Labsheet 3
Lab-3 Sampling and SVD

Ragja Palakkadavath
(SC18M003)

Sub: Image Processing Lab
Department: Maths(ML)
Date of Lab sheet: August 23, 2018
Date of Submission: September 5, 2018

Question 1:

Sampling an image
Use the peppers image and perform the following:

1. Downsample the image in spatial domain by removing alternate columns and rows , by a factor of 2 and 4
2. Reconstruct the image from the downsampled one by interpolation. Use bilinear interpolation in both cases and comment on the results obtained.
3. Comment on the various interpolation methods ,aliasing and about how you can avoid aliasing.

Aim

The **Aim** of the above program is to learn about the down sampling of image in spatial domain and reconstructing the original image via interpolation

Discussion

Sampling Theorem states that bandlimited signal can be reconstructed exactly if it is sampled at a rate atleast twice the maximum frequency component in it. Digital sampling of any signal can result in apparent signals at frequencies well below anything present in the original. Aliasing occurs when a signal is sampled at a less than twice the highest frequency present in the signal. ie, when it does not follow the sampling theory
Down sampling is performed in images to digitize the image especially for storage purposes.
Interpolation works by using known data to estimate values at unknown points.

Algorithm

- The given image was downsampled by a factor of 2 and then by a factor of 4 via block reduce function
- The downsampled images from part 1 were then up sampled by the method of bilinear interpolation technique
- The equations provided in the labsheet were used to reconstruct the image back using bilinear interpolation

Program Code

Down Sampling

```
from PIL import Image
from skimage.measure import block_reduce
import numpy as np
import imageio
def interpolate(image):
    new_image=block_reduce(image, block_size=(2, 2), func=np.sum)
    img=imageio.imwrite('1_a.png',new_image)

def interpolate2(image):
    new_image=block_reduce(image, block_size=(4, 4), func=np.sum)
    img=imageio.imwrite('1_a_4.png',new_image)

def get_image():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/3_SC18M003_Ragj
    a_IP2018/peppers.png').convert('L')
    img.load()
    data = np.asarray( img, dtype="float64" )
    imageio.imwrite('peppers_salt.png',data)
    interpolate2(data)

get_image()
```

Up Sampling via Bilinear Interpolation

```
import numpy as np
from PIL import Image
import imageio

def interpolation(image):
    width,height =image.shape
    out_image = np.zeros(2*height*2*width).reshape(2*height,2*width)
    for m in range(height-1):
        for n in range(width-1):
            out_image[2*m][2*n] = image[m][n]
            out_image[2*m][2*n+1]= (image[m][n] + image[m][n + 1])/2
            out_image[2*m + 1][2*n] = (image[m][n] + image[m + 1][n])/2
            out_image[2*m+1][2*n+1] = (image[m][n]+image[m+1][n]+image[m][n+1]+image[m+1
            [n + 1])/4

    im =Image.fromarray(out_image.astype('int8'),'L')
    im.show()
    return im

def sampleby4(data):
    data = np.asarray( data, dtype="int64" )
    im=interpolation(data)
    data = np.asarray( im, dtype="int64" )
    imageio.imwrite('upsampled_bilinear2.png',data)

def get_image():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/3_SC18M003_Ragj
    a_IP2018_latest/3_SC18M003_Ragja_IP2018/1_a.png').convert('L')
    img.load()
    data = np.asarray( img, dtype="int64" )
    #interpolated=interpolation(data)
```

```
    sampleby4(data)

get_image()
```

Result

Sampling and Interpolation

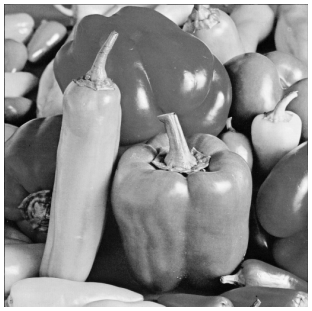

Figure 1: Original Pepper Image (in grayscale)

Figure 2: Down sample pepper image by 2

Figure 3: Down sample pepper image by 4

Figure 4: Up sample down sampled by 2 image

Figure 5: Up sample down sampled by 4 image

Inference

The given image was downsampled across rows and columns by a factor of 2 and then by a factor of 4 The downsampled image was reconstructed/ up sampled by bilinear interpolation techniques Various interpolation techniques and aliasing was discussed

Question 2:

Eigen values and Singular values
Find the eigen values and singular values of the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Explore further on the relation between eigen values and singular values.

Aim

The **Aim** of the above program is to obtain eigen values and singular values of the given matrix

Discussion

The singular values of a MN matrix X are the square roots of the eigenvalues of the NN matrix XX (where stands for the transpose-conjugate matrix if it has complex coefficients, or the transpose if it has real coefficients)

Written in matrix form, the defining equations for singular values and vectors are
AV = U$\sum$ A^HU = V $\sum$ ^{H}

Here U$\sum$ is almost always matrix the same size as A that is zero except possibly on its main diagonal.

It turns out that singular vectors can always be chosen to be perpendicular to each other, so the matrices U and V , whose columns are the normalized singular vectors, satisfy $U^H U = I$ and $V^H V = I$. In other words, U and V are orthogonal if they are real, or unitary if they are complex. Consequently, A = U$\sum V^H$, with diagonal $\sum$ and orthogonal or unitary U and V . This is known as the singular value decomposition, or SVD, of the matrix A.

Algorithm

- Read the image and find the eigen values using the characteristic eigen value equation
- Perform SVD on the image to obtain its singular values
- Singular values are obtained by solving the characteristic equations of $A^H A$ and taking its roots

Program Code

Code for 2(a):

```
from sympy import Symbol
from sympy.polys.polytools import Poly
import numpy as np
from sympy import Matrix

def eigen_calc(M):
    x = Symbol('x')
    M = Matrix(M)
    I=Matrix([[x,0,0],[0,x,0],[0,0,x]])
    MI=M-I
    char_equation=MI.det()
    a = Poly(char_equation, x)
    eigen_values=np.roots(a.coeffs())
    return eigen_values

print(eigen_calc([[1,0,1],[0,1,1],[0,0,0]]))
```

Result

Eigen Values and Singular Values

```
Eigen Values of the Matrix: Obtained via Characteristic Poly
[1. 1.]
Singular Values of the Matrix: Obtained from SVD
[1.73205081 1.         0.        ]
```

Inference

Eigen values of a matrix can be obtained from its characteristic equation for a considerably small matrix
To find singular values of a m x n matrix, Singular Value Decomposition needs to be performed.

Question 3:

Singular Value Decomposition

(a) Perform SVD on the given image and identify its eigen images.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(b) Show the different stages of SVD of the following image:

$$\begin{bmatrix} 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 200 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 255 & 255 & 50 & 255 & 255 & 255 \\ 50 & 50 & 50 & 50 & 255 & 255 & 255 & 255 \end{bmatrix}$$

Aim

The **Aim** of the above program is to perform Singular Value Decomposition of the given Matrix.

Discussion

The Singular Value Decomposition (SVD) of an image f is its expansion in terms of vector outer products, where the vectors used are the eigenvectors of ff^T and f^Tf, and the coefficients of the expansion are the eigenvalues of these matrices. It is given by:

$$f = \sum_{i=1}^{r} \lambda_i^{\frac{1}{2}} \mathbf{u_i} \mathbf{v_i}^T$$

Figure 6: SVD Equation

Algorithm

- Read the input image/matrix and convert to matrix
- Form ff^T and f^Tf matrices and find its eigenvalues and eigenvectors via eigh function present in linalg module
- Use the singular values and eigen vectors obtained to form the Singular Value Decomposition of the image as given in the discussion.

Program Code

```
import numpy as np
import math
from PIL import Image
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt2
from numpy.lib import scimath as sm
def normalize(v):
    norm = np.linalg.norm(v)
    if norm == 0:
       return v
```

```
    return v / norm

def svd_calc(M, size):
    g=np.array(M)
    g_transpose=np.transpose(g)
    gg_transpose=np.dot(g,g_transpose)
    g_transposeg=np.dot(g_transpose,g)
    eigen_val_1,eigen_vect_u=np.linalg.eigh(gg_transpose)
    eigen_val_2,eigen_vect_v=np.linalg.eigh(g_transposeg)

    eigen_vect_u=list(eigen_vect_u.T)
    eigen_vect_v=list(eigen_vect_v.T)

    v=np.zeros((size,size))
    u=np.zeros((size,size))

    #eigen_val = [float(np_float) for np_float in eigen_val_1]
    basis=np.array([])
    to_be_zipped=(zip(eigen_val_1, eigen_vect_u))
    to_be_zipped2=(zip(eigen_val_1, eigen_vect_v))

    eigen_val,eigen_vect_u=(list(t) for t in
    (zip(*sorted(to_be_zipped,key = lambda t: t[0], reverse=True))))
    eigen_val,eigen_vect_v=(list(t) for t in
    (zip(*sorted(to_be_zipped2,key = lambda t: t[0], reverse=True))))
    #size=size-3
    for i in range(0,size):
        vector=np.array(eigen_vect_u[i])
        u[i]=np.copy(vector)
    for i in range(0,size):
        vector=np.array(eigen_vect_u[i])
        #v_vector=np.dot(g_transpose,vector)
        v[i]=np.copy(vector)
    sing_values=sm.sqrt(eigen_val)
    print(sing_values)
    original_image=np.zeros((size,size))
    u=-1*u
    v=-1*v
    print(u)
    print(v)
    for i in range(0,size):
        basis=np.zeros((size,size))
        outer_product=np.outer(u[:,i],v[i,:])
        #print(outer_product)
        basis=basis+(sing_values[i]*outer_product)
        for j in range(0,basis.shape[0]):
            for k in range(0,basis.shape[1]):
                basis[j][k]=int(round(basis[j][k]))
        basis_img=Image.fromarray(basis.astype('uint8'))
        original_image=original_image+basis
        plt.figure()
        plt.imshow(basis.real, cmap = 'gray')
    print(original_image)
svd_calc([[0,1,0],[1,0,1],[0,1,0]],3)
'''svd_calc([[255,255,255,255,255,255,255,255],
     [255,255,255,100,100,100,255,255],
     [255,255,100,150,150,150,100,255],
     [255,255,100,150,200,150,100,255],
```

```
[255,255,100,150,150,150,100,255],
[255,255,255,100,100,100,255,255],
[255,255,255,255,50,255,255,255],
[50,50,50,50,255,255,255,255]],8)'''
```

Result

SVD in steps

```
Part (a)
Eigen Values:
[2. 2. 0.  ]
Singular Values:
[1.41421356 1.41421356 0.  ]
```

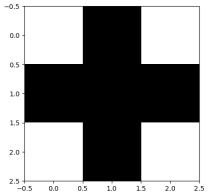

Figure 7: Eigen Image 1

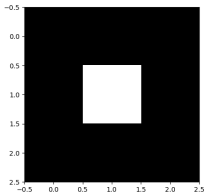

Figure 8: Eigen Image 2

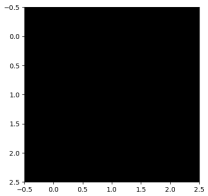

Figure 9: Eigen Image 3

```
Part (b)
Eigen Values:
[2593416.5554988617, 111621.52323579353, 71738.32225949703,
34790.85800319984, 11882.741002648192, 9.360841565415186e-12,
3.0706774996504958e-12, -6.655063958172905e-11]
Singular Values:
[1.61040882e+03+0.00000000e+00j 3.34098074e+02+0.00000000e+00j
```

```
2.67840106e+02+0.00000000e+00j 1.86523076e+02+0.00000000e+00j
1.09007986e+02+0.00000000e+00j 3.05954924e-06+0.00000000e+00j
1.75233487e-06+0.00000000e+00j 0.00000000e+00+8.15785754e-06j]
```

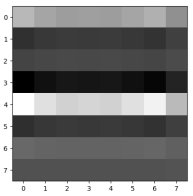

Figure 10: Eigen Image 1

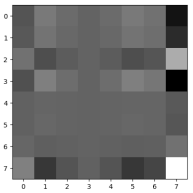

Figure 11: Eigen Image 2

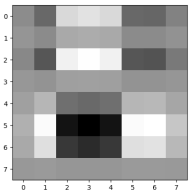

Figure 12: Eigen Image 3

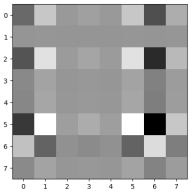

Figure 13: Eigen Image 4

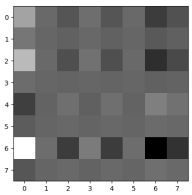

Figure 14: Eigen Image 5

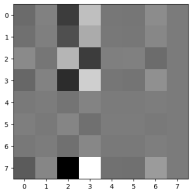

Figure 15: Eigen Image 6

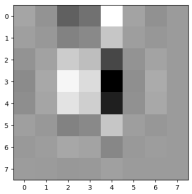

Figure 16: Eigen Image 7

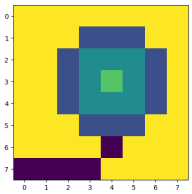

Figure 17: Eigen Image 8

Inference

The matrices given were decomposed into its eigen images using Single Value Decomposition method. Eigen images were added together to restore the original images.

Question 4:

Approximating using SVD

1. Use the cameraman image and find its singular values

2. Choose some value k<n, where n is the rank of $\sum$, the singular matrix
3. Keep the first k singular values and the rest zeroes to get $\sum$
4. Reconstruct the image using these singular values
5. Calculate the error
6. Plot the error as a function of k
7. Add some random noise to your input image and do similar approximation.Infer your results.

Aim

The **Aim** of the above program is to approximate an image using its Singular Value Decomposition

Discussion

An image has been decomposed into its components via Singular Value Decomposition. An approximated version of the image can be obtained :

$$f_k = \sum_{i=1}^{k} \lambda_i^{\frac{1}{2}} \mathbf{u_i} \mathbf{v_i}^T$$

Figure 18: Image Reconstruction Equation

Algorithm

- Read the input image(cameraman.tif)
- Compute the singular value decomposition of the image and obtain the singular values and u and v matrices
- Reconstruct the image after removing few singular values from the lower end of the singular value array(k is taken as 50 which means every singular value from 50 is now made 0)
- The reconstructed image after the decomposition is not very different from the original image

Program Code

```
from PIL import Image
import numpy as np
from numpy import linalg
import random
from matplotlib import pyplot as plt
import math
def get_image(image_path,image_name):
    path=image_path+image_name
    print(path)
    img = Image.open(path)
    img.load()
    data = np.asarray( img, dtype="float64" )
    approx_svd(data,data.shape[0])

def approx_svd(image,size):
    u,sing,vt = linalg.svd(image)
    rank=len(sing)
    sing2=list(sing)
    print(rank)
```

```
    k=50
    for i in range(0,len(sing2)):
        if i>=k:
            sing2[i]=0
    print(sing2)
    basis=np.zeros((size,size))
    for i in range(0,k):
       ui = u[:,i]
       vi = vt[i,:]
       appr = np.outer(ui,vi)
       appr=sing2[i]*appr
       basis=basis+appr
    #plt.figure()
    #plt.imshow(basis.real, cmap = 'gray')
    #Error via difference of actual and basis squared
    error=image-basis
    error_sum=np.sum(np.square(error))
    print(error_sum)
    #Error via sum of omitted singular values
    err = 0
    #Calculation of Error
    for i in range(50,256):
        err = err + (sing[i]**2)
    print(err)
    #Plotting error
    err = list()
    for i in range(0,256):
        k = 0
        if i==255:
            break
        for j in range(i+1,256):
            k = k + sing[j]
        err.append(k)
    err.append(0)
    kv = np.arange(0,256)
    plt.plot(kv,err)
    plt.title('k v/s error plot')
    plt.xlabel('k')
    plt.ylabel('error')

def noise_svd(image,size):
    noise = np.random.randint(20,255, size=(size, size))
    imgnoise = image + noise
    plt.imshow(imgnoise.real, cmap='gray')

    noised_image_array = np.array(imgnoise)
    u, sing, vt = np.linalg.svd(noised_image_array)
    sing2=list(sing)
    basis=np.zeros((size,size))
    k=50
    for i in range(0,len(sing2)):
        if i>=k:
            sing2[i]=0
    print(sing2)
    for i in range(0,k):
       ui = u[:,i]
       vi = vt[i,:]
       appr = np.outer(ui,vi)
```

```
        appr=sing[i]*appr
        basis=basis+appr
    plt.figure()
    plt.imshow(basis.real, cmap = 'gray')
get_image('C:/Users/IIST/Downloads/Ragja/3_SC18M003_Ragja_IP2018_latest/3_SC18M003_Ragj
```

Result

Approximating an image using SVD

Figure 19: Original Cameraman Image

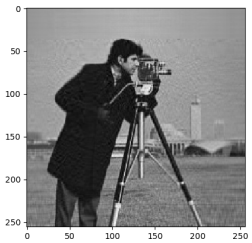

Figure 20: Restoration of Cameraman after SVD (k=50 eigen values were removed)

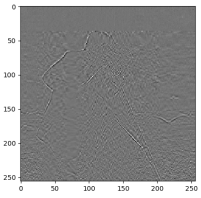

Figure 21: The error post the removal of insignificant eigenvalues

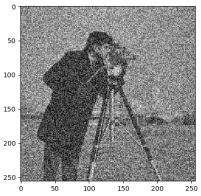

Figure 22: Cameraman Image with noise added to it

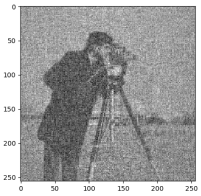

Figure 23: Cameraman Image with noise restored after SVD(with k=50) where insignificant eigenvalues were removed

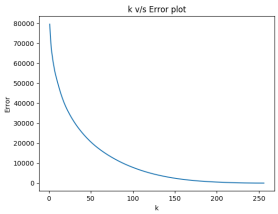

Figure 24: Error v/s K Plot

Inference

We apply the Single Value Decomposition to *cameraman.jpg* and decompose it into its eigenimages. The original image is restored via adding up the eigen images using the singular values of the image and the two Orthogonal matrices U and V. During Image reconstruction, few unimportant singular values were omitted from the eigenimages - However, the original was reconstructed successfully without huge error. Error was plotted as a function of k