    for each in labels_train:
        label = np.zeros(15)
        label[each] = 1
        labels.append(label)
    return np.asarray(labels)


def train_vgg16_model(labels, data):
    '''
    train vgg16 model
    :param labels: converted labels (N,15)
    :param data: converted images (N,224,224,3)
    :return:
    '''
    # load vgg model
    vgg16_model = VGG.VGG16(weights=None, include_top=True)

    # define new output layer  and remove the output layer from original vgg16
    x = Dense(15, activation='softmax', name='predictions')(vgg16_model.layers[-2].output
)

    # define new model
    model = Model(input=vgg16_model.input, output=x)

    model.summary()

    # learning rate of 0.00001
    model.compile(Adam(lr=0.00001), loss='categorical_crossentropy', metrics=['accuracy']
)
```

```
    # model = load_model('models/keras_vgg_model_30.h5')

    model.fit(data, labels, verbose=2, epochs=30)

    model.save('models/keras_model.h5')  # creates a HDF5 file 'my_model.h5'

    return model


def load_training_set_and_train():
    '''
    load training set and train the model
    :return:
    '''
    data, labels, paths, label_names = util.load_tagged_set(path_dataset="data/training",
 img_size=224)

    # reshape to fit the network
    labels = convert_label(labels)
    data = concentrate_gray2rgb(data)

    print(data.shape)
    print(labels.shape)

    model = train_vgg16_model(labels, data)

    evaluate_model(model, label_names)

    return model


def evaluate_model(model, label_names):
    '''

    :param model: vgg model
    :param label_names: actual names of label, not label number
    :return:
    '''

    logger = Logger('VGG_16')

    data_test, labels_test, paths_test, label_names_test = util.load_tagged_set(path_data
set="data/testing_tagged", img_size=224)

    data_test = concentrate_gray2rgb(data_test)

    predict = model.predict(data_test)

    correct_count = 0
    test_count = data_test.shape[0]
    label_count = np.zeros(test_count)
    label_correct_count = np.zeros(test_count)

    for index, each in enumerate(predict):
        label_predict = label_names[np.argmax(each)]
        label_actual = label_names_test[labels_test[index]]
        label_count[labels_test[index]] += 1
        logger.log(paths_test[index], label_predict, label_actual)
        if label_predict == label_actual:
            correct_count += 1
            label_correct_count[labels_test[index]] += 1

    for index, each in enumerate(label_names_test):
        a = label_correct_count[index]
        b = label_count[index]
        accuracy = a/b
        logger.log(each, accuracy)

    accuracy_total = correct_count/test_count
    logger.log("Total accuracy", accuracy_total)
    logger.print()
    logger.save('log/')


if __name__ == '__main__':
    # train from scratch
    # load_training_set_and_train()

    # load pre-trained model and predict
    data, labels, paths, label_names = util.load_tagged_set(path_dataset="data/training",
 img_size=224)
    model = load_model('models/keras_vgg_model_100.h5')
    evaluate_model(model, label_names)
```

```python
# COMP6223_CW3
# run3
# Created by Yigang Zhou on 2018/12/6.
# Copyright Â© 2018 Yigang Zhou. All rights reserved.
# SIFT + SVM classifier

import numpy as np
import util
import sift
import pickle
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import normalize

from sklearn.svm import NuSVC
from sklearn.svm import SVC
from logger import Logger
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB


def stack_sift_features(data):
    print("stacking sift features")
    sift_features = []

    for each in data:

        features = sift.get_sift_features(each) # n x 128
        if features is not None:
            for feature in features:

                sift_features.append(feature)

    return np.asarray(sift_features)


def normalize_sift(sift_feature):
    norm1 = sift_feature / np.sqrt(np.sum(np.square(sift_feature)))
    for i, each in enumerate(norm1):
        if each > 0.2:
            norm1[i] = 0.2

    return norm1


def train_k_means(model_path=None, k=1000, features=None):
    if model_path is None:
        predictor = MiniBatchKMeans(n_clusters=k, init_size=k*3).fit(features)
        filename = 'kmeans_model.sav'
        pickle.dump(predictor, open(filename, 'wb'))
        print("Model saved to ", filename)
        return predictor
    else:
        print("Loading kmeans model from ", model_path)
        return pickle.load(open(str(model_path), 'rb'))


def build_histograms(images,k, kmeans_predictor):
    histograms = []

    print("building histograms")
    for each_img in images:
        histogram = np.zeros(k)
        sift_features = sift.get_sift_features(each_img)

        if sift_features is not None:

            index_array = kmeans_predictor.predict(sift_features)

            for each in index_array:
                histogram[each] += 1

            histograms.append(histogram/np.sum(histogram))
        else:
            histograms.append(histogram)
    #L2 norm
    return np.asarray(normalize(histograms))


def train_svm_classifier(model_path=None, histograms=None, labels=None):
    if model_path is None:
        print("Training SVM")
        # classifier = OneVsRestClassifier(LinearSVC(random_state=0, max_iter=10000000)).
fit(histograms, labels)
        classifier = SVC(kernel='linear').fit(histograms, labels)
        # classifier = NuSVC().fit(histograms, labels)
        filename_classifier = 'svm_classifier_2.sav'
        pickle.dump(classifier, open(filename_classifier, 'wb'))
        print("SVM model saved to ", filename_classifier)
        return classifier
    else:
        print("Loading SVM classifier from", model_path)
        classifier = pickle.load(open(model_path, 'rb'))
        return classifier


def train_and_evaluate_model(k=500, classifier='linear_svm', img_size=None):
    data, labels, paths, label_names = util.load_tagged_set(path_dataset="data/training",
 img_size=img_size)

    all_sift_features = stack_sift_features(data)
    print(all_sift_features.shape)

    kmeans_predictor = train_k_means(model_path=None, k=k, features=all_sift_features)

    histograms = build_histograms(data, k, kmeans_predictor)

    if classifier == 'linear_svm':
        train_classifier = train_svm_classifier(model_path=None, histograms=histograms, l
abels=labels)
    elif classifier == 'gaussian_naive_bayes':
        train_classifier = GaussianNB().fit(histograms, labels)
    elif classifier == 'multinomial_naive_bayes':
        train_classifier = MultinomialNB().fit(histograms, labels)
    elif classifier == 'bernoulli_naive_bayes':
        train_classifier = BernoulliNB().fit(histograms, labels)

    data_test, labels_test, paths_test, label_names_test = util.load_tagged_set(path_data
set="data/testing_tagged", img_size=img_size)

    histograms_test = build_histograms(data_test, k, kmeans_predictor)

    print("test shape", histograms_test.shape)
    labels_predicted = train_classifier.predict(histograms_test)

    predicted_label_names = []
    for each in labels_predicted:
        predicted_label_names.append(label_names[each])
    predicted_label_names = np.asanyarray(predicted_label_names)

    actual_label_names = []
    for each in labels_test:
        actual_label_names.append(label_names_test[each])
    actual_label_names = np.asanyarray(actual_label_names)

    log_name = 'run3 k={} classifier={} image_size={}'.format(k, classifier, img_size)
    logger = Logger(log_name)
    util.evaluate_model(paths_test, actual_label_names, predicted_label_names, logger)
```

```
if __name__ == '__main__':

    classfiers = ['linear_svm', 'gaussian_naive_bayes', 'multinomial_naive_bayes', 'berno
ulli_naive_bayes']

    for each in classfiers:
        train_and_evaluate_model(k=800, img_size=None, classifier=each)
```

```python
# -*- coding: utf-8 -*-
# @Time    : 06/12/2018 15:56
# @Author  : weiziyang
# @FileName: sift.py
# @Software: PyCharm
import os

import cv2
from tqdm import tqdm
import numpy as np
from sklearn.cluster import KMeans, MiniBatchKMeans
from sklearn.svm import SVC

from base import Base, count_time


class SIFT(Base):
    def __init__(self, cluster_num, mini_patch=True, force_generate_again=False):
        super().__init__(force_generate_again=force_generate_again)
        self.parameter_token = "{name}-Cluster_num:{cluster}Kmeans_type:{M}".format(
            name=self.__class__.__name__, cluster=cluster_num, M='MINI' if mini_patch else 'NORMAL')
        self.cluster_num = cluster_num
        self.mini_patch = mini_patch

        self.working_dir = os.path.join(self.data_dir, self.parameter_token)
        if not os.path.exists(self.working_dir):
            os.makedirs(self.working_dir)
        self.sift_features_list_file = os.path.join(self.data_dir, 'sift_features_list.pkl')
        self.k_means_file = os.path.join(self.working_dir, 'k_means.pkl')
        self.svm_file = os.path.join(self.working_dir, 'svm.pkl')

        self.sift_features_list = None
        self.sift_features = None
        self.k_means = None
        self.svm = None

    @count_time
    def train(self):
        self.sift_features_list = self.compute_obj(self.sift_features_list_file, self.generate_sift)
        self.sift_features = np.vstack(self.sift_features_list)
        self.k_means = self.compute_obj(self.k_means_file, self.compute_k_means)
        self.svm = self.compute_obj(self.svm_file, self.compute_svm)


    @count_time
    def generate_sift(self):
        sift = cv2.xfeatures2d.SIFT_create()
        sift_features_list = []
        with tqdm(total=self.total_num) as pbar:
            for category in self.categories:
                for image_name in self.images_name:
                    image_path = os.path.join(self.raw_training_dir, category, image_name)
                    image_data = cv2.imread(image_path)
                    kp, sift_feature = sift.detectAndCompute(image_data, None)
                    if sift_feature is None:
                        sift_feature = np.zeros((1, 128))
                    pbar.update(1)
                    sift_features_list.append(sift_feature)
        return sift_features_list

    @count_time
    def compute_k_means(self):
        if self.mini_patch:
            k_means = MiniBatchKMeans(n_clusters=self.cluster_num, init_size=3 * self.cluster_num).fit(self.sift_features)
        else:
            k_means = KMeans(n_clusters=self.cluster_num).fit(self.sift_features)
        return k_means

    @count_time
    def compute_svm(self):
        feature_matrix = np.zeros((self.total_num, self.cluster_num))
        index = 0
        with tqdm(total=self.total_num) as pbar:
            for sift_features in self.sift_features_list:
                feature_histogram = self.convert2visual_words(sift_features)
                feature_matrix[index, :] = feature_histogram
                index += 1
                pbar.update(1)
        svm = SVC(kernel='linear')
        svm.fit(feature_matrix, self.training_y)
        return svm

    def convert2visual_words(self, sift_features):
        feature_histogram = np.zeros(self.cluster_num)
        locations = self.k_means.predict(sift_features)
        for loc in locations:
            feature_histogram[loc] += 1
        return feature_histogram

    def predict(self, image):
        sift = cv2.xfeatures2d.SIFT_create()
        kp, sift_features = sift.detectAndCompute(image, None)
        feature_histogram = self.convert2visual_words(sift_features)
        prediction = self.svm.predict(np.atleast_2d(feature_histogram))
        return prediction


if __name__ == "__main__":
    sift = SIFT(cluster_num=600, mini_patch=True, force_generate_again=True)
    sift.train()
    sift.test_model()
```

```
# -*- coding: utf-8 -*-
# @Time    : 13/12/2018 12:44
# @Author  : weiziyang
# @FileName: run.py
# @Software: PyCharm

from K_neighbour import KNeighbour
from words_bag import WordsBag
from GIST import GIST

if __name__ == "__main__":
    with open('run_result/run1.txt', 'w') as f:
            run1 = KNeighbour(k=1, pic_size=8)
            run1.train()
            text = run1.predict_all()
            f.write(text)

    with open('run_result/run2.txt', 'w') as f:
        run2 = WordsBag(patch_gap=4, patch_size=8, cluster_num=400, output_patch_image=Fa
lse,
                        mini_patch=True, pic_size=256, force_generate_again=False)
        run2.train()
        text = run2.predict_all()
        f.write(text)

    with open('run_result/run3.txt', 'w') as f:
        run3 = GIST(pic_size=(160, 160))
        run3.train()
        text = run3.predict_all()
        f.write(text)
```

```
# -*- coding: utf-8 -*-
# @Time    : 06/12/2018 22:36
# @Author  : weiziyang
# @FileName: deploy.py
# @Software: PyCharm
from K_neighbour import KNeighbour
from words_bag import WordsBag
from sift import SIFT
from GIST import GIST

import time
import logging

if __name__ == "__main__":
    # find the best parameter for run1
    # for k in range(1, 8):
    #     for size in (i*8 for i in range(1, 8)):
    #         k_neigh = KNeighbour(k=k, pic_size=size)
    #         k_neigh.train()
    #         mark = k_neigh.test_model()
    #         time.sleep(10)

    # find the best parameter for run2
    # for size in [32*2**i for i in range(0, 4)] + [None]:
    #     for gap in [4*_ for _ in range(1, 3)]:
    #         patch_size = gap * 2
    #         for cluster_num in [200*_ for _ in range(1, 6)]:
    #             try:
    #                 bag = WordsBag(patch_gap=gap, patch_size=patch_size, cluster_num=cl
uster_num,
    #                                output_patch_image=False, mini_patch=True, pic_size=
size,
    #                                force_generate_again=True)
    #                 bag.train()
    #                 mark = bag.test_model()
    #             except Exception as e:
    #                 continue
    #
    # # find the best parameter for run3:
    for cluster in [100*i for i in range(1, 11)]:
        try:
            sift = SIFT(cluster_num=cluster, mini_patch=True, force_generate_again=True)
            sift.train()
            mark = sift.test_model()
        except Exception as e:
            continue

    # find the best parameter for gist
    # for img_size in [32*i for i in range(1, 9)]:
    #     gist = GIST(pic_size=(img_size, img_size))
    #     gist.train()
    #     gist.test_model()
# coding: utf-8

# In[1]:


import numpy as np
import matplotlib.pyplot as plt
f = open('production_log.txt','r')
log = f.readlines()
f.close()


# In[4]:


len(log)


# In[5]:


log[0]


# In[14]:


n = 0
for line in log:
    if 'KNeigh' in str(line):
        n += 1
print(n)


# In[18]:


Kneighbour_performance = np.zeros((51,3))


# In[23]:


import re
n = 0
for line in log:
    if 'KNeigh' in line:
        result = re.search(r'K:(\d+)-Pic_size:(\d+).+Correct: (\d+)', line)
        K, pic_size,mark = result.group(1), result.group(2), result.group(3)
        Kneighbour_performance[n][0] = int(K)
        Kneighbour_performance[n][1] = int(pic_size)
        Kneighbour_performance[n][2] = int(mark)
        n += 1


# In[24]:


Kneighbour_performance


# In[39]:


size8 = Kneighbour_performance[Kneighbour_performance[:,1] == 8][2:]
size16 = Kneighbour_performance[Kneighbour_performance[:,1] == 16]
size24 = Kneighbour_performance[Kneighbour_performance[:,1] == 24]
plt.plot(size8[:,0], size8[:,2], label='$size=8*8$')
plt.plot(size16[:,0], size16[:,2],label='$size=16*16$')
plt.plot(size24[:,0], size24[:,2], label='$size=24*24$')
plt.xlabel('k-nearest')
plt.ylabel('performance')
plt.title('KNN Performance Benchmark')
plt.legend()


# In[3]:


n = 0
for line in log:
    if 'WordsBag' in line and 'Correct' in line:
        n += 1
print(n)


# In[12]:


import re
BOW_performance = np.zeros((199,5))
n = 0
for line in log:
    if 'WordsBag' in line and 'Correct' in line:
        result = re.search(r'WordsBag-Patch_size:(\d+)-Patch_gap:(\d+)-Pic_size:(\w+)-Clu
ster_num:(\d+).+Correct: (\d+)', line)
        patch_size, patch_gap, pic_size,cluster_num, correct = result.group(1), result.gr
oup(2), result.group(3), result.group(4), result.group(5)
        BOW_performance[n][0] = int(patch_size)
        BOW_performance[n][1] = int(patch_gap)
        BOW_performance[n][2] = np.nan if pic_size == 'None' else int(pic_size)
        BOW_performance[n][3] = int(cluster_num)
        BOW_performance[n][4] = int(correct)
        n += 1


# In[13]:


BOW_performance


# In[14]:


n = 0
for line in log:
    if 'GIST' in line and 'Correct' in line:
        n += 1
print(n)


# In[20]:


import re
GIST_performance = np.zeros((8,2))
n = 0
for line in log:
    if 'GIST' in line and 'Correct' in line:
        result = re.search(r'GIST-pic_size:\((\d+).+Correct: (\d+)', line)
        pic_size,correct = result.group(1), result.group(2)
        print(pic_size,correct)
        GIST_performance[n][0] = int(pic_size)
        GIST_performance[n][1] = int(correct)
        n += 1


# In[21]:
```

```
GIST_performance


# In[23]:


plt.plot(GIST_performance[:,0], GIST_performance[:,1])
plt.title(’GIST Performance’)
plt.xlabel(’picture size’)
plt.ylabel(’Correct rate (%)’)


# In[39]:


for line in log_string.split(’\n’):
    result = re.search(r’category:(\w+) correct_ratio:(\d+)’,line)
    if result:
        category, correct_ratio = result.group(1), result.group(2)
        print(category, correct_ratio)


# In[33]:


log_string.split(’\n’)[-3]


# In[34]:


string = log_string.split(’\n’)[-3]


# In[38]:


re.search(r’category:(\w+) correct_ratio:(\d+)’,string).group(2)


# In[55]:


f = open(’normalize_data.txt’,’r’)
log = f.readlines()
f.close()
n = 0
for line in log:
    if ’WordsBag’ in line and ’Correct’ in line:
        n += 1
import re
BOW_performance = np.zeros((n,5))
n = 0
for line in log:
    if ’WordsBag’ in line and ’Correct’ in line:
        result = re.search(r’WordsBag-Patch_size:(\d+)-Patch_gap:(\d+)-Pic_size:(\w+)-Clu
ster_num:(\d+).+Correct: (\d+)’, line)
        patch_size, patch_gap, pic_size,cluster_num, correct = result.group(1), result.gr
oup(2), result.group(3), result.group(4), result.group(5)
        BOW_performance[n][0] = int(patch_size)
        BOW_performance[n][1] = int(patch_gap)
        BOW_performance[n][2] = np.nan if pic_size == ’None’ else int(pic_size)
        BOW_performance[n][3] = int(cluster_num)
        BOW_performance[n][4] = int(correct)
        n += 1


# In[56]:
```

```
BOW_performance


# In[60]:


new_per = BOW_performance[BOW_performance[:,0] == 8]


# In[61]:


new_per


# In[62]:


size32 = new_per[new_per[:,2] == 32]
size64 = new_per[new_per[:,2] == 64]
size128 = new_per[new_per[:,2] == 128]
size256 = new_per[new_per[:,2] == 256]
plt.plot(size32[:,-2], size32[:,-1], label='$size=32*32$')
plt.plot(size64[:,-2], size64[:,-1], label='$size=64*64$')
plt.plot(size128[:,-2], size128[:,-1], label='$size=128*128$')
plt.plot(size256[:,-2], size256[:,-1], label='$size=256*256$')

plt.xlabel('Cluster number')
plt.ylabel('Performance')
plt.title('BOW Performance Benchmark')
plt.legend()


# In[64]:


f = open('sift_data.txt','r')
log = f.readlines()
f.close()
n = 0
for line in log:
    if 'SIFT' in line and 'Correct' in line:
        n += 1
import re
SIFT_performance = np.zeros((n,2))
n = 0
for line in log:
    if 'SIFT' in line and 'SIFT' in line:
        result = re.search(r'Parameter:SIFT-Cluster_num:(\d+)Kmeans_type:MINI:sample num:
300 - Correct: (\d+\.\d+)', line)
        cluster_num, mark = result.group(1), result.group(2)
        SIFT_performance[n,0] = cluster_num
        SIFT_performance[n,1] = mark
        n += 1


# In[65]:


SIFT_performance


# In[70]:


plt.plot(SIFT_performance[:,0], SIFT_performance[:,1])

plt.xlabel('Cluster number')
plt.ylabel('Correct Ratio(%)')
plt.ylim((20,80))
plt.title('SIFT Performance Benchmark')


# In[49]:


BOW_performance


# In[7]:


log[0]
```

```python
# -*- coding: utf-8 -*-
# @Time    : 07/11/2018 23:49
# @Author  : weiziyang
# @FileName: K_means.py
# @Software: PyCharm
import os
from collections import Counter

import numpy as np
import cv2

import base


class KNeighbour(base.Base):
    def __init__(self, k, pic_size=16):
        super().__init__()
        self.parameter_token = '{name}-K:{k}-Pic_size:{S}'.format(name=self.__class__.__name__, k=k, S=pic_size)
        self.k = k
        self.pic_size = pic_size

        self.image_matrix = None
        self.image_vector_matrix = None
        self.training_data_matrix_file = os.path.join(self.data_dir,
                                                      '{size}*{size}matrix.pkl'.format(size=self.pic_size))

    def train(self):
        self.image_matrix = self.compute_obj(self.training_data_matrix_file, self.convert_pic2matrix)
        self.image_vector_matrix = np.zeros((self.total_num, self.pic_size * self.pic_size))
        for i in range(self.total_num):
            image_2d = self.image_matrix[:, :, i]
            image_1d = image_2d.flatten()
            self.image_vector_matrix[i, :] = image_1d

    def convert_pic2matrix(self):
        image_matrix = np.zeros((self.pic_size, self.pic_size, self.total_num))
        index = 0
        for category in self.categories:
            for image_name in self.images_name:
                image_path = os.path.join(self.raw_training_dir, category, image_name)
                image_data = cv2.resize(cv2.imread(image_path)[:, :, 0], (self.pic_size, self.pic_size))
                image_matrix[:, :, index] = image_data
                index += 1
        return image_matrix

    def predict(self, image):
        image = image[:, :, 0]
        resized_image = cv2.resize(image, (self.pic_size, self.pic_size))
        test_image_vector = resized_image.flatten()
        # calculate the nearest image
        diff = self.image_vector_matrix - test_image_vector
        distance = np.sum(diff * diff, axis=1)
        most_similar_image = distance.argsort()[:self.k]
        most_similar_class = list(map(lambda a: a//100, most_similar_image))
        counter = Counter(most_similar_class)
        label = max(most_similar_class, key=lambda a: counter[a])
        return label


if __name__ == '__main__':
    run1 = KNeighbour(k=1, pic_size=8)
    run1.train()
    run1.test_model()
```

```
# -*- coding: utf-8 -*-
# @Time    : 12/12/2018 22:27
# @Author  : weiziyang
# @FileName: gist.py
# @Software: PyCharm

import os

import cv2
import gist
from tqdm import tqdm
import numpy as np
from sklearn.svm import SVC

from base import Base, count_time


class GIST(Base):
    def __init__(self, pic_size, force_generate_again=False):
        super().__init__(force_generate_again=force_generate_again)
        self.parameter_token = "{name}-pic_size:{size}".format(
            name=self.__class__.__name__, size=pic_size)
        self.pic_size = pic_size

        self.working_dir = os.path.join(self.data_dir, self.parameter_token)
        if not os.path.exists(self.working_dir):
            os.makedirs(self.working_dir)
        self.svm_file = os.path.join(self.working_dir, ’svm.pkl’)

        self.gist_features = None
        self.svm = None

    @count_time
    def train(self):
        self.gist_features = self.generate_gist(self.pic_size)
        self.svm = self.compute_obj(self.svm_file, self.compute_svm)

    @count_time
    def generate_gist(self, size):
        gist_feature_matrix = np.zeros((self.total_num, 960))
        n = 0
        with tqdm(total=self.total_num) as pbar:
            for category in self.categories:
                for image_name in self.images_name:
                    image_path = os.path.join(self.raw_training_dir, category, image_name
)
                    data = cv2.resize(cv2.imread(image_path), (size[0], size[1]))
                    gist_feature = gist.extract(data)
                    pbar.update(1)
                    gist_feature_matrix[n, :] = gist_feature
                    n += 1
        return gist_feature_matrix

    @count_time
    def compute_svm(self):
        svm = SVC(kernel=’linear’)
        svm.fit(self.gist_features, self.training_y)
        return svm

    def predict(self, image):
        data = cv2.resize(image, (self.pic_size[0], self.pic_size[1]))
        feature = gist.extract(data)
        prediction = self.svm.predict(np.atleast_2d(feature))
        return prediction


if __name__ == "__main__":
    g = GIST(pic_size=(512, 512))
    g.train()
    g.test_model()
```

```
# -*- coding: utf-8 -*-
# @Time    : 05/12/2018 18:11
# @Author  : weiziyang
# @FileName: run2.py
# @Software: PyCharm
import os

import cv2

from tqdm import tqdm
import numpy as np
from sklearn.svm import SVC
from sklearn.cluster import KMeans, MiniBatchKMeans

import base


class WordsBag(base.Base):
    def __init__(self, patch_gap, patch_size, cluster_num, pic_size=None, output_patch_im
age=False, mini_patch=True,
                 force_generate_again=False):
        super().__init__(force_generate_again=force_generate_again)
        self.parameter_token = "{name}-Patch_size:{patch_size}-Patch_gap:{gap}-Pic_size:{
size}-" \
                               "Cluster_num:{n_cluster}-Kmeans_type:{m}".format(
                                name=self.__class__.__name__, patch_size=patch_size, gap=
patch_gap,
                                size=pic_size, n_cluster=cluster_num, m='MINI' if mini_pa
tch else 'NORMAL')
        self.training_dir = self.training_dir.format(pic_size=pic_size)

        # initializing var
        self.logger.info('Start initializing, parameter token is:{token}'.format(token=se
lf.parameter_token))
        self.pic_size = pic_size
        self.patch_gap = patch_gap
        self.patch_size = patch_size
        self.cluster_num = cluster_num
        self.output_patch_image = output_patch_image
        self.mini_patch = mini_patch

        # initializing file path
        self.working_dir = os.path.join(self.data_dir, self.parameter_token)
        if not os.path.exists(self.working_dir):
            os.makedirs(self.working_dir)

        # Some data have been generated so that we don't need to compute them again
        self.training_data_matrix_file = os.path.join(self.data_dir, '{size}*{size}matrix
.pkl'.format(size=self.pic_size))
        self.svm_file = os.path.join(self.working_dir, 'svm.pkl')
        self.k_means_file = os.path.join(self.working_dir, 'k_means.pkl')

        self.training_matrix = None
        self.k_means = None
        self.svm = None

    @base.count_time
    def train(self):
        self.training_matrix = self.compute_obj(self.training_data_matrix_file, self.conv
ert_pic2matrix)
        self.k_means = self.compute_obj(self.k_means_file, self.compute_k_means)
        self.svm = self.compute_obj(self.svm_file, self.compute_svm)

    @base.count_time
    def compute_k_means(self):
        """
        For every images, we divide them into many patches, and cluster into 500 category
 or more...
        :return:k means object
        """
        # divide many patches
        patch_matrix_list = []
        for image_index in range(self.total_num):
            if self.pic_size:
                image = self.training_matrix[:, :, image_index]
            else:
                image = self.training_matrix[image_index]
            y_size, x_size = image.shape[0], image.shape[1]
            y = 0
            while y + self.patch_size <= y_size:
                x = 0
                while x + self.patch_size <= x_size:
                    patch = image[y:y + self.patch_size, x:x + self.patch_size]
                    patch_vector = base.normalize(patch.flatten())
                    patch_matrix_list.append(patch_vector)
                    x += self.patch_gap
                y += self.patch_gap
        patch_matrix = np.vstack(patch_matrix_list)
        if self.mini_patch:
            k_means = MiniBatchKMeans(n_clusters=self.cluster_num, init_size=3*self.clust
er_num).fit(patch_matrix)
        else:
            k_means = KMeans(n_clusters=self.cluster_num).fit(patch_matrix)
        return k_means

    @base.count_time
    def compute_svm(self):
        # compute visual word vector
        image_batch_matrics = np.zeros((self.total_num, self.cluster_num))
        with tqdm(total=self.total_num) as pbar:
            for image_index in range(self.total_num):
                image = self.training_matrix[:, :, image_index] if self.pic_size else sel
f.training_matrix[image_index]
                visual_word_vector = self.convert2visual_word(image)
                image_batch_matrics[image_index, :] = visual_word_vector
                pbar.update(1)
        svm = SVC(kernel='linear')
        svm.fit(image_batch_matrics, self.training_y)
        return svm

    def convert2visual_word(self, image):
        """
        For every image,this method can convert it into a vector that is represented by t
he number of patches.
        :param image:
        :return:
        """
        image_batch_vector = np.zeros(self.cluster_num)
        patch_matrix_list = []
        y_size, x_size = image.shape[0], image.shape[1]
        y = 0
        while y + self.patch_size <= y_size:
            x = 0
            while x + self.patch_size <= x_size:
                patch = image[y:y + self.patch_size, x:x + self.patch_size]
                patch_vector = base.normalize(patch.flatten())
                patch_matrix_list.append(patch_vector)
                x += self.patch_gap
            y += self.patch_gap
        patch_matrix = np.vstack(patch_matrix_list)
        locations = self.k_means.predict(patch_matrix)
        if self.output_patch_image:
            self.logger.info('Output image...')
            index = 0
            for patch_vector, location in zip(patch_matrix, locations):
                patch_img = patch_vector.reshape((self.patch_size, self.patch_size))
                patch_path = os.path.join(self.working_dir, 'patches', str(location))
                if not os.path.exists(patch_path):
                    os.makedirs(patch_path)
                cv2.imwrite(os.path.join(patch_path, '{}.jpg'.format(index)), patch_img)
                index += 1
        for loc in locations:
            image_batch_vector[loc] += 1
        return np.atleast_2d(image_batch_vector)

    def convert_pic2matrix(self):
        """
        resize and convert the origin image to matrix and save them into a file so that w
e don’t need to read the file
        and do the same stuff again and again
        :return:
        """
        if self.pic_size:
            image_matrix = np.zeros((self.pic_size, self.pic_size, self.total_num))
        else:
            image_matrix = []
        index = 0
        for category in self.categories:
            for image_name in self.images_name:
                image_path = os.path.join(self.raw_training_dir, category, image_name)
                if self.pic_size:
                    image_data = cv2.resize(cv2.imread(image_path)[:, :, 0], (self.pic_si
ze, self.pic_size))
                    image_matrix[:, :, index] = image_data
                else:
                    image_data = cv2.imread(image_path)[:, :, 0]
                    image_matrix.append(image_data)
                index += 1
        return image_matrix

    def predict(self, image):
        image = image[:, :, 0]
        if self.pic_size:
            image = cv2.resize(image, (self.pic_size, self.pic_size))
        batch_image = self.convert2visual_word(image)
        prediction = self.svm.predict(batch_image)
        return prediction


if __name__ == "__main__":
    pic_size = None
    patch_gap = 4
    patch_size = 8
    cluster_num = 700
    output_patch_image = True
    mini_patch = True
    run2 = WordsBag(patch_gap=patch_gap, patch_size=patch_size, cluster_num=cluster_num,
                    output_patch_image=output_patch_image, mini_patch=mini_patch, pic_siz
e=pic_size,
                    force_generate_again=True)
    run2.train()
    run2.test_model()
```

```
# -*- coding: utf-8 -*-
# @Time    : 06/12/2018 20:05
# @Author  : weiziyang
# @FileName: base.py
# @Software: PyCharm

import os
import getpass
import logging
import pickle
import datetime
from tqdm import tqdm

import yaml
import cv2
import numpy as np


def count_time(func):
    """
    :param func: Any methods or func
    :return: return how many time the function consumes
    """
    def int_time(self, *args, **kwargs):
        start_time = datetime.datetime.now()  # start time
        result = func(self, *args, **kwargs)
        over_time = datetime.datetime.now()  # end time
        total_time = (over_time - start_time).total_seconds()
        self.logger.info('The function %s costs %s seconds' % (func.__name__, total_time)
)
        return result
    return int_time


def normalize(matrix):
    """
    :param matrix: any shape matrix
    :return: a normalized matrix that has a the same shape with the input matrix
    """
    mean = np.mean(matrix)
    mean_matrix = matrix - mean
    matrix_range = np.max(matrix) - np.min(matrix)
    if matrix_range == 0:
        matrix = np.zeros(matrix.shape)
    else:
        matrix = mean_matrix / matrix_range
    return matrix


class Base(object):
    def __init__(self, force_generate_again=False):
        if getpass.getuser() in ['root', 'weiziyang666']:
            config_file = 'production_config.yaml'
        elif getpass.getuser() == 'weiziyang':
            config_file = 'config.yaml'
        else:
            config_file = None
            raise Exception('You should config your file')
        with open(config_file, 'r') as f:
            config = yaml.load(f)
            self.root_dir = config['root_dir']
            self.data_dir = os.path.join(self.root_dir, config['data_root_dir'])
            self.raw_training_dir = os.path.join(self.root_dir, config['raw_training_dir'
])
            self.training_dir = os.path.join(self.root_dir, config['training_dir'])
            self.test_dir = os.path.join(self.root_dir, config['test_dir'])

        # initializing logging
        self.parameter_token = ''
        self.logger = logging.getLogger(datetime.datetime.now().strftime('%y-%m-%d_%H:%M:
```

```
%S'))
        self.logger.setLevel(logging.INFO)
        handler = logging.FileHandler(config['logging_file'])
        handler.setLevel(logging.INFO)
        formatter = logging.Formatter(config['logging_format'])
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)

        if config['logging_print']:
            console = logging.StreamHandler()
            console.setLevel(logging.INFO)
            console.setFormatter(formatter)
            self.logger.addHandler(console)

        # if it set to False, it will not read the existing model, it will generate a new
 one instead.
        self.force_generate = force_generate_again
        self.categories = ['Forest', 'bedroom', 'Office', 'Highway', 'Coast', 'Insidecity
', 'TallBuilding',
                           'industrial', 'Street', 'livingroom', 'Suburb', 'Mountain', 'k
itchen', 'OpenCountry', 'store']
        self.images_name = [str(i)+'.jpg' for i in range(100)]
        self.total_num = 1500
        self.training_y = np.zeros(self.total_num)
        for n in range(self.total_num):
            category = n // 100
            self.training_y[n] = category

    def compute_obj(self, file, method):
        """
        Like a decorator, we can use this method to save the model so as to speed up the
procedure
        :param file: The file that will save the object(svm object or k_means object...
        :param method:The method that can be used to return a model
        :return:object
        """
        if os.path.exists(file) and not self.force_generate:
            with open(file, 'rb') as f:
                obj = pickle.load(f)
        else:
            obj = method()
            with open(file, 'wb') as f:
                pickle.dump(obj, f)
        return obj

    def test_model(self):
        """
        To test the accuracy of the generated model
        :return:
        """
        self.logger.info('Start test model')
        total_correct = 0
        total_sample = 0
        for n, category in enumerate(self.categories):
            category_correct = 0
            true_category = n
            test_category_path = os.path.join(self.test_dir, category)
            image_names = [each for each in os.listdir(test_category_path) if not each.st
artswith('.')]
            category_sample = len(image_names)
            for path in image_names:
                image_path = os.path.join(test_category_path, path)
                image = cv2.imread(image_path)
                prediction = self.predict(image)
                predict_category = self.categories[int(prediction)]
                self.logger.info('image:{image_path}, predict:{predict} fact:{true_catego
ry}'.format(
                    image_path=image_path, predict=predict_category, true_category=catego
ry))
                if prediction == true_category:
                    category_correct += 1
            correct_ratio = category_correct / category_sample
            total_correct += category_correct
            total_sample += category_sample
            self.logger.log(logging.INFO, 'category:{category} correct_ratio:{ratio}%'.fo
rmat(
                category=category, ratio=correct_ratio * 100))
        self.logger.info('Parameter:{parameter}:sample num:{total} - Correct: {correct}%'
.format(
            parameter=self.parameter_token, total=total_sample, correct=total_correct / t
otal_sample * 100))
        return total_correct / total_sample * 100

    def predict_all(self):
        """
        Mark all of the test image...
        :return:
        """
        testing_catalogue = os.path.join(self.root_dir, 'testing')
        files = sorted([each for each in os.listdir(testing_catalogue) if not each.starts
with('.')], key=lambda a: int(a.split('.')[0]))
        prediction_text = ''''''
        with tqdm(total=len(files)) as pbar:
            for file in files:
                image_path = os.path.join(testing_catalogue, file)
                image = cv2.imread(image_path)
                category_no = self.predict(image)
                category_name = self.categories[int(category_no)]
                temp_string = file + ' ' + category_name.lower() + '\n'
                prediction_text += temp_string
                pbar.update(1)
        return prediction_text

    def predict(self, image):
        raise Exception('Predict method has not been implemented yet')

    def train(self):
        raise Exception('Train method has not been implemented yet')
```