ECE 661 Computer Vision: HW8

Qiuchen Zhai
qzhai@purdue.edu

October 27, 2020

1 Gram Matrix Approach for Texture Characterization

The implementation of gram matrix based approach is described as follows:

1. Generate convolutional operators for each channel.
   For each convolution channel, we generate a random convolutional operator of size $M \times M$. Each weight in the convolutional operator is uniformly drawn from the interval $[-1, 1]$ and the weights in each operator add up to 0. Let C denote the number of channels, then we need to generate C convolutional operators in total.

2. Convolve images with the generated convolutional operators.
   Each image is convolved with C different convolutional operators generated in previous step. The output of each channel is down-sampled to size of $K \times K$ and then vectorized to K^2-element vector.

3. Compute the Gram Matrix.
   We take the inner products of output of each channel and obtain a symmetric matrix of size $C \times C$. Let H denote the symmetric matrix that $H = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}$. Then we could benefit from the symmetric property and save storage space and computation cost by retaining only the upper triangular part of the matrix that ends up with $\begin{pmatrix} a & b & c & d & e & f \end{pmatrix}$ or lower triangular part $\begin{pmatrix} a & b & d & c & e & f \end{pmatrix}$. With symmetry property, the upper triangular part and lower triangular part of the matrix consist same information which is enough for constructing the feature vector for each image.

4. Train the SVM (Support Vector Machine) classifier.
   The SVM classifier will be trained by the training data set using available implementations in OpenCV or scikit-learn packages. In the following experiment, the SVM implementation from scikit-learn package is used for fitting training data and perform validation verification on the validation dataset.

5. Choose convolutional operators and SVM model.
   The above steps are repeated by a 'for' loop. Then the convolutional operators and svm model that generate the most accurate prediction on validation set are saved for prediction on test data.

6. Implement the SVM classfier on test images.
   With the saved parameters and model, we could classify the test images to known categories and compute the confusion matrix.

2 Results

2.1 Things to know

In the implementation of gram matrix approach,

1. Each input image is down-sampled to size 256×256.
2. The training images are split into training and validation data sets using 70% - 30 % splitting criteria.
3. The output of each channel is resized to 16×16.
4. An outer loop for $n = 10$ trials is executed for searching number of channels parameters that generates best prediction for the validation set.

2.2 Set of parameters

2.2.1 Number of channels

If the number of channels is too low that the length of feature vector would be short and the feature vectors are insufficient for classification, the model could be under-fitting. If the number of channels is too large then the model could be over-fitting which is also not good for the classification model. Therefore, the number of channels is determined to be $C = 65$ which yields highest accuracy of validation set.

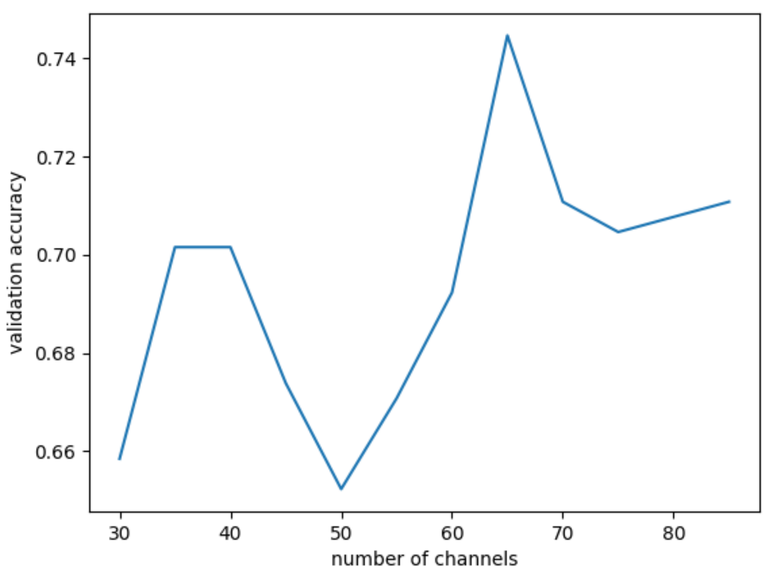

Figure 1: Number of channels vs. Accuracy of validation set

2.2.2 Convolutional operators

With number of channels $C = 65$, the convolutional operators shown in Appendix produce the best classification accuracy of 0.7446153846153846 on the validation images.

2.3 Confusion matrix

True label \Prediction label	cloudy	rain	shine	sunrise
cloudy	7	0	3	0
rain	1	9	0	0
shine	3	0	7	0
sunrise	2	0	2	6
Overall accuracy: 72.5 %				

2.4 Examples of wrong predictions

Figure 2: rain $\longrightarrow$ cloudy

Figure 3: shine $\longrightarrow$ cloudy

Figure 4: shine $\longrightarrow$ cloudy

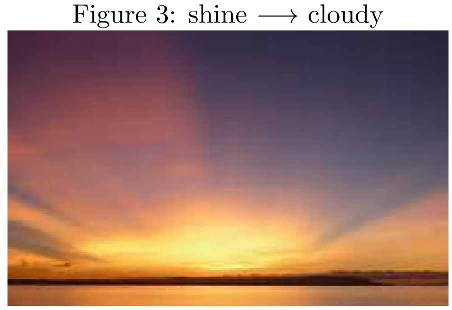

Figure 5: sunrise $\longrightarrow$ shine

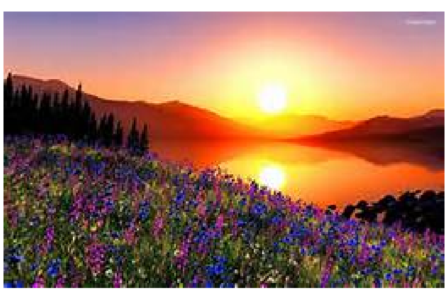

Figure 6: sunrise $\longrightarrow$ shine

Figure 7: sunrise $\longrightarrow$ cloudy

2.5 Observation

From the experiment, we can see that

1. While using the linear model to train the SVM classifier, the model could be over-fitting the training dataset if we want to obtain a decent validation accuracy. In addition, there is no evidence proving the corresponding relation between the validation accuracy and test accuracy. The test accuracy might be high even the validation accuracy is low, and the test accuracy might be low when the validation accuracy is decent.
2. While using nonlinear model for training, it's more possible to obtain close prediction accuracy for the validation dataset and test dataset.
3. The model accuracy on training set is highly related to the number of channels. The model accuracy on validation and test sets could be improved by running for more outer loops and find better combination of convolutional operators.
4. The other factors could impact the model accuracy include the pre-processing approach of data sets such as down-sampling and cropping.

3 Extra Credit: LBP feature and kNN classifier

3.1 Confusion Matrix and Overall Accuracy

True label \Prediction label	cloudy	rain	shine	sunrise
cloudy	9	0	1	0
rain	0	10	0	0
shine	1	0	7	2
sunrise	3	0	2	5
Overall accuracy: 77.5 %				

3.2 Observation

From the results, we can see that

1. Using the LBP feature extraction and kNN classifier produces better prediction accuracy on test set than using gram matrix and SVM classifier.
2. The LBP feature extraction and kNN classifier is computationally expensive and time consuming, especially when size of the data set is large. Using Gram matrix based feature and SVM classifier is more time efficient.

3.3 Wrong Predictions

Figure 8: cloudy $\longrightarrow$ shine

Figure 9: shine $\longrightarrow$ sunrise

Figure 10: shine $\longrightarrow$ cloudy

Figure 11: shine $\longrightarrow$ sunrise

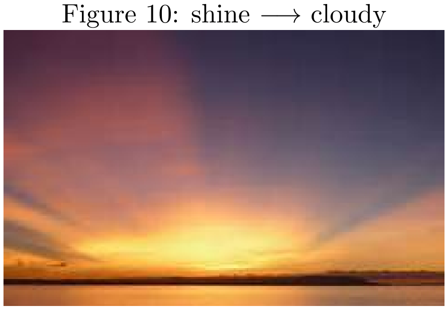

Figure 12: sunrise $\longrightarrow$ shine

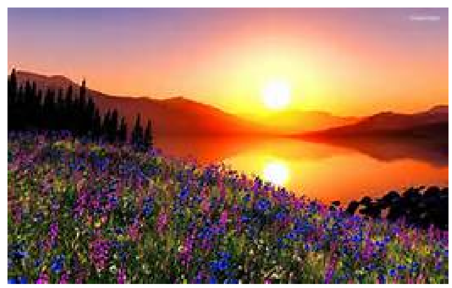

Figure 13: sunrise $\longrightarrow$ shine

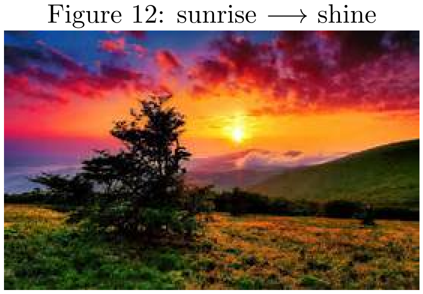

Figure 14: sunrise $\longrightarrow$ cloudy

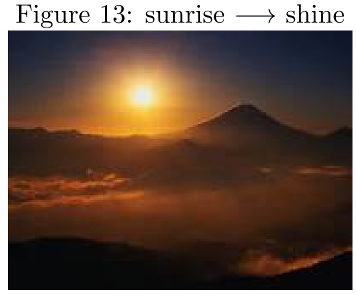

Figure 15: sunrise $\longrightarrow$ cloudy

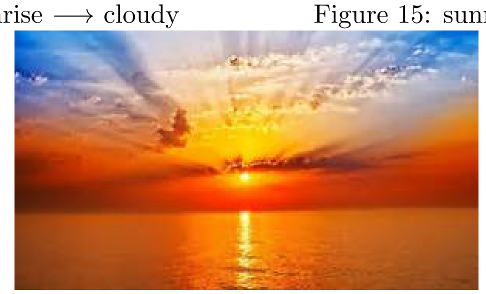

Figure 16: sunrise $\longrightarrow$ cloudy

4 Appendix

4.1 Convolutional operators

[[0.53253613 0.05658985 0.15278211] [-0.52041812 0.16456248 -0.26094664] [0.11174234 0.65193751 -0.88878566]]

[[0.66258559 -0.41886745 -0.44137226] [0.08548079 0.33913716 1.04424363] [-0.62023095 -0.5287931 -0.1221834]]

[[-0.91189696 -0.29252674 0.34945234] [0.75941637 0.95933828 0.62594986] [-0.77115613 -0.41696027 -0.30161674]]

[[-0.83411038 -0.43770523 -0.06854327] [0.18966146 0.37778996 0.63099505] [-0.11977972 0.45544169 -0.19374956]]

[[-0.28737136 -0.34560787 -0.12446293] [0.36023226 -0.33284779 0.63492329] [-0.83560567 0.53385441 0.39688566]]

[[-0.40357025 -0.13155131 -0.62534079] [-0.03067804 0.41104605 -0.12129768] [1.23700997 0.3119037 -0.64752166]]

[[-0.35979186 0.70145183 0.57002191] [-0.54954491 0.24791528 -0.94617968] [0.56298851 -0.55365285 0.32679176]]

[[-0.20792071 0.66900235 -0.73185335] [-0.03963164 -0.38407594 0.09496121] [0.29803294 -0.24610351 0.54758865]]

[[-0.8827014 0.86160704 0.24156058] [-0.76414843 0.65830839 -0.64927728] [0.78764165 0.55258706 -0.80557761]]

[[0.69947738 0.23775698 -0.03073137] [-0.69224787 0.18455376 -0.28685053] [-0.17305994 0.75271939 -0.6916178]]

[[0.48537743 -0.49437328 0.08430298] [-0.13389059 -0.96753844 0.62095785] [0.16206028 -0.01315073 0.25625451]]

[[0.56822524 -0.69209932 0.21084038] [-1.0053346 0.19726597 0.34895222] [0.40855487 0.4577278 -0.49413255]]

[[0.3975039 0.67291397 -0.09063058] [0.52890277 -1.06363658 0.09242085] [0.07362014 -1.092707 0.48161253]]

[[0.94959485 -0.17939093 -0.49794979] [0.2324832 0.12708352 -0.7403468] [-0.23163832 1.03947664 -0.69931237]]

[[0.46451624 0.45061667 -0.17711161] [-0.04208712 0.46454465 -0.55911473] [-0.41586706 -0.05355235 -0.13194467]]

[[0.75935166 -0.19195292 0.18613047] [0.25709787 -0.59107898 -0.24702898] [0.55725568 -0.07248617 -0.65728861]]

[[0.3500546 -0.17501218 0.74081045] [-0.15046428 -0.2320731 -1.02782322] [-0.53249271 0.34652662 0.68047382]]

[[0.6120352 0.1304518 0.56094657] [-0.20209044 0.21167537 -0.7025743] [-0.78058722 -0.62422944 0.79437248]]

[[-0.85564619 0.42551161 -0.12057648] [-0.45060644 0.21492057 -0.42213329] [-0.16417508 0.47311364 0.89959167]]

[[0.23235405 1.10050635 -0.4214552] [-0.67988344 -0.01924291 -0.36740836] [0.0090176 0.80821825 -0.66210633]]

[[-0.56849331 0.89256362 0.17367682] [0.17895547 -0.26763876 0.85907647] [-0.22873124 -0.55484995 -0.48455913]]

[[-0.77639344 -0.21529382 0.50940035] [0.20245933 0.19324141 -0.61012897] [-0.04058001 0.77810322 -0.04080807]]

[[0.92808533 0.14094506 -0.08130677] [0.0305648 -0.4672992 -0.80581493] [-0.63772403 0.09718819 0.79536155]]

[[-0.62053805 -0.27059664 0.50994946] [-0.73588106 -0.41090916 0.16146968] [0.781241 -0.03674192 0.62200669]]

[[-0.68452128 0.69571714 0.6103016] [0.37804694 0.72623455 -0.48533786] [-0.6741458 -0.45785976 -0.10843553]]

[[0.25550831 -0.57453671 0.14104804] [0.29650965 -0.37000825 -0.52714638] [0.52012883 -0.66500474 0.92350125]]

[[-0.66461278 -0.48511249 0.63026251] [1.13928366 -0.03645938 -0.54294947] [-0.15188444 -0.16814811 0.27962049]]

[[-0.54736031 -0.73496214 -0.49937121] [-0.70870399 0.87645856 0.96997573] [0.1169392 0.64361817 -0.116594]]

[[-2.66954636e-01 8.05245268e-01 -2.63324149e-01] [4.20254140e-01 -1.63451988e-01 -3.25440965e-04] [-4.33792778e-01 1.67922614e-01 -2.65573030e-01]]

[[0.37341338 0.887111 0.52112566] [-0.20476953 0.0271284 -0.74414956] [-0.00091805 -0.33526757 -0.52367374]]

[[0.57345262 -0.19179128 0.90340073] [-0.11521953 -0.35417201 -0.1732251] [-0.18918341 -0.7853795 0.33211748]]

[[-0.26690041 1.00509348 0.27388978] [-0.67312221 0.42920772 -0.61294575] [0.30518427 -0.70317914 0.24277227]]

[[0.03770368 0.38552269 -0.27754537] [-0.83482151 0.6657598 0.97531882] [-0.53817702 0.25405594 -0.66781705]]

[[-0.24019877 0.14026883 -0.22327158] [-0.26798231 0.33765455 0.01764036] [0.58403223 -0.29480282 -0.05334048]]

[[0.71315768 -0.32965065 0.79877308] [0.03675913 0.18276979 -0.38480294] [-0.1171218 -0.04340815 -0.85647615]]

[[-0.05402724 -0.47867595 -0.71436413] [0.02537104 -0.45069609 0.19646659] [0.82922983 0.57032052 0.07637544]]

[[0.40619974 -0.24205174 0.7127759] [0.76451686 -0.6227364 0.11541756] [-0.21454087 -0.89360563 -0.02597542]]

[[0.41820113 -0.35699802 -0.08584701] [-0.45327088 0.47871159 0.05138397] [0.31768518 0.3553779 -0.72524386]]

[[-0.14930926 -0.26232903 0.24964283] [-0.67713871 -0.42162822 -0.71338228] [1.17664183 0.25370205 0.54380079]]

[[0.08975402 0.58565675 0.08007927] [0.51006255 -0.11626526 -0.70392643] [-0.17494363 -0.60012819 0.32971093]]

[[0.775364 0.80811987 -0.65502957] [-0.49232013 0.74967684 -0.67203202] [-0.5684766 -0.40450328 0.4592009]]

[[0.32823906 0.10930638 0.35426106] [0.41520427 -0.43100685 -0.0225765] [-0.12804648 -0.34031933 -0.2850616]]

[[-0.76288355 -0.43789255 0.41378234] [-0.47337875 0.29652479 0.42619378] [-0.4691014 1.1007114 -0.09395607]]

[[-0.97770352 0.06700617 0.32138056] [-0.82580831 0.08109414 0.45955101] [-0.35728241 0.6148027 0.61695967]]

[[-0.50830083 0.52461988 -0.15670928] [0.07443337 0.38562878 -0.91688729] [-0.14563464 0.45279495 0.29005506]]

[[0.47092405 -0.1480969 -0.25258328] [1.00527126 0.18841802 -0.90026148] [0.74727056 -0.59682602 -0.51411621]]

[[1.02943945 -0.61155381 0.29274896] [-0.57488037 0.06191757 -0.1723666] [-0.6178679 0.88689307 -0.29433038]]

[[-0.71178919 -1.27770447 -0.05554407] [0.07060623 0.35687956 0.13930428] [0.57524018 0.32261115 0.58039633]]

[[-0.09769016 0.35587548 0.33264804] [0.794961 0.51254704 -1.03191809] [-0.69683117 0.44656381 -0.61615595]]

[[0.51040973 0.78413836 -0.28318618] [-0.66046617 0.82897561 0.05774421] [-0.56784724 -0.75776838 0.08800007]]

[[1.05327459 -0.19421313 0.78076617] [-0.71702849 0.05345754 -0.69109179] [-0.40424587 0.98077986 -0.86169887]]

[[-0.6202233 -0.62360975 0.44315305] [1.02968336 -0.6412476 0.22045012] [1.03384624 -0.36773748 -0.47431463]]

[[-0.17613918 -0.57360771 0.49523863] [1.19070417 0.04074183 -0.13810856] [0.1780803 -0.38668694 -0.63022254]]

[[-0.47226767 0.53531744 -0.96371767] [-1.05338865 0.20732474 0.35726349] [-0.26673515 0.88842543 0.76777803]]

[[-0.94446543 0.95849537 -0.07467832] [-0.07717423 -0.1517725 0.57426943] [-0.8378814 -0.16268692 0.715894]]

[[-0.98333778 -0.21449258 0.93246817] [-0.92739202 0.43287685 -0.16262893] [0.45757394 0.31405655 0.1508758]]

[[-0.1074397 -0.23735451 0.27022972] [-0.71004261 0.49890216 0.27911219] [0.05517112 -0.61035452 0.56177614]]

[[-0.49205304 0.7674982 0.34463916] [0.77578511 0.48880187 -0.9053763] [0.54994752 -0.94139985 -0.58784267]]

[[0.4443837 -0.08164884 -0.4612539] [0.48482856 0.50329119 0.51114597] [-0.0484201 -0.46027966 -0.89204691]]

[[-0.60516427 0.61651852 0.04909116] [-0.9596776 0.7866353 -0.63072675] [0.92019512 0.30147927 -0.47835075]]

[[0.6002239 0.09770715 -0.8789051] [0.48445265 0.76045439 -0.88449743] [0.08270349 -0.19048881 -0.07165025]]

[[-0.38055136 -0.09306064 0.73937122] [0.74111373 0.74641006 -0.38394396] [0.61231886 -1.09288967 -0.88876825]]

[[-0.49787014 -0.16549953 -0.30407073] [0.30585174 0.75717747 0.12705294] [-0.13239909 0.73586423 -0.82610688]]

[[0.63859411 0.03902199 -0.15498906] [-0.49369357 0.38783072 -0.15033977] [-0.38948429 0.45376728 -0.33070741]]

[[-0.17448987 -0.48658794 0.6592274] [-0.50097806 0.25187021 0.06971307] [-0.01694089 0.55676477 -0.35857869]]

4.2 Source Code

```
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from sklearn import svm, metrics
from sklearn.metrics import confusion_matrix
import pickle


# Load images
def load_img(img_class, img_path):
    # initialization
    train_dataset = []
    test_dataset = []
    testing_labels = []
    # load training data
    for facade in img_class:
        folder = img_path + "/training/{}".format(facade)
        for filename in os.listdir(folder):
            img = cv2.imread(os.path.join(folder, filename))
            if img.shape[2] == 3:
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (256, 256), interpolation=cv2.INTER_AREA)
            if img is not None:
                train_dataset.append(img)
    # load testing data
    folder = img_path + "/testing"
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder, filename))
        if img.shape[2] == 3:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (256, 256), interpolation=cv2.INTER_AREA)
        if img is not None:
            test_dataset.append(img)
        if 'cloudy' in filename:
            testing_labels.append(0)
        if 'rain' in filename:
            testing_labels.append(1)
        if 'shine' in filename:
            testing_labels.append(2)
        if 'sunrise' in filename:
            testing_labels.append(3)
    a = np.concatenate(([0] * 290, [1] * 204), axis=None)
    b = np.concatenate(([2] * 242, [3] * 347), axis=None)
    training_labels = np.concatenate((a, b), axis=None)
```

```
    return train_dataset, test_dataset, training_labels, testing_labels


img_class = ["cloudy", "rain", "shine", "sunrise"]
img_path = "/Users/qiuchen/PycharmProjects/trial folders/hw8/imagesHW8"
train_data, test_data, training_labels, test_labels = load_img(img_class, img_path)

# split training set into training and validation set
training_set, validation_set, training_label, validation_label = train_test_split(
    train_data, training_labels,

    test_size=0.30, random_state=0)


def compute_conv_kernel(M=3):
    # Generate random uniformly distributed convolutional operators
    kernel = np.random.uniform(low=-1, high=1, size=(M, M))

    return np.asarray(kernel - np.sum(kernel) / (M * M))


def compute_convd_vec(dataset, conv_operator, **kwargs):
    # convolve the image with convolutional operator and downsample the output
    dsample_sz = kwargs.pop("downsample_size", 16)
    output = np.zeros((len(dataset), dsample_sz * dsample_sz), dtype=float)
    for idx, img in enumerate(dataset):
        feature = cv2.resize(cv2.filter2D(img, -1, conv_operator), (dsample_sz,
    dsample_sz), interpolation=cv2.INTER_AREA).flatten()
        output[idx] = feature

    return output


def get_gram_mat(vector):
    """
    Generate Gram-matrix based C^2 / 2 dimensional feature vectors
    :param vector: vector with size: (len(dataset), 256)
    :return: C^2 / 2 dimensional feature vectors
    """

    output = np.zeros((len(vector), Nchannels, Nchannels))
    gram_mat = []
    for i in range(len(vector)):
        output[i] = np.dot(vector[i], vector[i].transpose())
        gram_mat.append(output[i][np.triu_indices(Nchannels, k=0)])

    return np.asarray(gram_mat)


# perform the classification task using Gram matrix
Ntrials = 10
Nchannels = 65
accuracy = 0
for j in range(Ntrials):
    # Compute C different KxK feature maps
    dsample_sz = 16
    train_temp = np.zeros((len(training_set), Nchannels, dsample_sz * dsample_sz),
    dtype=float)
    validation_temp = np.zeros((len(validation_set), Nchannels, dsample_sz *
    dsample_sz), dtype=float)
    kernel_list = []
    for c in range(Nchannels):
        kernel = compute_conv_kernel(M=3)
```

```python
        kernel_list.append(kernel.flatten())
        train_temp[:, c, :] = compute_convd_vec(training_set, conv_operator=kernel, downsample_size=dsample_sz)
        validation_temp[:, c, :] = compute_convd_vec(validation_set, conv_operator=kernel, downsample_size=dsample_sz)
    # Generate Gram-matrx-based C^2/2 dimensional feature vectors for both training and validation images
    train_gram_mat = get_gram_mat(train_temp)
    validation_gram_mat = get_gram_mat(validation_temp)
    # Train a SVM classifier using Opencv or scikit-learn
    clf = svm.SVC(kernel='linear')
    clf.fit(train_gram_mat, training_label)
    train_predictions = clf.predict(train_gram_mat)
    print('train_prediction :', metrics.accuracy_score(training_label, train_predictions))
    # Evaluate the classification accuray on validation set, check if new features improve the accuracy
    clf_predictions = clf.predict(validation_gram_mat)
    # print(clf_predictions)
    validation_accuracy = metrics.accuracy_score(validation_label, clf_predictions)
    print('validation_prediction :', validation_accuracy)
    # Save best convolutional operators and SVM model in .xml file format for reproduciability
    if validation_accuracy > accuracy:
        best_kernel = kernel_list
        accuracy = metrics.accuracy_score(validation_label, clf_predictions)
        pkl_filename = "svm_model.pkl"
        with open(pkl_filename, 'wb') as file:
            pickle.dump(clf, file)


# Test image
test_temp = np.zeros((len(test_data), Nchannels, dsample_sz * dsample_sz), dtype=float)
for c in range(Nchannels):
    kernel = np.asarray(best_kernel[c]).reshape((3, 3))
    test_temp[:, c, :] = compute_convd_vec(test_data, conv_operator=kernel, downsample_size=dsample_sz)
test_gram_mat = get_gram_mat(test_temp)
# Compute the confusion matrix for test images using best model parameters
filename = "svm_model.pkl"
model = pickle.load(open(filename, 'rb'))
test_predictions = model.predict(test_gram_mat)
test_accuracy = metrics.accuracy_score(test_labels, test_predictions)
print('Test Accuracy =', test_accuracy)
print(confusion_matrix(test_labels, test_predictions))
```

hw8.py