**ECE 661 Homework 7**

**Weichen Xu**

**xu1363@purdue.edu**

**1. Gram-matrix Based Classification Using SVM**

**Step 1: Preprocessing the data**

Since all images in the task do not necessarily have the same size, we need to convert them to a common size. We can either directly resize the image to a designated size, or we can crop images from the center to the size of image which has the minimal size in all images. In this case, my implementation is to resize all image to the shape of . Besides for color images, we need to convert them into grayscale images.

**Step 2: Build Gram-matrix based features**

First, we need to build different random convolutional kernels, while the sum of kernel should be . Then convolve these convolutional kernels with the grayscale image, thus we can get output convolution results, where we refer them as channels. Then for each channel, we will downsample it to the size , here I choose . We can further flatten the 2D outputs into vectors of elements. For vectors, we can take the inner products for all pairs to get the symmetric Gram matrix of size .

Since the Gram matrix is symmetric, we can represent the matrix using only its upper triangular matrix. An image can thus be represented with a point in -dimension space.

For , we have following Gram representation for an image,

**Step 3: Training with support vector machine (SVM)**

For the training with SVM, we need to separate all training data into two subsets, training set and validation set. In my implementation, I randomly assign 70% to the training set, 30% to the validation set. Then use OpenCV library to generate the SVM model for training and obtain prediction accuracy for validation set.

Step 4: Run for trials of training for best SVM model and convolutional kernels

For trials of running step 1-3, we get the validation accuracy and SVM model and convolutional kernels at the end of each trial. Then save the SVM mode and convolutional kernels only when the validation accuracy improves comparing to current best validation accuracy.

**Step 5: Test on testing images**

Using saved SVM model and convolutional kernels to predict with the testing images, obtain the accuracy as the testing data accuracy.

**2. Experiment Results**

Trial #1

Training accuracy is 0.3034300791556728

Validation accuracy is 0.31384615384615383

svm has been saved

Trial #2

Training accuracy is 0.1741424802110818

Validation accuracy is 0.16615384615384615

Trial #3

Training accuracy is 0.27440633245382584

Validation accuracy is 0.27692307692307694

Trial #4

Training accuracy is 0.21108179419525067

Validation accuracy is 0.23692307692307693

Trial #5

Training accuracy is 0.3113456464379947

Validation accuracy is 0.3230769230769231

svm has been saved

Trial #6

Training accuracy is 0.23218997361477572

Validation accuracy is 0.24615384615384617

Trial #7

Training accuracy is 0.4179419525065963

Validation accuracy is 0.4569230769230769

svm has been saved

Trial #8

Training accuracy is 0.287598944591029

Validation accuracy is 0.23076923076923078

….

|  |  |
| --- | --- |
| Number of trials |  |
| Number of channels |  |
| Number of features |  |
| SVM kernel | Linear |

Table 1: Parameter for best trained SVM model

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Prediction | | | |
|  |  | Cloudy | Rain | Shine | Sunrise |
| Ground Truth | Cloudy | **6** | 0 | 0 | 4 |
| Rain | 1 | **1** | 2 | 6 |
| Shine | 0 | 0 | **2** | 8 |
| Sunrise | 0 | 0 | 0 | **10** |

Table 2: Confusion matrix for testing images

The overall accuracy is **47.5%**, it is not a very high result comparing to the deep learning methods, but it at least shows improvement than random guess accuracy **25%**.

From the confusion matrix we can notice that for all “sunrise” image, the trained SVM model give correct prediction. However, for testing images of other classes, the SVM model still tends to predict it as the “sunrise”, it makes some sense that “shine” image which has very similar features as “sunrise” images, the SVM predict most such class’s images as “sunrise”. For other classes, the SVM model still largely likely to predict it as “sunrise”, I suspect since the convolutional kernels are randomly generated, it cannot exactly represent the real features in the images.

**3. Observations**  
For this classification task, I have tried with different set of and different setting for SVM classifier, including kernel type, i.e. linear, polynomial, RBF kernels.

For values, I start with , the training process is very fast. But for each trial, the validation accuracy never exceeds 30%, which naturally gives a poor testing accuracy around 20%, which is even worse than random guess. Then I go with higher value, roughly speaking this does increase the overall accuracy. But when is above 20, In my case I tried as high as 100 it does not have enough positive affect on the accuracy.

For the selection of SVM model, I mainly experiment with two SVM kernel, linear and radial basis function (RBF). The RBF kernel tends to best fit the training data, which corresponds to the definition of such function, but returns same class prediction for validation and testing data. So I decide to go with the linear kernel, which is the easy and simple-minded method, and it gives reasonable predictions. The final SVM model still rely heavily on the convolutional kernels, however, in this case convolutional kernels are totally random.

**4. Code**

*#!/usr/bin/env python  
# coding: utf-8*import numpy as np  
import cv2  
import time  
import os  
import matplotlib.pyplot as plt  
from scipy import signal  
from sklearn import svm, metrics  
from sklearn.metrics import confusion\_matrix  
  
def get\_conv\_mask(C):  
 *# create C of different convolutional masks* masks = np.zeros((C,3,3))  
 for i in range(C):  
 conv\_mask = np.random.uniform(-1, 1, size = (3,3))  
 conv\_mask = conv\_mask - np.sum(conv\_mask) / 9  
 masks[i, :, :] = conv\_mask  
 return masks  
  
def get\_Gram\_feature(img, masks):  
 *# masks contatin C different convolutional masks* vec = []  
 for i in range(masks.shape[0]):  
 mask = masks[i,:,:]  
 *# obtain result of convolving image with mask* img\_conv = signal.convolve2d(img, mask, mode=**'same'**)  
 img\_output = cv2.resize(img\_conv, (16, 16), interpolation = cv2.INTER\_AREA)  
 vec.append(img\_output.flatten())  
   
 vec = np.array(vec)  
 gram = np.dot(vec, np.transpose(vec)) *# obtain gram matrix  
   
 # retain the upper triangular part of gram matrix* vec\_feature = []  
 for i in range(C):  
 for j in range(C):  
 if j >= i:  
 vec\_feature.append(gram[i][j])  
   
 return np.array(vec\_feature)  
  
def check\_label(name):  
 *# obtain image label from file name* if **'cloudy'** in name:  
 return 0  
 if **'rain'** in name:  
 return 1  
 if **'shine'** in name:  
 return 2  
 if **'sunrise'** in name:  
 return 3  
  
def seperate\_dataset(inputs, labels):  
 *# seperate data into training data and validation data* idx\_arr = np.arange(len(inputs))  
 np.random.shuffle(idx\_arr)  
   
 ratio = 0.7 *# ratio of training data in all data* idx\_train, idx\_test = idx\_arr[: int(ratio \* len(inputs))], idx\_arr[int(ratio \* len(inputs)) : ]   
   
 inputs\_train, labels\_train = inputs[idx\_train, :], labels[idx\_train]  
 inputs\_test, labels\_test = inputs[idx\_test, :], labels[idx\_test]  
   
  
 return inputs\_train, labels\_train, inputs\_test, labels\_test  
  
def training(N, C):  
 training\_path = **'/home/xu1363/Documents/ECE 661/hw8/imagesDatabaseHW8/training/'** *# resizing image to common size* h = 128  
 w = 128  
 accuracy\_best = 0  
   
 for n in range(N):  
 print(**'Trial #'** + str(n+1))  
 labels = []  
 inputs = []  
 masks = get\_conv\_mask(C)  
 for name in os.listdir(training\_path):  
 label = check\_label(name)  
 img = cv2.imread(training\_path + name)  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img = cv2.resize(img, (h, w), interpolation = cv2.INTER\_AREA)  
 vec\_feature = get\_Gram\_feature(img, masks)  
  
 labels.append(int(label))  
 inputs.append(vec\_feature)  
  
 inputs = np.array(inputs)  
 labels = np.array(labels)  
   
 *# build SVM model* svm=cv2.ml.SVM\_create()  
 svm.setC(1)  
 svm.setGamma(0.1)  
 svm.setType(cv2.ml.SVM\_C\_SVC)  
 svm.setKernel(cv2.ml.SVM\_LINEAR)  
 *#svm.setKernel(cv2.ml.SVM\_RBF)  
 #svm.setTermCriteria((cv2.TERM\_CRITERIA\_MAX\_ITER, 1000, 1e-6))  
  
 # train* inputs\_train, labels\_train, inputs\_valid, labels\_valid = seperate\_dataset(inputs, labels)  
 svm.train(np.float32(inputs\_train), cv2.ml.ROW\_SAMPLE, labels\_train)  
   
 \_,preds =svm.predict(np.float32(np.array(inputs\_train)))  
 preds = preds.ravel()  
 accuracy\_train = metrics.accuracy\_score(labels\_train, preds)  
 print(**'Training accuracy is '**, accuracy\_train)  
   
 *# validate* \_,preds =svm.predict(np.float32(np.array(inputs\_valid)))  
 preds = preds.ravel()  
 accuracy\_valid = metrics.accuracy\_score(labels\_valid, preds)  
 print(**'Validation accuracy is '**, accuracy\_valid)  
   
 *# save model only when validation accuracy improves* if accuracy\_valid > accuracy\_best:  
 accuracy\_best = accuracy\_valid  
 masks\_best = masks  
 svm.save(**'/home/xu1363/Documents/ECE 661/hw8/trained\_svm.xml'**)  
 print(**'svm has been saved'**)  
   
 print(**''**)  
   
 *# return with convolutional masks generating best validation accuracy* return masks  
  
def testing(masks):  
 testing\_path = **'/home/xu1363/Documents/ECE 661/hw8/imagesDatabaseHW8/testing/'** h = 128  
 w = 128  
 *#C = 8* labels = []  
 inputs = []  
 svm=cv2.ml.SVM\_load(**'trained\_svm.xml'**)  
 for name in os.listdir(testing\_path):  
 label = check\_label(name)  
 img = cv2.imread(testing\_path + name)  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img = cv2.resize(img, (h, w), interpolation = cv2.INTER\_AREA)  
 vec\_feature = get\_Gram\_feature(img, masks)  
 labels.append(int(label))  
 inputs.append(vec\_feature)  
   
 \_, preds=svm.predict(np.float32(np.array(inputs)))  
 preds = preds.ravel()  
 print(labels)  
 print(preds)  
   
 accuracy = metrics.accuracy\_score(labels,preds)  
 print(**'Test accuracy is'**, accuracy)  
 print(metrics.confusion\_matrix(labels, preds))  
  
  
C = 20  
N = 100  
masks = training(N, C)  
testing(masks)