SIT789 – Robotics, Computer Vision and Speech Processing

Pass Task 4.1: Image recognition

Objectives

The objectives of this lab include:

- Practising machine learning methods including K-means, K-NN, SVM, and Adaboost using scit-learn
- Practising Bag-of-Words model
- Applying machine learning methods to image recognition

Tasks

1. Bag-of-Words (BoW) model

In this task, we practise with the **K-means** algorithm via building a **BoW** model for image recognition. You first download the food image database provided in FoodImages.zip in OnTrack, unzip this file to a folder named FoodImages into your working directory. This dataset was used in the following paper:

```
D. T. Nguyen et al. Food image classification using local appearance and
global structure information. Neurocomputing, vol. 140, pp. 242-251,
2014.
```

In FoodImages, there are two folders: Train and Test containing training and test images respectively. Each Train/Test folder contains three sub-folders corresponding to three different food types including Cakes, Pasta, and Pizza. For each food category, there are equal numbers of images (30 images) used for training and testing. We will build a BoW model for food image recognition based on the training images of the supplied food database. You should revisit Week 3 handouts for more details about BoW models. The Dictionary class below is developed to build a BoW model using the K-means algorithm. Although this code is provided to you, you are encouraged to read through and understand it, especially the learn method where the K-means algorithm is used to learn words and the create_word_histograms method which constructs word histograms for a given list of images. You should also visit sklearn.cluster for more details on the K-means algorithm. Note that Python is indentation sensitive, therefore make sure that you strictly follow the indentation used in the given code below.

```python
import numpy as np
import cv2 as cv
from sklearn.cluster import KMeans

class Dictionary(object):
    def __init__(self, name, img_filenames, num_words):
        self.name = name #name of your dictionary
        self.img_filenames = img_filenames #list of image filenames
        self.num_words = num_words #the number of words

        self.training_data = [] #training data used to learn clusters
        self.words = [] #list of words, which are the centroids of clusters

    def learn(self):
        sift = cv.SIFT_create()

        num_keypoints = [] #used to store the number of keypoints in each image

        #load training images and compute SIFT descriptors
        for filename in self.img_filenames:
            img = cv.imread(filename)
            img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
            list_des = sift.detectAndCompute(img_gray, None)[1]
            if list_des is None:
                num_keypoints.append(0)
            else:
                num_keypoints.append(len(list_des))
                for des in list_des:
                    self.training_data.append(des)

        #cluster SIFT descriptors using K-means algorithm
        kmeans = KMeans(self.num_words)
        kmeans.fit(self.training_data)
        self.words = kmeans.cluster_centers_

        #create word histograms for training images
        training_word_histograms = [] #list of word histograms of all training images
        index = 0
        for i in range(0, len(self.img_filenames)): #for each file, create a histogram
            histogram = np.zeros(self.num_words, np.float32)
            #if some keypoints exist
            if num_keypoints[i] > 0:
                for j in range(0, num_keypoints[i]):
                    histogram[kmeans.labels_[j + index]] += 1
                index += num_keypoints[i]
                histogram /= num_keypoints[i]
                training_word_histograms.append(histogram)

        return training_word_histograms
```

```python
    def create_word_histograms(self, img_filenames):
        sift = cv.SIFT_create()
        histograms = []

        for filename in img_filenames:
            img = cv.imread(filename)
            img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
            descriptors = sift.detectAndCompute(img_gray, None)[1]

            histogram = np.zeros(self.num_words, np.float32) #word histogram

            if descriptors is not None:
                for des in descriptors:
                    #find the best matching word
                    min_distance = 1111111 #this can be any large number
                    matching_word_ID = -1 #initialise ID with an impractical value

                    for i in range(0, self.num_words): #find the best matching word
                        distance = np.linalg.norm(des - self.words[i])
                        if distance < min_distance:
                            min_distance = distance
                            matching_word_ID = i

                    histogram[matching_word_ID] += 1

                histogram /= len(descriptors) #make histogram a prob distribution

            histograms.append(histogram)

        return histograms
```

Before building the BoW model for our food database, we need to prepare training data. The following code creates two lists:

- training_file_names: containing the file names of all training images
- training_food_labels: containing the food labels of all training images, e.g., Cakes have labels as 0, Pasta as 1, and Pizza as 2.

```python
import os

foods = ['Cakes', 'Pasta', 'Pizza']
path = 'FoodImages/'
training_file_names = []
training_food_labels = []
for i in range(0, len(foods)):
    sub_path = path + 'Train/' + foods[i] + '/'
    sub_file_names = [os.path.join(sub_path, f) for f in os.listdir(sub_path)]
    sub_food_labels = [i] * len(sub_file_names) #create a list of N elements, all are i
    training_file_names += sub_file_names
    training_food_labels += sub_food_labels

print(training_file_names)
print(training_food_labels)
```

We are now ready for building the BoW model for our food recognition problem. Suppose that we want to have 50 words in our dictionary (though this number can vary). We name our dictionary 'food' and define it as:

```python
num_words = 50
dictionary_name = 'food'
dictionary = Dictionary(dictionary_name, training_file_names, num_words)
```

We learn the dictionary, i.e., finding words, by calling the learn method as below. The learn method not only extracts words from a training dataset but also creates the word histograms for all the training images in the training set.
Note: **The learning process takes time. Please be patient!**

```
training_word_histograms = dictionary.learn()
```

Since training a dictionary is time consuming, we should save the dictionary into file once the training is complete and reload it for use without retraining. To save the dictionary into file, you can use pickle as follows.

```
import pickle
#save dictionary
with open('food_dictionary.dic', 'wb') as f: #'wb' is for binary write
    pickle.dump(dictionary, f)
```

To load the dictionary, we can do as,

```
import pickle #you may not need to import it if this has been done
with open('food_dictionary.dic', 'rb') as f: #'rb' is for binary read
    dictionary = pickle.load(f)
```

2. k-NN

In this section, we will apply the k-NN technique to food image recognition by using sklearn.neighbors.KNeighborsClassifier. We first declare a k-NN classifier and train it using the training_word_histograms and training_food_labels.

```
from sklearn.neighbors import KNeighborsClassifier
num_nearest_neighbours = 5 #number of neighbours
knn = KNeighborsClassifier(n_neighbors = num_nearest_neighbours)
knn.fit(training_word_histograms, training_food_labels)
```

We now test the knn classifier with a random food image in our test sets. For example, we choose the food image in FoodImages/Test/Pasta/pasta35.jpg. In this test, we use dictionary.create_word_histograms to extract the word histogram for the image in FoodImages/Test/Pasta/pasta35.jpg, then feed the histogram into the knn classifier by calling knn.predict to get the food label (class): '0' for Cakes, '1' for Pasta and '2' for Pizza. The expected food label for this image is '1'.

Note knn.predict receives input as a list of test samples (each sample is represented as a word histogram) and returns output as a list food labels.

```
test_file_names = ['FoodImages/Test/Pasta/pasta35.jpg']
word_histograms = dictionary.create_word_histograms(test_file_names)

predicted_food_labels = knn.predict(word_histograms)
print('Food label: ', predicted_food_labels)
```

Your task now is to evaluate the knn classifer on the whole dataset. In particular,

1. Test the knn classifer with all the test images of all the food types, i.e., test_file_names should include all the images in the Test folder. You therefore do need to reload ALL the image filenames in the Test folder into test_file_names and then re-calculate word_histograms for all the test images.

2. Report the recognition accuracy, i.e., the ratio of the number of correct predictions (of food labels) and the total number of test images.
3. Calculate the confusion matrix using the following code

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
cm = confusion_matrix(test_food_labels, predicted_food_labels)
```

```
print(cm)
```

4. Vary num_nearest_neighbours in the range [10, 15, 20, 25, 30] and measure the corresponding accuracies. What is the best value for num_nearest_neighbours?

3. SVM

In this section, we will use linear SVM to classify food images. We will learn how to use SVM from sklearn.svm.SVC. To train an SVM classifier, we use the following code

```
from sklearn import svm
svm_classifier = svm.SVC(C = 50, #see slide 32 in week 4 handouts
                         kernel = 'linear') #see slide 35 in week 4 handouts
svm_classifier.fit(training_word_histograms, training_food_labels)
```

Like knn, we test the svm_classifier with the food image in FoodImages/Test/Pasta/pasta35.jpg as,

```
test_file_names = ['FoodImages/Test/Pasta/pasta35.jpg']
word_histograms = dictionary.create_word_histograms(test_file_names)

predicted_food_labels = svm_classifier.predict(word_histograms)
print('Food label: ', predicted_food_labels)
```

Your task now is to evaluate the svm_classifier on the whole dataset. In particular,

1. Test the svm_classifier with all the test images of all the food types, i.e., test_file_names should include all the images in the Test folder. You therefore do need to reload ALL the image filenames in the Test folder into test_file_names and then re-calculate word_histograms for all the test images.
2. Report the recognition accuracy, i.e., the ratio of the number of correct predictions (of food labels) and the total number of test images.
3. Calculate the confusion matrix using the following code

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
cm = confusion_matrix(test_food_labels, predicted_food_labels)
```

```
print(cm)
```

4. Vary C in the range [10, 20, 30, 40, 50] and measure the corresponding accuracies. What is the best value for C?

4. AdaBoost

In this section, we will use AdaBoost for food image recognition using sklearn.ensemble.AdaBoostClassifier. To declare and train an AdaBoost classifier, we use the following code

```
from sklearn.ensemble import AdaBoostClassifier
adb_classifier = AdaBoostClassifier(n_estimators = 150, #number of weak classifiers
                                    random_state = 0)
adb_classifier.fit(training_word_histograms, training_food_labels)
```

We test the adb_classifier with the food image in FoodImages/Test/Pasta/pasta35.jpg as,

```
test_file_names = ['FoodImages/Test/Pasta/pasta35.jpg']
word_histograms = dictionary.create_word_histograms(test_file_names)

predicted_food_labels = adb_classifier.predict(word_histograms)
print('Food label: ', predicted_food_labels)
```

Your task now is to evaluate the adb_classifier on the whole dataset. In particular,

1. Test the adb_classifier with all the test images of all the food types, i.e., test_file_names should include all the images in the Test folder. You therefore do need to reload ALL the image filenames in the Test folder into test_file_names and then re-calculate word_histograms for all the test images.
2. Report the recognition accuracy, i.e., the ratio of the number of correct predictions (of food labels) and the total number of test images.
3. Calculate the confusion matrix using the following code

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
cm = confusion_matrix(test_food_labels, predicted_food_labels)
```

```
print(cm)
```

4. Vary n_estimators in the range [50, 100, 150, 200, 250] and measure the corresponding accuracies. What is the best value for n_estimators?

Submission instructions

1. Perform tasks required in Section 1, 2, 3, and 4.
2. Complete the provided answer sheet and submit the answer sheet (.pdf) to OnTrack.
3. Submit your code (.py) to OnTrack.