Purdue University

ECE 661 Computer Vision

Homework 8

Submission: Arjun Kramadhati Gopi

Email: akramadh@purdue.edu

Tasks for this homework

This homework requires us to build an image classification algorithm which is based on the Gram Matrix representation of image textures. This will be a deep learning based approach at classifying images. To achieve this objective, we will be seeing to that the following two algorithms are in place:

1. Gram matrix based characterization of image textures.
2. Support Vector Machine implementation to classify the images based on the gram matrices.

Gram Matrix texture characterisation

We basically convolve the image with C different convolutional operators. The operators which are also called as kernels are M x M in size. Usually we will have the kernel size as M=3 which makes it a 3x3 operator. We construct these kernels by populating them with random weights. The kernels are built using the following two conditions or standards:

- The weights are random floating point variables which are in the range [-1,1]
- The weights in the kernel all add upto 0.

To make the kernels add up to zero we will be looking at the mean value of the kernel. If the mean of the kernel becomes zero that means that all the weights in the kernel will add up to zero.
Using these C kernels we will convolve the image to get C different channels. Usually we set C=3. This makes the output of the convolutions as a three channel output. Therefore, after convolving all the three kernels on the image, we get three different channels.
Additionally, we will be downsampling the images into a K x K array. Therefore, if the input image size is 256 x 256, we will be downsampling it into a K x K image. For all our purposes in this homework, we set K = 16. Therefore, we get output channels with each of them sized at 16 X 16.
For each convolution, we vectorise the output by a 256-element vector. After C convolutions, we will end up with three unique 256-element vectors. To get one single vector representation of the entire image, we take inner products of the C different vectors.
The resulting feature representation matrix is of size C x C. This will be a symmetric matrix. Therefore, it is sufficient to retain just the upper triangle part of the C x C matrix. This is the fundamental underlying procedure of generating gram matrices for each image.

Support Vector Machine (SVM)

In the previous section we saw how we generate a C x C gram matrix which represents the texture features of an image. If we use this information to represent each image in a $C^2/2$ representational space, we can classify images using the SVM implementation. So what is SVM?
Support Vector Machine based classification has its roots in the Statistical Learning Theory. SVM is popular because of the fact that it is very reliable and accurate even on small training data sets.

If the data point is a p-dimensional vector, the aim is to separate such points using a p-1 dimensional hyperplane. The best separation occurs when we find the hyperplane which represents the largest separation, or largest margin between two classes. So, larger the margin of separation between the classes, lower the generalization error.

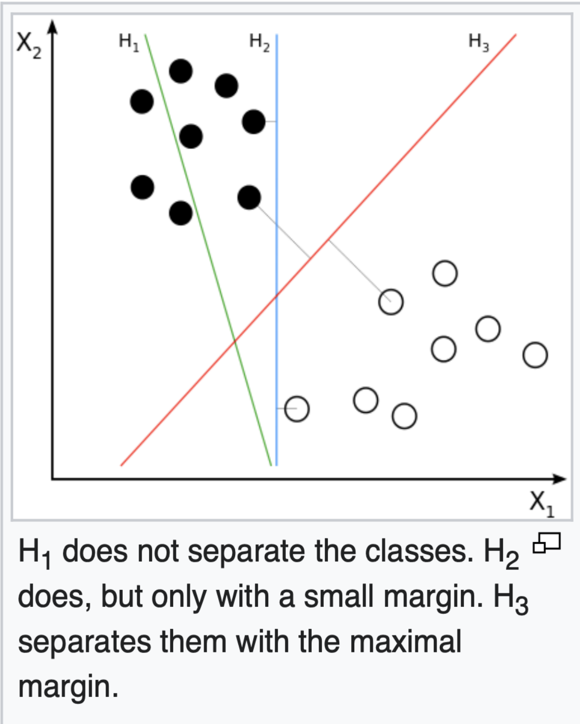

Figure 1: **Image source: wikipedia.com**

In the image above we can see the general idea behind a linear classifier using a SVM. Our aim is to find the best hyperplane which separates the classes. In the image, we see that the H_3 is the best fit.
Using the gram matrix based $C^2/2$ dimensional feature vector, we train the SVM classifier. For this homework, we will be using an inbuilt SVM classifier in the OpenCV or the Scikit-learn libraries in Python.

Result and analysis

After trying a lot of iterations of different values for:

1. C value for the number of convolution channels
2. M value for the kernel/operator size
3. K value for the downsampling size

Here are the summary of the results that I have obtained
Best results parameters:

- C = 30
- M = 11

- K = 16

For the above parameters I obtained the following accuracy scores:

- Validation accuracy: 60.1%
- Testing accuracy: 80.0%

Here is the summarised result sheet for the best result:

```
SVM Training complete.
------------------------------------
------------------------------------
Validation complete...
Validation accuracy score: 60.147601476014756%
Printing confusion matrix
[[51.  0.  5.  0.]
 [ 1. 21. 34.  9.]
 [ 4.  7. 76.  3.]
 [11.  6. 28. 15.]]
------------------------------------
------------------------------------
Testing complete...
Testing accuracy score: 80.0%
Printing confusion matrix
[[ 9.  0.  1.  0.]
 [ 0.  5.  4.  1.]
 [ 0.  0. 10.  0.]
 [ 0.  1.  1.  8.]]
------------------------------------
------------------------------------
Parameter summary
C value for the number of channels: 30
M value for the kernel size: 11
K value for the downsampling: 16
------------------------------------
------------------------------------
```

Figure 2: **Summary sheet - Best Results**

While trying to find the best parameters, here were the general observations made:

- Increasing the downsampling parameter while keeping the other two parameters constant gave poor results.

```
SVM Training complete.
-----------------------------------
-----------------------------------
Validation complete...
Validation accuracy score: 43.91143911439114%
Printing confusion matrix
[[42. 12.  2.  0.]
 [18. 29. 17.  1.]
 [16. 26. 48.  0.]
 [22. 24. 14.  0.]]
-----------------------------------
-----------------------------------
Testing complete...
Testing accuracy score: 40.0%
Printing confusion matrix
[[7. 2. 0. 1.]
 [5. 5. 0. 0.]
 [2. 3. 4. 1.]
 [6. 4. 0. 0.]]
-----------------------------------
-----------------------------------
Parameter summary
C value for the number of channels: 10
M value for the kernel size: 3
K value for the downsampling: 32
-----------------------------------
-----------------------------------
```

Figure 3: **Summary sheet - Trying to increase K value**

- Decreasing the downsampling paramter while keeping the other two parameters constant gives us slightly better results than the above modification.

```
SVM Training complete.
-----------------------------------
-----------------------------------
Validation complete...
Validation accuracy score: 54.24354243542435%
Printing confusion matrix
[[49.  3.  0.  4.]
 [ 8. 38. 18.  1.]
 [11. 17. 58.  4.]
 [20. 23. 15.  2.]]
-----------------------------------
-----------------------------------
Testing complete...
Testing accuracy score: 55.00000000000001%
Printing confusion matrix
[[10.  0.  0.  0.]
 [ 2.  7.  1.  0.]
 [ 1.  4.  5.  0.]
 [ 6.  3.  1.  0.]]
-----------------------------------
-----------------------------------
Parameter summary
C value for the number of channels: 10
M value for the kernel size: 3
K value for the downsampling: 8
-----------------------------------
-----------------------------------
```

Figure 4: **Summary sheet - Trying to decrease K value**

- Increasing the C value, it is observed that there is a gradual increase in the accuracy scores. But, beyond a certain C value, I found that the accuracy change becomes stable and tapers to zero.

```
SVM Training complete.
------------------------------------
------------------------------------
Validation complete...
Validation accuracy score: 53.874538745387454%
Printing confusion matrix
[[45.  7.  4.  0.]
 [ 7. 32. 26.  0.]
 [12.  9. 69.  0.]
 [16. 22. 22.  0.]]
------------------------------------
------------------------------------
Testing complete...
Testing accuracy score: 57.49999999999999%
Printing confusion matrix
[[10.  0.  0.  0.]
 [ 0.  6.  4.  0.]
 [ 0.  3.  7.  0.]
 [ 3.  4.  3.  0.]]
------------------------------------
------------------------------------
Parameter summary
C value for the number of channels: 3
M value for the kernel size: 3
K value for the downsampling: 16
------------------------------------
------------------------------------
```

Figure 5: **Summary sheet - C value 03**

```
SVM Training complete.
------------------------------------
------------------------------------
Validation complete...
Validation accuracy score: 60.147601476014756%
Printing confusion matrix
[[51.  0.  5.  0.]
 [ 1. 21. 34.  9.]
 [ 4.  7. 76.  3.]
 [11.  6. 28. 15.]]
------------------------------------
------------------------------------
Testing complete...
Testing accuracy score: 80.0%
Printing confusion matrix
[[ 9.  0.  1.  0.]
 [ 0.  5.  4.  1.]
 [ 0.  0. 10.  0.]
 [ 0.  1.  1.  8.]]
------------------------------------
------------------------------------
Parameter summary
C value for the number of channels: 30
M value for the kernel size: 11
K value for the downsampling: 16
------------------------------------
------------------------------------
```

Figure 6: **Summary sheet - C value 30**

```
SVM Training complete.
------------------------------------
------------------------------------
Validation complete...
Validation accuracy score: 60.147601476014756%
Printing confusion matrix
[[47.  0.  7.  2.]
 [ 1. 27. 29.  8.]
 [ 4.  8. 76.  2.]
 [11.  8. 28. 13.]]
------------------------------------
------------------------------------
Testing complete...
Testing accuracy score: 80.0%
Printing confusion matrix
[[ 9.  0.  1.  0.]
 [ 0.  5.  3.  2.]
 [ 0.  0. 10.  0.]
 [ 0.  1.  1.  8.]]
------------------------------------
------------------------------------
Parameter summary
C value for the number of channels: 40
M value for the kernel size: 11
K value for the downsampling: 16
------------------------------------
------------------------------------
```

Figure 7: **Summary sheet - C value 40**

- For the M value, I found that very high M values and very low M values will give poor accuracy. I found that a M value of 11 which is slightly higher than the half of the kvalue gives consistent and stable accuracy scores.

```
SVM Training complete.
------------------------------------
------------------------------------
Validation complete...
Validation accuracy score: 53.50553505535055%
Printing confusion matrix
[[42.  8.  4.  2.]
 [ 3. 28. 34.  0.]
 [ 7.  8. 75.  0.]
 [12. 23. 25.  0.]]
------------------------------------
------------------------------------
Testing complete...
Testing accuracy score: 50.0%
Printing confusion matrix
[[7. 3. 0. 0.]
 [0. 7. 3. 0.]
 [0. 4. 6. 0.]
 [2. 5. 3. 0.]]
------------------------------------
------------------------------------
Parameter summary
C value for the number of channels: 7
M value for the kernel size: 3
K value for the downsampling: 16
------------------------------------
------------------------------------
```

Figure 8: **Summary sheet - M value 03**

```
SVM Training complete.
-----------------------------------
-----------------------------------
Validation complete...
Validation accuracy score: 59.77859778597786%
Printing confusion matrix
[[51.  1.  3.  1.]
 [ 6. 33. 20.  6.]
 [15. 10. 62.  3.]
 [17.  6. 21. 16.]]
-----------------------------------
-----------------------------------
Testing complete...
Testing accuracy score: 55.00000000000001%
Printing confusion matrix
[[8. 0. 2. 0.]
 [1. 3. 5. 1.]
 [1. 1. 6. 2.]
 [3. 1. 1. 5.]]
-----------------------------------
-----------------------------------
Parameter summary
C value for the number of channels: 15
M value for the kernel size: 15
K value for the downsampling: 16
-----------------------------------
-----------------------------------
```

Figure 9: **Summary sheet - M value 15**

```
SVM Training complete.
-----------------------------------
-----------------------------------
Validation complete...
Validation accuracy score: 57.93357933579336%
Printing confusion matrix
[[45.  0.  9.  2.]
 [ 0. 23. 35.  7.]
 [ 5.  6. 75.  4.]
 [ 9. 10. 27. 14.]]
-----------------------------------
-----------------------------------
Testing complete...
Testing accuracy score: 75.0%
Printing confusion matrix
[[ 8.  0.  2.  0.]
 [ 0.  5.  4.  1.]
 [ 0.  0. 10.  0.]
 [ 0.  1.  2.  7.]]
-----------------------------------
-----------------------------------
Parameter summary
C value for the number of channels: 20
M value for the kernel size: 11
K value for the downsampling: 16
-----------------------------------
-----------------------------------
```

Figure 10: **Summary sheet - M value 11 - which gives the best results**

In summary we have the following observations:

1. C value of 30 produced the best results. Increasing the C value gives better results but only up until a certain C value, beyond which there is no significant change in accuracy.

2. M value of 11 produced the best results. I noticed that a M value which is slightly higher than the half of the K value usually gives the best results. High and low M value give poor results.

3. K value is also similar to the case of the M value. High K values and very low K values give bad results. For our case, I found that a K value of 16 gives us the best results.

Final observation on the overall accuracy

```
Testing accuracy score: 80.0%
Printing confusion matrix
[[ 9.  0.  1.  0.]
 [ 0.  5.  4.  1.]
 [ 0.  0. 10.  0.]
 [ 0.  1.  1.  8.]]
```

Figure 11: **Confusion matrix for the best result**

- C value = 30, M value = 11, K value = 16 gives the best results
- Testing accuracy score of 80%
- Cloudy class predictions 9/10 correct
- Rain class predictions 5/10 correct
- Shine class predictions 10/10 correct
- Sunrise class predictions 8/10 correct
- C value = 10, M value = 03, K value = 32 gave the worst accuracy of 40%

Source code

```
"""
Computer Vision - Purdue University - Homework 8

```

```python
Author : Arjun Kramadhati Gopi, MS-Computer & Information Technology, Purdue University.
Date: Oct 19, 2020


[TO RUN CODE]: python3 deeplearnclassifier.py
Output:
    [labels]: Predictions for the input images in the form of a confusion matrix.
"""
import re
import glob
import pickle
import cv2 as cv
import numpy as np
from sklearn import svm
from scipy import signal
from sklearn.model_selection import train_test_split


class Gramclassify:

    def __init__(self, trainingset_path, testingset_path, cvalue = 30, mvalue = 11, kvalue = 16):
        """
        Initialise the gram classify object with the parameters
        :param trainingset_path: Path to the training data set
        :param testingset_path: Path to the testing data set
        :param cvalue: Value for the number of channels for convolution
        :param mvalue: Value for the kernel size of mvalue X mvalue
        :param kvalue: Value for the kvalue X kvalue downsampling
        """
        np.random.seed(0)
        self.cvalue = cvalue
        self.mvalue = mvalue
        self.kvalue = kvalue
        self.operators = None
        self.pattern = re.compile("([a-zA-Z]+)([0-9]+)")
        self.training_path = glob.glob(trainingset_path)
        self.testing_path = glob.glob(testingset_path)
        self.training_images_path, self.validation_images_path = train_test_split(self.training_path, shuffle=True,


        self.prepare_convolutional_operators()
        print('Initialization complete')
```

```

    def get_label_string(self, element):
        """
        Since the data has just one directory where images of all
            the classes
        are present, we will need to mine for the label or the
           class name
        from the file name. This function returns the class or
           the label name
        from the given image path
        :param element: Image path
        :return: Return image label or class
        """
        return self.pattern.match(element.split('/')[-1].split
           ('.')[0]).groups()[0]

    def downsample_vectorise(self, image):
        """
        Downsample the convolution output into kvalue X kvalue
           size.
        Next vectorise the downsampled array.
        :param image: Image channel to be downsampled
        :return: Returns the vector representation of the texture
            for that channel
        """
        return np.reshape(image[::self.kvalue, ::self.kvalue, :],
            (-1, self.cvalue))

    def prepare_convolutional_operators(self):
        """
        This function prepares the convolution operators which we
            will be
        using to convolve the image into C different channels.
        :return: Set the operator to the global operator value
        """
        operators = np.zeros((self.mvalue, self.mvalue, self.
           cvalue), np.float)
        for index in range(operators.shape[2]):
            operators[:, :, index] = np.random.rand(self.mvalue,
               self.mvalue) * 2 - 1
            operators[:, :, index] -= np.mean(operators[:, :,
               index])
        self.operators = operators

    def generate_gram_matrix(self, image):
        """
        This function generates the gram matrix for the given
           image.
        :param image: Input image for which we need the gram
           matrix
        :return: Return the gram matrix
        """
        if len(image.shape) > 2:
```

```python
            image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
        convolved_image = np.zeros((image.shape[0] - self.mvalue + 1, image.shape[1] - self.mvalue + 1, self.cvalue), np.float)
        for channel in range(self.cvalue):
            convolved_image[:, :, channel] = signal.convolve(image, self.operators[:, :, channel], mode='valid')
        vector = self.downsample_vectorise(convolved_image)
        gram_matrix = np.matmul(vector.T, vector)
        gram_matrix = gram_matrix/ np.sum(gram_matrix)
        return gram_matrix.reshape(1, -1)

    def dump_data(self, classes, labels, grams):
        """
        We dump the data so that we need not train and generate the model every time we need to
        predict classes for new images.
        :param classes: List of the class names
        :param labels: List of the label names
        :param grams: List of all the gram matrices
        :return: enumerated label list used for prediction and also the dictionary needed for the same task
        """
        classdict = dict()
        enumerated_labels = []
        for index, element in enumerate(classes):
            classdict[element] = index
        for label in labels:
            enumerated_labels.append(classdict[label])
        pickle.dump(enumerated_labels, open('enumerated_labels.obj', 'wb'))
        pickle.dump(classes, open('classes.obj', 'wb'))
        pickle.dump(classdict, open('classdict.obj', 'wb'))
        pickle.dump(labels, open('labels.obj', 'wb'))
        pickle.dump(grams, open('grams.obj', 'wb'))
        return enumerated_labels, classdict

    def construct_confusion_matrix(self, image_path, classes, dictionary, model):
        """
        We construct the confusion matrix given the image paths, classes
        and other necessary parameters
        :param image_path: Image paths
        :param classes: List of the classes
        :param dictionary: Dictionary of the classes and their indices
        :param model: SVM model needed to perform the predictions
        :return: Return the confusion matrix and the accuracy scores.
        """
        confusion_matrix = np.zeros((len(classes), len(classes)),
```

```
            np.float)
        for element in image_path:
            grayimage = cv.resize(cv.imread(element, 0), (300, 200))
            label = self.get_label_string(element)
            gram_matrix = self.generate_gram_matrix(grayimage)
            prediction = model.predict(gram_matrix)
            label_enumerate = dictionary[label]
            confusion_matrix[label_enumerate, prediction] += 1
        return confusion_matrix, np.trace(confusion_matrix)/np.sum(confusion_matrix)

    def construct_representational_space(self):
        """
        This function does the following:
        1) Constructs the C^2/2 dimensional representational space.
        2) Train the Support Vector Machine
        3) Validate the training
        4) Test the trained model by making predictions of new images
        :return: None. Prints the final result summary.
        """
        classes = []
        labels = []
        grams = np.zeros((len(self.training_images_path), self.cvalue*self.cvalue), np.float)
        for index, element in enumerate(self.training_images_path):
            print("Process complete: " + str(index/len(self.training_images_path)))
            grayimage = cv.resize(cv.imread(element, 0), (300, 200))
            label = self.get_label_string(element)
            if label not in classes:
                classes.append(label)
            gram_matrix = self.generate_gram_matrix(grayimage)
            grams[index] = gram_matrix
            labels.append(label)
        enumerated_labels, classdict = self.dump_data(classes, labels, grams)
        model = svm.SVC(kernel='poly')
        model.fit(grams, enumerated_labels)
        pickle.dump(model, open('model.pkl', 'wb'))
        confusion_matrix, accuracy = self.construct_confusion_matrix(self.validation_images_path,classes, classdict, model)
        print("SVM Training complete.")
        print("-----------------------------------")
        print("-----------------------------------")
        print('Validation complete...')
        print('Validation accuracy score: ' + str(accuracy * 100) + "%")
```

```
        print('Printing confusion matrix')
        print(confusion_matrix)
        confusion_matrix, accuracy = self.construct_confusion_matrix(self.testing_path, classes, classdict, model)
        print("-----------------------------------")
        print("-----------------------------------")
        print('Testing complete...')
        print('Testing accuracy score: ' + str(accuracy * 100) + "%")
        print('Printing confusion matrix')
        print(confusion_matrix)
        print("-----------------------------------")
        print("-----------------------------------")
        print("Parameter summary")
        print("C value for the number of channels: " + str(self.cvalue))
        print("M value for the kernel size: " + str(self.mvalue))
        print("K value for the downsampling: " + str(self.kvalue))
        print("-----------------------------------")
        print("-----------------------------------")


if __name__ == "__main__":
    """
    Code begins here
    """
    tester = Gramclassify('./imagesDatabaseHW8/training/*','./imagesDatabaseHW8/testing/*', )
    tester.construct_representational_space()
```