ECE 661: Homework 8

Fall 2020

Wenrui Li

Due Data: Oct 26, 2020

1 Introduction

The goal of this homework is,

- Construct a Gram Matrix for an image by convolving the image with C different convolutional operators.
- Retain its upper triangular part and use Support Vector Machine for texture classification.
- A outdoor multi-class weather dataset is given. The last 10 images from each category are used to create the test for this homework.
- We should find proper parameters for both Gram Matrix and SVM.

2 MethologMethodology and Implementationy

Since C different $M \times M$ convolutional operators generated randomly, we need n iterations to find the best $M \times M$ convolutional operators and the best SVM model. For each iteration, we will apply following steps. For each step, we will explain how does it implement and why doing those step can satisfy conditions.

2.1 Compute C different $K \times K$ feature maps

2.1.1 Operators generation

I generate C different $K \times K$ feature maps with gaussian distribution.

Assume $X_{M,M,3,C} \sim N(0,1)$, where 3 represents that we use color channels in this homework. Also, kernel size $M = 3$. Totally, we need $3 \times C$ filters to generate C different $K \times K$ feature maps.

For each filter, $F \in R_{M \times M}$,

$$F'_{i,j} = (F_{i,j} - \overline{F}) / \max_{i,j} |F_{i,j}| \tag{1}$$

where $\overline{F}$ is mean of F, $F'_{i,j}$ is the final filter element which satisfies

- The weights in each operator add up to zero.
- each weight is drawn randomly form the interval $[-1, 1]$.

2.1.2 Filtering

In this homework, the size of the output with each channel is 32×32.

- Find out the minimum index t of an image.
- Crop the middle square $t \times t$ from the image.
- Resize the $t \times t$ cropped image to 34×34.
- Convolve generated operators with this cropped image. Since we use valid padding, the size of the output with each channel is 32×32. The size of the output feature maps is $32 \times 32 \times C$.

2.2 Gram Matrix

2.2.1 Gram Matrix generated from feature map

Now, we have feature maps with size of $32 \times 32 \times C$. Next, we are going to use it to generate the Gram Matrix.

- Flatten each feature map to a vector, we get features with size of $1024 \times C$. We assign these features to $a_1, a_2, ..., a_C$
- A Gram matrix of vectors $a_1, a_2, ..., a_C$ is a matrix G
  - $s.t. G =< a_i, a_j > \forall i, j$
  - If vectors $a_1, a_2, ..., a_C$ are columns of a matrix A, then
  - $G = A^T A$
- Now, we obtain the Gram Matrix with size of $C \times C$

2.2.2 Generate $C \times (C+1)/2$ dimensional feature vectors

- Crop out an upper triangular matrix from the Gram matrix.
- List out all the nonzeros elements from the upper triangular matrix.
- Then, we got $C \times (C+1)/2$ dimensional feature vectors.

2.3 Support Vector Machine(SVM)

In this section, we will introduce how to use SVM to classify weather images. Since we use OpenCV's SVM to do classification in this homework, we will introduce a breif idea of the solution to multi-class classification with OpenCV's SVM.

OpenCV's SVM implements multi-class classification in the form of one-vs-one problems. Assuem we have N classes to classify, OpenCV's SVM will do following steps,

- Each class is compared with each other class, so there is $N(N-1)/2$ decision functinos.
- Each comparison has a winner class, and such class receives one vote be the final best match.

Now, we have a brief overview on how OpenCV's SVM deal with multi-class classification. Next, we will introduce the core optimization algorithm we used in this homework, C-SVM with linear kernel.

Given training vectors $x_i \in \mathbb{R}^p$, i $= 1, \ldots,$ n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by sign $\left(w^T \phi(x) + b\right)$ is correct for most samples.

SVC solves the following primal problem:

$$\begin{gathered} \min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i \left(w^T \phi\left(x_i\right) + b\right) \geq 1 - \zeta_i \\ \zeta_i \geq 0, i = 1, \ldots, n \end{gathered} \tag{2}$$

Intuitively, we're trying to maximize the margin (by minimizing $\|w\|^2 = w^T w$), while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i \left(w^T \phi\left(x_i\right) + b\right)$ would be ≥ 1 for all samples, which indicates a perfect prediction. However, problems are often not always perfectly separable with a hyperplane, so we allow some samples to be at a distance ζ_i from their correct margin boundary. The penalty term C controls the strengh of this penalty.

Rewrite the condition $y_i \left(w^T \phi\left(x_i\right) + b \right) \geq 1 - \zeta_i$,

$$\zeta_i \geq 1 - y_i \left(w^T \phi\left(x_i\right) + b \right) \tag{3}$$

Combine the above inequality with $\zeta_i \geq 0$, rewrite equation (2)

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1} \max\left(0, 1 - y_i \left(w^T \phi\left(x_i\right) + b \right)\right) \tag{4}$$

where we make use of the hinge loss. This is the form that is directly optimized by LinearSVC, but unlike the dual form, this one does not involve inner products between samples, so the famous kernel trick cannot be applied. This is why only the linear kernel is supported by LinearSVC ($\phi\left(x_i\right)$ is the identity function).

3 Result

In this section, we will compare the performance for the different values of C. First, we will introduce hyper-parameters in operator generations and SVM classification. Second, we will compare all confusion matrix and overall accuracy.

3.1 Hyper-parameters

For each different Filter channel size C, we try $n = 100$ times to obtain the best performance on validate set. We use 760 out of 1085 images in training set for training and 325 out of 1085 images for validation.

Filter generation

Kernel size M	3
Filter channel size C	[16,24,32,40,64]
Input channel size	Color 3
Feature Map shape	[32,32,C]

C-SVM with linear kernel

C	0.01
Maximum iteration	20000
Terminal Criteria	1e-7

3.2 Comparison on different Filter channel size C

In this section, we compare performance on different Filter channel size C.

$C = 16$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	7	0	3	0
	rain	2	6	2	0
	shine	5	0	4	1
	sunrise	0	0	1	9
Validate Accuracy	81.85%(266/325)				
Overall Accuracy	65%(26/40)				
Wrong Prediction	'shine248.jpg', 'cloudy297.jpg', 'rain208.jpg', 'rain211.jpg', 'cloudy299.jpg', 'shine244.jpg', 'shine246.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'sunrise353.jpg', 'rain215.jpg', 'shine253.jpg', 'shine247.jpg'				

$C = 24$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	6	1	3	0
	rain	0	8	2	0
	shine	0	2	7	1
	sunrise	0	0	1	9
Validate Accuracy	82.77%(269/325)				
Overall Accuracy	75%(30/40)				
Wrong Prediction	'cloudy293.jpg', 'shine249.jpg', 'cloudy291.jpg', 'cloudy295.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'sunrise353.jpg', 'rain215.jpg', 'shine247.jpg'				

$C = 32$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	8	0	2	0
	rain	0	8	2	0
	shine	2	0	7	1
	sunrise	0	0	0	10
Validate Accuracy	85.85%(279/325)				
Overall Accuracy	82.5%(33/40)				
Wrong Prediction	'cloudy291.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'rain215.jpg', 'shine253.jpg', 'shine247.jpg'				

$C = 40$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	9	0	1	0
	rain	0	8	2	0
	shine	3	0	6	1
	sunrise	0	0	0	10
Validate Accuracy	86.77%(282/325)				
Overall Accuracy	82.5%(33/40)				
Wrong Prediction	'shine249.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'rain215.jpg', 'shine253.jpg', 'shine247.jpg'				

$C = 64$

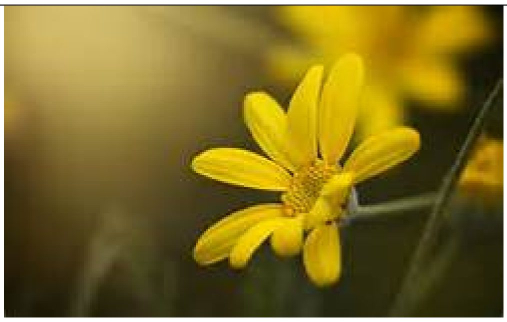
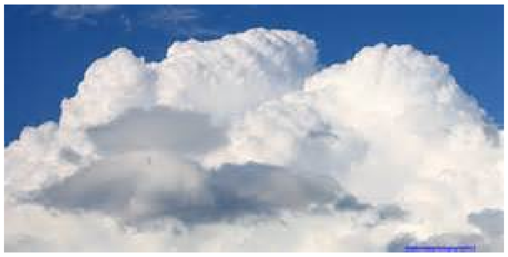
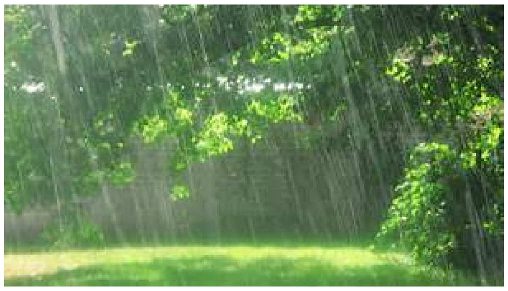
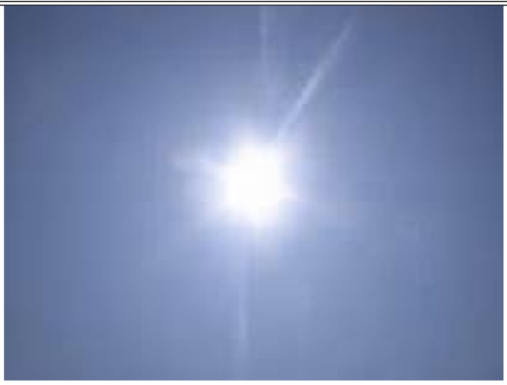
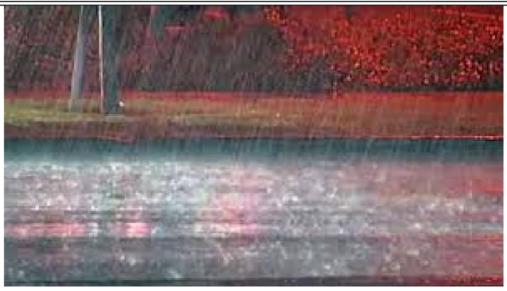
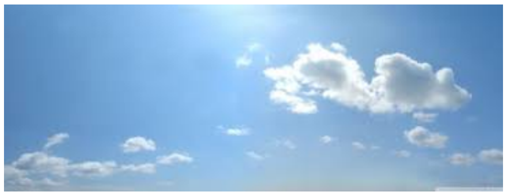
		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	6	1	3	0
	rain	0	7	3	0
	shine	1	0	8	1
	sunrise	0	0	1	9
Validate Accuracy	84.62%(275/325)				
Overall Accuracy	75%(30/40)				
Wrong Prediction	'shine249.jpg', 'cloudy291.jpg', 'cloudy295.jpg', 'cloudy296.jpg', 'rain211.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'sunrise353.jpg' 'rain215.jpg'				

From the above result, we can say that when in this SVM setting, features maps around 32 to 40 can produce good classification results. The best overall accuracy is 82.5%. Test image in shine class are always be misclassified and sunrise image are always can be classied correctly.

Also, when we focus on wrong prediction list, there are files always be mis-classified: ['shine252.jpg', 'rain214.jpg','cloudy300.jpg', 'rain215.jpg', 'shine253.jpg','shine247.jpg']

3.3 Wrong Prediction Analysis

We use model trained with $C = 32$ as the best model and see what classes are following images classified to.

	
1.'GT:', 'shine', 'Pred:', 'sunrise'	2.'GT:', 'cloudy', 'Pred:', 'shine'
	
3.'GT:', 'rain', 'Pred:', 'shine'	4.'GT:', 'shine', 'Pred:', 'cloudy'
	
5.'GT:', 'rain', 'Pred:', 'shine'	6.'GT:', 'shine', 'Pred:', 'cloudy'

By listing those always misclassified images, we try to explain that result.

- For image 1, in training set most of the sunshine images are a blue sky with some cloud. This one is a flower in the middle of the image. As a human, I also cannot determine if it is sunrise or shine.

- For image 2, in training set most of the cloudy images are dark and gray. It is reasonable that our model will classify it as shine.

- For image 3 and 5, resize image may ruin its variance.
- For image 4, In training set, most of the cloudy images are dark and gray in surrounding but bright in some center.
- For image 6, this image is ambiguous, if we crop the left size, it should be in shine class; if we crop the right size, it will be in cloudy class.

4 Extra Credit Task: Comparison between Gram matrix and LBP

In this section, we will compare two different classification method. One use Gram matrix features with SVM classifier and the other use LBP features with NN classifier.

The best model from Gram Matrix

$C = 32$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	8	0	2	0
	rain	0	8	2	0
	shine	2	0	7	1
	sunrise	0	0	0	10
Validate Accuracy	85.85%(279/325)				
Overall Accuracy	82.5%(33/40)				
Wrong Prediction	'cloudy291.jpg', 'shine252.jpg', 'rain214.jpg', 'cloudy300.jpg', 'rain215.jpg', 'shine253.jpg', 'shine247.jpg'				

Result generated by LBP

$R = 1, P = 8$, NN classifier n-neighborhood $n = 5$

		Prediction			
		cloudy	rain	shine	sunrise
Ground Truth	cloudy	9	0	0	1
	rain	1	9	0	0
	shine	4	0	4	2
	sunrise	4	0	0	6
Overall Accuracy	70%(28/40)				
Wrong Prediction	'shine249.jpg', 'shine247.jpg', 'shine252.jpg', 'sunrise356.jpg', 'shine245.jpg', 'sunrise354.jpg', 'cloudy300.jpg', 'rain209.jpg', 'shine251.jpg', 'sunrise350.jpg', 'shine248.jpg', 'sunrise353.jpg',				

According to above accuracy results, we can say that Gram matrix features with SVM classifier have a better performance on test set than LBP features with NN classifier. Gram matrix features with SVM classifier has 82.5 % accuracy on test set, while LBP features with NN classifier only has 70% accuracy on test set.

Compare two confusion matrix, we notice that LBP have a better performance on rain class. It is reasonable since texture of raining is very different compare with cloudy, shine, and sunrise. However, other classes classification's performance is worse than Gram Matrix. It is very easy for LBP feature to classify shine and sunrise classes images into cloudy class.

Focus on wrong prediction, we notice the overlap images between mis-classification set are only 'shine252.jpg' and 'cloudy300.jpg'. My guess is that Gram matrix with random filters is good at extract color features and LBP is good at extract texture spectrum.

5 Code

5. Code

5.1 main.py

```
#!/usr/bin/python
import os
import cv2
from sklearn.metrics import confusion_matrix
from gram_matrix import gram_feature,filter_generate
import numpy as np
import re
class StatModel(object):
    '''parent class - starting point to add abstraction '''
    def load( self , fn):
        self .model.load(fn)
    def save( self , fn):
        self .model.save(fn)

class SVM(StatModel):
    '''wrapper for OpenCV SimpleVectorMachine algorithm'''
    def __init__ ( self ):
                self .svm = cv2.ml.SVM_create()
                self .svm.setType(cv2.ml.SVM_C_SVC)
                self .svm.setKernel(cv2.ml.SVM_LINEAR)
                # self.svm.setDegree(0.0)
                #self.svm.setGamma(0.5)
                # self.svm.setCoef0(0.0)
                self .svm.setC(0.01)
                # self.svm.setNu(0.5)
                # self.svm.setP(0.0)
                # self.svm.setClassWeights(None)
                self .svm.setTermCriteria((cv2.TERM_CRITERIA_COUNT, 20000, 1.e-07))

    def train( self , samples, responses):
        #setting algorithm parameters
        self.svm.train(samples, cv2.ml.ROW_SAMPLE, responses)

    def predict(self, samples):
        print(samples[0].shape)
        return np.float32( [self.svm.predict(s) for s in samples])


def train(M=3,C=6,c=3,K=16,iterations=10):
        '''
                Extract Gram Matrix features from training images.
                Agrs:
                        M: kernel size
                        C: filter channel size
                        c: input channel size
                        iterations : number of tirals
                Returns:
                        filters : numpy array, MxMxcxC size, the best filters
        '''
        training_dir = os.path.join(os.getcwd(),'imagesDatabaseHW8','training')
        hists = np.array([])
        labels = np.array([])
        print("Training started...")

        samples = os.listdir(training_dir)
        temp=re.compile("([a-zA-Z]+)([0-9]+)")
        images=[]
        for s in samples:
                # Read the image and convert to gray scale
                print(s)
                res=temp.match(s).groups()
                label=res[0]
                fpath = os.path.join(training_dir,s)
                print("Read training database ...", fpath)
                image = cv2.imread(fpath)
                a,b,l=image.shape
                t=np.min(image.shape[:2])
                astart=(a-t)//2
                bstart=(b-t)//2
                print(image.shape)
                images.append(cv2.resize(image[astart:astart+t,bstart:bstart+t,:],(K+2,K+2),
                    interpolation = cv2.INTER_AREA))


                if labels != np.array([]):
                        labels = np.hstack((labels, label))
                else:
                        labels = np.array([label])
        split_ind =760
        lookupTable, indexed_dataSet = np.unique(labels, return_inverse=True)
        for i in range(len(lookupTable)):
                print('Training_%s:%d'%(lookupTable[i],np.sum(indexed_dataSet[:split_ind]==i
                    )))
        for i in range(len(lookupTable)):
                print('Validate_%s:%d'%(lookupTable[i],np.sum(indexed_dataSet[split_ind:]==i)
                    ))
        print(labels [:10], indexed_dataSet[:10])
        print(np.array(images)[0].shape)
        indexed_dataSet=indexed_dataSet.reshape(-1,1).astype(np.int64)
        accuracy_best=0

        filters_best =None
        clf_best =None
        for iter in range(iterations):
                print(' iterations : ', iter)
                filters =filter_generate (M=M,C=C,c=3)
                #print( filters )
                gm_feature_train=gram_feature(np.array(images[:split_ind]) , filters ,samples[:
                    split_ind ], C=C,K=K).astype(np.float32)
                gm_feature_valid=gram_feature(np.array(images[split_ind:]) , filters ,samples[
                    split_ind :], C=C,K=K).astype(np.float32)
                print(gm_feature_train.dtype)
                print(gm_feature_train.shape)
                print(indexed_dataSet[: split_ind ]. shape)
                clf=SVM()
                clf . train (gm_feature_train,indexed_dataSet[: split_ind ])
                y_train = clf.svm.predict(gm_feature_train)
                accuracy=np.sum(y_train[1].astype(np.int64)==indexed_dataSet[:split_ind])/
                    float(len(y_train[1]))
                print('Training_accuracy:',accuracy,np.sum(y_train[1].astype(np.int64)==
                    indexed_dataSet[:split_ind]), '/', float(len(y_train [1]) ))

                y_val = clf.svm.predict(gm_feature_valid)
                accuracy=np.sum(y_val[1].astype(np.int64)==indexed_dataSet[split_ind:])/float(
                    len(y_val[1]))
                #print(y_val [:10], indexed_dataSet[ split_ind : split_ind +10])
                print('Valid_accuracy:',accuracy,np.sum(y_val[1].astype(np.int64)==
                    indexed_dataSet[split_ind:]), '/', float(len(y_val [1]) ))
                if accuracy>accuracy_best:
                        accuracy_best=accuracy
                        filters_best = filters
                        clf_best =clf
        print('Best_accuracy:', accuracy_best)
        return filters_best , clf_best
def test( filters , clf ,C=24,K=16):
        '''
```

```
        Extract Gram Matrix features from test images.
        Print confusion matrix and overall accuracy.
        Agrs:
                filters : numpy array, MxMxcxC size, the best filters obtained in
                    training set.
                clf: Best SVM model obtained in training set.
                C: filter channel size
                K: Feature map size K * K
        Returns:

    '''
    test_dir = os.path.join(os.getcwd(),'imagesDatabaseHW8','testing')
    hists = np.array([])
    labels = np.array([])
    print("Testing started...")

    samples = os.listdir(test_dir)
    temp=re.compile("([a-zA-Z]+)([0-9]+)")
    images=[]
    for s in samples:
        # Read the image and convert to gray scale
        # print(s)
        res=temp.match(s).groups()
        label=res[0]
        fpath = os.path.join(test_dir,s)
        # print("Read testing database ...", fpath)
        image = cv2.imread(fpath)
        a,b,l=image.shape
        t=np.min(image.shape[:2])
        astart=(a-t)//2
        bstart=(b-t)//2
        images.append(cv2.resize(image[astart:astart+t,bstart:bstart+t,:],(K+2,K+2),
            interpolation = cv2.INTER_AREA))


        if labels != np.array([]):
            labels = np.hstack((labels, label))
        else:
            labels = np.array([label])
    lookupTable, indexed_dataSet = np.unique(labels, return_inverse=True)
    indexed_dataSet=indexed_dataSet.reshape(-1,1).astype(np.int64)
    print(np.array(images)[0].shape)
    gm_feature_test=gram_feature(np.array(images),filters,samples,C=C,K=K).astype(np.
        float32)
    print(gm_feature_test.dtype)
    print(gm_feature_test.shape)
    print(indexed_dataSet.shape)
    y_test = clf.svm.predict(gm_feature_test)
```

```
        accuracy=np.sum(y_test[1].astype(np.int64)==indexed_dataSet)/float(len(y_test[1]))
        print('Test accuracy:',accuracy,np.sum(y_test[1].astype(np.int64)==indexed_dataSet),'
            /',float(len(y_test[1])))
        print(lookupTable)
        print(confusion_matrix(indexed_dataSet,y_test[1].astype(np.int64)))
        misclass=np.reshape(y_test[1].astype(np.int64)!=indexed_dataSet,-1)
        print([(s,lookupTable[pred]) for s,ms,pred in zip(samples,misclass,y_test[1].astype(np.
            int64)) if ms==True ])

def main():
        '''
        Find the best model for both filter and svm
        '''
        M = 3
        Cs = [16,24,32,40,64]
        K=32
        n = 100
        for C in Cs:
                filters, clf=train(M=M,C=C,c=3,K=K,iterations=n)
                clf.svm.save('classify_M%02d_C%02d_K%02d_n%03d_normal_0.01.xml'%(M,C,K
                    ,n))
                np.save('filters.npy_M%02d_C%02d_K%02d_n%03d_normal_0.01'%(M,C,K,n),
                    filters)
                test(filters,clf,C=C,K=K)
        # samples = np.array(np.random.random((4,2)), dtype = np.float32)
        # y_train = np.array ([1.,0.,0.,1.], dtype = np.int64)
        # y_train=y_train.reshape(-1,1)
        # clf = SVM()
        # clf.train(samples, y_train)
        # y_val = clf.svm.predict(samples[:3])
        # print(y_train, y_val)
def summary():
        '''
        Summarize all models.
        '''
        M = 3
        Cs = [16,24,32,40,64]
        K=32
        n = 100
        clf=SVM()
        for C in Cs:
                print('Normal distribution, C=%02d'%(C))
                clf.svm=cv2.ml.SVM_load('classify_M%02d_C%02d_K%02d_n%03d_normal_0.01.
                    xml'%(M,C,K,n))
                filters=np.load('filters.npy_M%02d_C%02d_K%02d_n%03d_normal_0.01.npy'
                    %(M,C,K,n))
                test(filters,clf,C=C,K=K)
```

```
if __name__ == '__main__':
        main()
        summary()
```

5.2 Gram Matrix

```
import numpy as np
import cv2
import os
def filter2D(input_arr, filter):
    """
    2D filtering (i.e. convolution but without mirroring the filter). Mostly a convenience
        wrapper
    around OpenCV.

    Args:
    input_arr : numpy array, HxW size
    filter : numpy array, H1xW1 size

    Returns:
    result : numpy array, HxW size

    """
    return cv2.filter2D(input_arr,
                        -1,
                        filter)

def batch_filter3D(input_arr, filters ,K=16):
    """
    3D filtering (i.e. convolution but without mirroring the filter).

    Agrs:
    input_arr : numpy array, NxHxWxC size where N is the number of images to be filtered
    filter : numpy array, H1xW1xC size

    Returns:
    result : numpy array, NxHxW size

    """
    #assert input_arr.shape[3] == filters.shape[2]
    num_input = input_arr.shape[0]
    output = np.zeros((num_input,K+2,K+2) + (filters.shape[-1],))
    print(output.shape)
    for n in xrange(num_input):
        input1 = input_arr[n]
        for f in xrange(filters.shape[-1]):
            for c in xrange(filters.shape[-2]):
                output[n,:,:, f] += filter2D(input1[..., c].copy(), filters[..., c,f].copy())
    return output

def filter_generate (M=3,C=6,c=3):
    '''
    filter generator

    Agrs:
    M: kernel size
    C: filter channel size
    c: input channel size

    Returns:
    filter : numpy array, MxMxcxC size

    '''
    filters =np.random.randn(M,M,c,C)
    #filters =np.random.uniform(0,1,(M,M,c,C))
    # print('Original :', filters [:,:,0], np.mean(filters , axis=(0,1)))
    filters = filters −np.mean(filters,axis=(0,1))
    # print('Mean Normalization:',filters [:,:,0], np.max(filters , axis=(0,1)))
    filters = filters /np.max(np.absolute(filters), axis=(0,1))
    # print('Output:', filters [:,:,0])
    # print(np.sum(filters , axis=(0,1)))
    return filters



def gram_matrix(image):
    '''
    Generate Gram Matrix

    Args:
    image: numpy array MxMxC size

    Returns:
    Upper traingular Gram Matrix: CxC size
    '''

    vec=image.reshape(−1,image.shape[−1])
    return np.triu(np.dot(vec.T,vec))

def gram_feature(images, filters ,samples,C,K=16):
    conv= batch_filter3D(images, filters ,K)
    gm_features=np.zeros((len(images),C∗(C+1)//2))
    for i in range(len(images)):
        gm=gram_matrix(conv[i][1:−1,1:−1,:])# Valid
        gm_features[i]=gm[np.nonzero(gm)]
    return gm_features
```

```
if __name__ == '__main__':
    filters = filter_generate (M=3,C=6,c=3)
    training_dir  = os.path.join(os.getcwd(),'imagesDatabaseHW8','training')
    samples = os. listdir ( training_dir )
    fpath = os.path.join( training_dir ,samples[0])
    images = cv2.imread(fpath)
    conv= batch_filter3D(np.array([images]), filters )
    gm=gram_matrix(conv[0])
    gm_features=gm[np.nonzero(gm)]
```