ECE 661: Homework 8

Xingguang Zhang
(Fall 2020)

Introduction

Gram matrix is widely used for measure the texture of images. If we filter an image with m convolutional kernels (denoted by $k_1, k_2, \ldots, k_m$), we get m output maps, resize them and flatten these maps to m vectors $v_1, v_2, \ldots, v_m$ because we only need texture but not structural information. Gram matrix G is then defined as:

$$G = \begin{bmatrix} (v_1, v_1) & (v_1, v_2) & \cdots & (v_1, v_m) \\ (v_2, v_1) & (v_2, v_2) & \cdots & (v_2, v_m) \\ \vdots & \vdots & \ddots & \vdots \\ (v_m, v_1) & (v_m, v_2) & \cdots & (v_m, v_m) \end{bmatrix}$$

Where $(,)$ is the inner product of two vectors. The Gram matrix is symmetric, if we use it as the feature for images, we only need the upper triangular elements of it. So we take and expand them to a $\frac{m^2+m}{2}$-element vector.

Implementation

1. For each image, we first filter them by C random filters (in uniform distribution on $[-1, 1]$) with size $k \times k$, then we resize these C output maps into $K \times K$, finally we can compute the Gram matrix G for that image as its feature.

2. After extracting the Gram matrix for every image in training dataset, we split the it into training and validation set, I randomly select 75% of images for training and the rest for validation for each iteration.

3. Train the support vector machine (SVM) on the training set, I used an SVM with 3-degree polynomial kernel, and evaluate its performance on the validation set.

4. Do steps 1-3 for n iterations, here I set $n = 10 \sim 30$ based on the running time. We select the convolutional kernel with the highest accuracy on validation set. The overall validation accuracy is defined as the average of all trails.

5. Retrain the SVM on the whole training set, and apply the best kernel for testing images to generate Gram matrix features. Finally we can compute the accuracy and confusion matrix on testing set, save the convolutional kernel and SVM parameters.

Results and extra credits

The results of all experiment are shown below. The main hyper-parameters I tried are the convolutional kernel size and channels.

kernel size	3×3	5×5	7×7	9×9
channel $= 8$	0.498	0.518	0.553	0.513
channel $= 16$	0.558	0.558	0.589	0.590
channel $= 32$	0.565	0.557	0.579	0.615

Table 1: average accuracy on validation set

kernel size	3×3	5×5	7×7	9×9
channel $= 8$	0.500	0.525	0.575	0.550
channel $= 16$	0.550	0.550	0.725	0.600
channel $= 32$	0.525	0.625	0.700	0.625

Table 2: accuracy on testing set

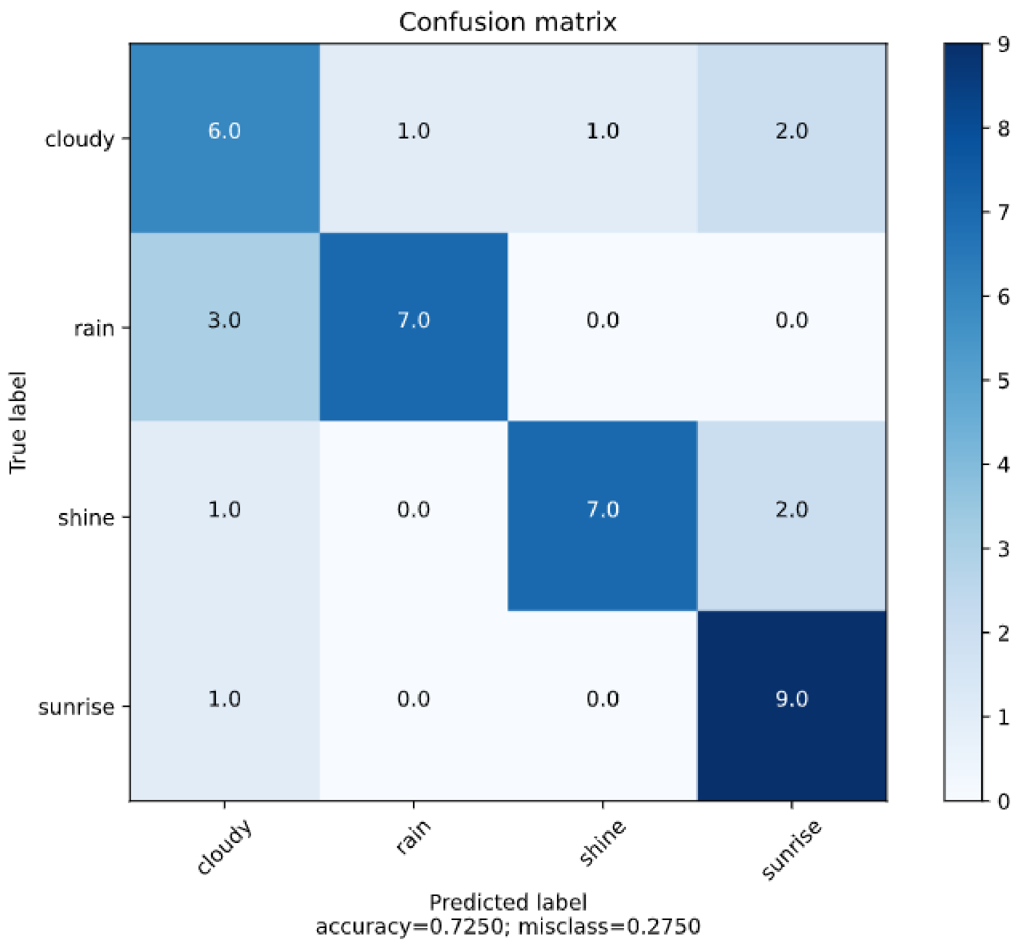


Figure 1: confusion matrix for the best result when $K = 7$ and $C = 16$

Extra credits

Run my code on hw7 images and I finally got the best result when $K = 11$ and $C = 8$, all experiment results are shown as:

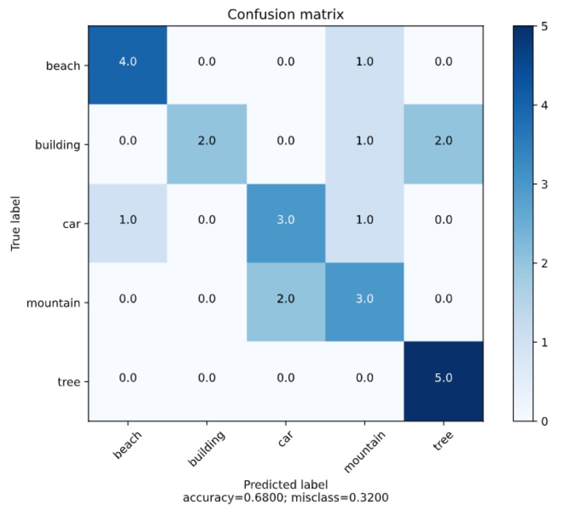
kernel size	3×3	5×5	7×7	9×9	11×11
channel = 4	0.278	0.292	0.360	0.360	0.352
channel = 8	0.282	0.286	0.360	0.356	0.376
channel = 16	0.274	0.356	0.360	0.382	0.372
channel = 32	0.292	0.338	0.366	0.366	0.426

Table 3: average accuracy on validation set

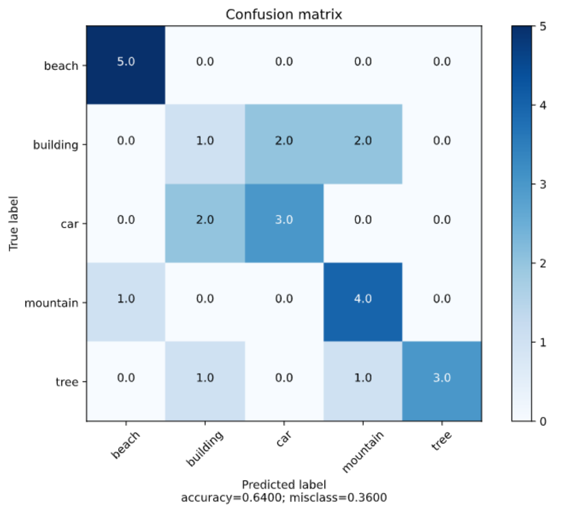
kernel size	3×3	5×5	7×7	9×9	11×11
channel = 4	0.360	0.600	0.440	0.680	0.600
channel = 8	0.400	0.440	0.600	0.600	0.680
channel = 16	0.480	0.560	0.520	0.640	0.640
channel = 32	0.400	0.520	0.440	0.520	0.560

Table 4: accuracy on testing set

The best result is 68% accuracy, which outperforms the LBP based approach (64%). The confusion matrices of these two are shown below:



(a) Gram matrix based result



(b) LBP based result

Figure 2: comparison for the confusion matrix

Observation

1. The result is related to the kernel number and size, usually the larger the kernel size is, the better result will we have, but if we set kernel size larger than 7, the improvement will be trivial. For the channel, or kernel number, which directly defines the feature size of the image, we

can clearly observe that if best channel number is 16, although there's a obvious improvement from 8 to 16, it doesn't mean 32 channels are better than 16.

2. I tested the resize of the images to different sizes before computing the Gram matrix, and the size 16 is relatively good enough, increasing and decreasing the sizes didn't improve the result in my experiments.

3. The best hyper parameters for images in howework 8 and homework 7 are different. Basically because there are less images in homework 7 dataset, the best channel number appears to be 4 and 8. Usually with larger data size, we will need higher dimension features for classification.

4. Generally, in this homework, the SVM with polynomial kernel performs better than with the RBF kernel.

5. We can observe the validation result and testing result are positively correlated, but their absolute accuracy differs a lot. Generally the testing accuracy is higher, partly because we selected the best kernel for testing images, partly because the testing images are similar in distribution with the training set.

Code

```python
import numpy as np
import cv2
from sklearn import svm
import matplotlib.pyplot as plt
import glob
import pickle
import os

# data load for hw8 images
folder_train = 'images/training/'
folder_test = 'images/testing/'
classes = ['cloudy', 'rain', 'shine', 'sunrise']
train_val_images = {'cloudy':[], 'rain':[], 'shine':[], 'sunrise':[]}
test_images = {'cloudy':[], 'rain':[], 'shine':[], 'sunrise':[]}
for c in classes:
    name_list_train = glob.glob(folder_train + c + "*.jpg")
    name_list_test = glob.glob(folder_test + c + "*.jpg")
    for f in name_list_train:
        img = cv2.imread(f)
        try:
            s = img.shape
        except AttributeError:
            print(f)
        train_val_images[c].append(img)
    for f in name_list_test:
        test_images[c].append(cv2.imread(f))
    print(len(train_val_images[c]))

# data load for hw7 images
folder_train = '/home/xingguang/Documents/ECE661/hw7/images/training'
folder_test = '/home/xingguang/Documents/ECE661/hw7/images/testing'
classes = ['beach', 'building', 'car', 'mountain', 'tree']
train_val_images = {'beach':[], 'building':[], 'car':[], 'mountain':[], 'tree':[]}
test_images = {'beach':[], 'building':[], 'car':[], 'mountain':[], 'tree':[]}
```

```
for c in classes:
    for i in range(1, 21):
        img_path = os.path.join(folder_train, c, '{0:02d}.jpg'.format(i))
        img = cv2.imread(img_path)
        try:
            s = img.shape
        except AttributeError:
            print(img_path)
        train_val_images[c].append(img)
    for i in range(1, 6):
        img_path = os.path.join(folder_test, c + '_' + str(i) +'.jpg')
        test_images[c].append(cv2.imread(img_path))

def gram(data_dict, kernel, resized=16, normalize=False):
    channel = kernel.shape[0]
    kernel_size = kernel.shape[1]
    classes = list(data_dict.keys())
    iu = np.triu_indices(channel)
    data = []
    label = []
    for label_idx, c in enumerate(classes):
        for img in data_dict[c]:
            feature_img = np.zeros((channel, resized*resized*3))
            if normalize:
                feature_img = feature_img / np.linalg.norm(feature_img)
            for i in range(channel):
                filtered = cv2.resize(cv2.filter2D(img,-1,kernel[i]), (resized,
   resized))
                feature_img[i,:] = filtered.reshape((1, -1))
            gram = np.dot(feature_img, feature_img.T)
            data.append(gram[iu].reshape((1, -1)))
            label.append(label_idx)
    data_vec = np.concatenate(data, axis=0)
    label_vec = np.array(label)
    return data_vec, label_vec

def split_data(data, label, train_ratio=0.75):
    num = data.shape[0]
    pidx = np.random.permutation(num)
    train_num = int(num * train_ratio)
    train_idx = pidx[:train_num]
    valid_idx = pidx[train_num:]
    train_data = data[train_idx]
    valid_data = data[valid_idx]
    train_label = label[train_idx]
    valid_label = label[valid_idx]
    return train_data, train_label, valid_data, valid_label


def train(data, kernel_size, channel, resized, train_ratio, train_steps):
    clf = svm.SVC(kernel='poly')
    acc_list = []
    kernel_list = []
    for n in range(train_steps):
        kernel = np.random.uniform(-1, 1, size=(channel, kernel_size, kernel_size)
   )
        train_val_vecs, train_val_label = gram(data, kernel, resized)
        train_data, train_label, valid_data, valid_label = split_data(
   train_val_vecs, train_val_label, train_ratio)
```

```
        clf.fit(train_data, train_label)
        predicted = clf.predict(valid_data)
        acc = np.sum(predicted==valid_label) / valid_label.shape[0]
        print(n, acc)
        acc_list.append(acc)
        kernel_list.append(kernel)
    acc_avg = np.sum(acc_list) / len(acc_list)
    best_idx = np.argmax(acc_list)
    print("acc_avg:", acc_avg, "best_idx:", best_idx)
    return kernel_list[best_idx]


def make_confusion_matrix(predicted, gt, class_num=4):
    cm = np.zeros((class_num, class_num))
    for i in range(class_num):
        idx_class_i = np.where(gt == i)
        predicted_part = predicted[idx_class_i]
        for j in range(class_num):
            cm[i, j] = np.sum(predicted_part == j)
    return cm

def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(
    accuracy, misclass))
    plt.show()

kernel_size = 7
channel = 16
resized = 16
train_ratio = 0.75
train_steps = 10

train_vecs, train_label = gram(train_val_images, kernel, resized)
test_data, test_label = gram(test_images, kernel, resized)
clf = svm.SVC(kernel='poly')
clf.fit(train_vecs, train_label)
predicted = clf.predict(test_data)
confusion_matrix = make_confusion_matrix(predicted, test_label, 4)
plot_confusion_matrix(confusion_matrix,
                          target_names=classes,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=False)

with open('best_svm.pkl', 'wb') as handle:
    pickle.dump(clf, handle)
np.save('best_kernel.npy', kernel)


def reproduce(test_images, kernel_path='best_kernel7.npy', svm_path='best_svm7.pkl
    '):
    kernel_size = 5
    channel = 16
    resized = 16
    train_ratio = 0.75
    train_steps = 20
    kernel = np.load(kernel_path)
    clf = pickle.load(open(svm_path, 'rb'))
    # train_vecs, train_label = gram(train_val_images, kernel, resized)
    test_data, test_label = gram(test_images, kernel, resized)
    # clf.fit(train_vecs, train_label)
    predicted = clf.predict(test_data)
    confusion_matrix = make_confusion_matrix(predicted, test_label, 5)
    plot_confusion_matrix(confusion_matrix,
                              target_names=classes,
                              title='Confusion matrix',
                              cmap=None,
                              normalize=False)
reproduce(test_images)
```