hw1

March 16, 2018

1 Computer Vision

1.1 Assigment 1

1.2 March 11, 2018

Welcome to Oversea Research Program - Computer Vision. This program will give you a comprehensive introduction to computer vison providing board coverage including low level vision, inferring 3D properties from image, and object recognition. We will be using a varity of tools in this class that will require some initial configuration. To ensure everything smoothly moving forward, we will setup the majority of the tools to be used in this course in this assignment. You will also practice some basic image manipulation techniques. At the end, you will need to export this Ipython notebook as pdf.

1.2.1 Python

Python

We will use the Python programming language for all assignments in this course, with a few popular libraries (numpy, matplotlib). And assignment starters will be given in format of the browser-based Jupyter/Ipython notebook that you are currently viewing. If you have previous knowledge in Matlab, check out the numpy for Matlab users page. The section below will serve as a quick introduction on Numpy and some other libraries.

Setup Python environment

We can install Anaconda from the links given below. You can setup your environment using Anaconda for Python 2.7 or 3.6.

The Anaconda versions for Python can be downloaded from the following:

https://www.anaconda.com/download/#linux

https://www.anaconda.com/download/#macos

https://www.anaconda.com/download/#windows

After downloading and installing one of these, one needs to set the /path/to/anaconda2 in $PATH variable.

Then we can run >> jupyter notebook from terminal or use the Anaconda UI. Otherwise a more "geeky" procedure for Linux users is given here:

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-jupyter-notebook-to-run-ipython-on-ubuntu-16-04.

For submitting your assignments, you can submit your python notebook file with result shown or PDF file. PDF file is needed to setup using LaTex.

Please use nbconvert tool for this. This can be installed from instructions given on: nbconvert: "conda install nbconvert" (or http://nbconvert.readthedocs.io/en/latest/install.html) The above link also gives instructions for installing Pandoc and Latex for different OS. Please follow those instructions as installing these might be required for nbconvert.

1.3 Get started with Numpy

Numpy is the fundamental package for scientific computing with Python. It provides a powerful N-dimensional array object and functions for working with these arrays.

1.3.1 Arrays

```
In [1]: import numpy as np

        v = np.array([1, 0, 0])         # a 1d array
        print("1d array")
        print(v)
        print(v.shape)                  # print the size of v
        v = np.array([[1], [2], [3]])   # a 2d array
        print("\n2d array")
        print(v)
        print(v.shape)                  # print the size of v, notice the difference
        v = v.T                         # transpose of a 2d array

        m = np.zeros([2, 3])            # a 2x3 array of zeros
        v = np.ones([1, 3])             # a 1x3 array of ones
        m = np.eye(3)                   # identity matrix
        v = np.random.rand(3, 1)        # random matrix with values in [0, 1]
        m = np.ones(v.shape) * 3        # create a matrix from shape
```

```
1d array
[1 0 0]
(3,)

2d array
[[1]
 [2]
 [3]]
(3, 1)
```

1.3.2 Array indexing

```
In [2]: import numpy as np

        m = np.array([[1, 2, 3], [4, 5, 6]])  # create a 2d array with shape (2, 3)
```

```
print("Access a single element")
print(m[0, 2])                          # access an element
m[0, 2] = 252                           # a slice of an array is a view into the same da
print("\nModified a single element")
print(m)                                # this will modify the original array

print("\nAccess a subarray")
print(m[1, :])                          # access a row (to 1d array)
print(m[1:, :])                         # access a row (to 2d array)
print("\nTranspose a subarray")
print(m[1, :].T)                        # notice the difference of the dimension of resu
print(m[1:, :].T)                       # this will be helpful if you want to transpose

# Boolean array indexing
# Given a array m, create a new array with values equal to m
# if they are greater than 0, and equal to 0 if they less than or equal 0

m = np.array([[3, 5, -2], [5, -1, 0]])
n = np.zeros(m.shape)
n[m > 0] = m[m > 0]
print("\nBoolean array indexing")
print(n)
```

```
Access a single element
3

Modified a single element
[[  1   2 252]
 [  4   5   6]]

Access a subarray
[4 5 6]
[[4 5 6]]

Transpose a subarray
[4 5 6]
[[4]
 [5]
 [6]]

Boolean array indexing
[[ 3.  5.  0.]
 [ 5.  0.  0.]]
```

1.3.3 Operations on array

Elementwise Operations

```
In [3]: import numpy as np

        a = np.array([[1, 2, 3], [2, 3, 4]], dtype=np.float64)
        print(a * 2)                                                # scalar multiplication
        print(a / 4)                                                # scalar division
        print(np.round(a / 4))
        print(np.power(a, 2))
        print(np.log(a))


        b = np.array([[5, 6, 7], [5, 7, 8]], dtype=np.float64)
        print(a + b)                                                # elementwise sum
        print(a - b)                                                # elementwise difference
        print(a * b)                                                # elementwise product
        print(a / b)                                                # elementwise division
```

```
[[ 2.  4.  6.]
 [ 4.  6.  8.]]
[[ 0.25  0.5   0.75]
 [ 0.5   0.75  1.  ]]
[[ 0.  0.  1.]
 [ 0.  1.  1.]]
[[  1.   4.   9.]
 [  4.   9.  16.]]
[[ 0.          0.69314718  1.09861229]
 [ 0.69314718  1.09861229  1.38629436]]
[[  6.   8.  10.]
 [  7.  10.  12.]]
[[-4. -4. -4.]
 [-3. -4. -4.]]
[[  5.  12.  21.]
 [ 10.  21.  32.]]
[[ 0.2         0.33333333  0.42857143]
 [ 0.4         0.42857143  0.5       ]]
```

Vector Operations

```
In [4]: import numpy as np

        a = np.array([[1, 2], [3, 4]])
        print("sum of array")
        print(np.sum(a))                # sum of all array elements
        print(np.sum(a, axis=0))        # sum of each column
        print(np.sum(a, axis=1))        # sum of each row
        print("\nmean of array")
        print(np.mean(a))               # mean of all array elements
        print(np.mean(a, axis=0))       # mean of each column
        print(np.mean(a, axis=1))       # mean of each row
```

```
sum of array
10
[4 6]
[3 7]

mean of array
2.5
[ 2.  3.]
[ 1.5  3.5]
```

Matrix Operations

```
In [5]: import numpy as np

        a = np.array([[1, 2], [3, 4]])
        b = np.array([[5, 6], [7, 8]])
        print("matrix-matrix product")
        print(a.dot(b))                 # matrix product
        print(a.T.dot(b.T))

        x = np.array([1, 2])
        print("\nmatrix-vector product")
        print(a.dot(x))                 # matrix / vector product
```

```
matrix-matrix product
[[19 22]
 [43 50]]
[[23 31]
 [34 46]]

matrix-vector product
[ 5 11]
```

1.3.4 SciPy image operations

SciPy builds on the Numpy array object and provides a large number of functions useful for scientific and engineering applications. We will show some examples of image operation below which are useful for this class.

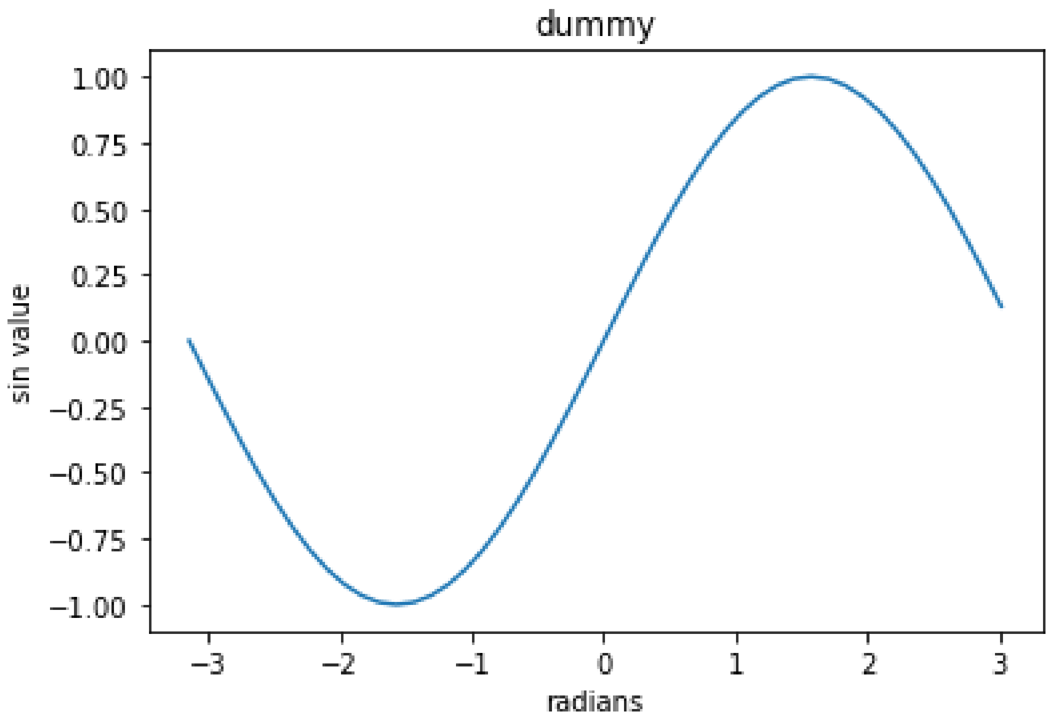
```
In [6]: from scipy.misc import imread, imsave
        import numpy as np

        img = imread('Lenna.png')  # read an JPEG image into a numpy array
        print(img.shape)                      # print image size and color depth

        img_gb = img * np.array([0., 1., 1.])      # leave out the red channel
        imsave('Lenna_gb.png', img_gb)
```

```
(512, 512, 3)
```

1.3.5 Matplotlib

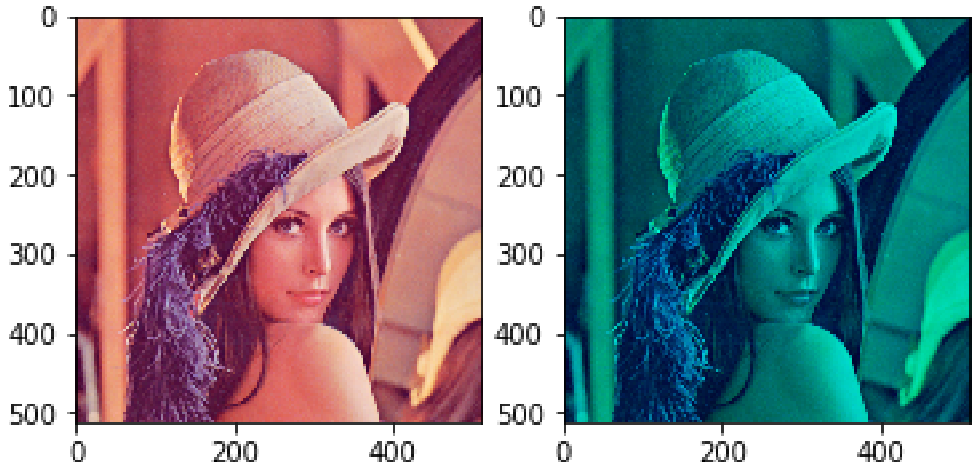
Matplotlib is a plotting library. We will use it to show result in this assignment.

```
In [7]: # this line prepares IPython for working with matplotlib
        %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
        import math

        x = np.arange(-24, 24) / 24. * math.pi
        plt.plot(x, np.sin(x))
        plt.xlabel('radians')
        plt.ylabel('sin value')
        plt.title('dummy')

        plt.show()
```



```
In [8]: # images and subplot
        import numpy as np
```

```
from scipy.misc import imread
import matplotlib.pyplot as plt

img1 = imread('Lenna.png')
img2 = imread('Lenna_gb.png')

plt.subplot(1, 2, 1)  # first plot
plt.imshow(img1)

plt.subplot(1, 2, 2) # second plot
plt.imshow(img2)
plt.show()
```



This breif overview introduces many basic functions from a few popular libraries, but is far from complete. Check out the documentations for Numpy, Scipy and Matplotlib to find out more.

1.4 Problem 1 Function

```
In [1]: # This is the most basis practices in Python.
        # Please print'Welcome to Oversea Rearch Program for Computer Vision'
        # to complete this problem.

        import numpy as np

        def fcn():
            return 'Welcome to Oversea Rearch Program for Computer Vision'
```

```
In [2]: # test the function
        fcn()
```

```
Out[2]: 'Welcome to Oversea Rearch Program for Computer Vision'
```

1.5 Problem 2 Matrix Manipulation

```
In [12]: import numpy as np
         print('question (1):')
         A=np.array([[2,59,2,5],[41,11,0,4],[18,2,3,9],[6,23,27,10],[5,8,5,1]])
         B=np.array([[0,1,0,1],[0,1,1,1],[0,0,0,1],[1,1,0,1],[0,1,0,0]])
         print('A=',A,sep='\n')
         print('B=',B,sep='\n')
         print('\nquestion (2):')
         C=A*B
         print('C=',C,sep='\n')
         print('\nquestion (3):')
         print('inner_product=',np.sum(C[1,:]*C[4,:]))
         print('\nquestion (4):')
         print('C_max=',np.amax(C))#amax, find the max element
         print('C_min=',np.amin(C))
         row, column=C.shape
         i_max,j_max=divmod(np.argmax(C),column)
         #argmax:find the position of the max element.
         #c,d=divmod(a,b): a/b=c......d
         i_min,j_min=divmod(np.argmin(C),column)
         print('max_position=',(i_max,j_max))
         print('min_position=',(i_min,j_min))
         print('\nquestion (5):')
         D=C-C[0,:]
         print('D=',D,sep='\n')
         print('\nquestion (6):')
         print('D_max=',np.amax(D))
         print('D_min=',np.amin(D))
         row,column=D.shape
         i_max,j_max=divmod(np.argmax(D),column)
         i_min,j_min=divmod(np.argmin(D),column)
         print('max_position=',(i_max,j_max))
         print('min_position=',(i_min,j_min))
```

```
question (1):
A=
[[ 2 59  2  5]
 [41 11  0  4]
 [18  2  3  9]
 [ 6 23 27 10]
 [ 5  8  5  1]]
B=
[[0 1 0 1]
 [0 1 1 1]
 [0 0 0 1]
 [1 1 0 1]
 [0 1 0 0]]
```

```
question (2):
C=
[[ 0 59  0  5]
 [ 0 11  0  4]
 [ 0  0  0  9]
 [ 6 23  0 10]
 [ 0  8  0  0]]

question (3):
inner_product= 88

question (4):
C_max= 59
C_min= 0
max_position= (0, 1)
min_position= (0, 0)

question (5):
D=
[[  0   0   0   0]
 [  0 -48   0  -1]
 [  0 -59   0   4]
 [  6 -36   0   5]
 [  0 -51   0  -5]]

question (6):
D_max= 6
D_min= -59
max_position= (3, 0)
min_position= (2, 1)
```

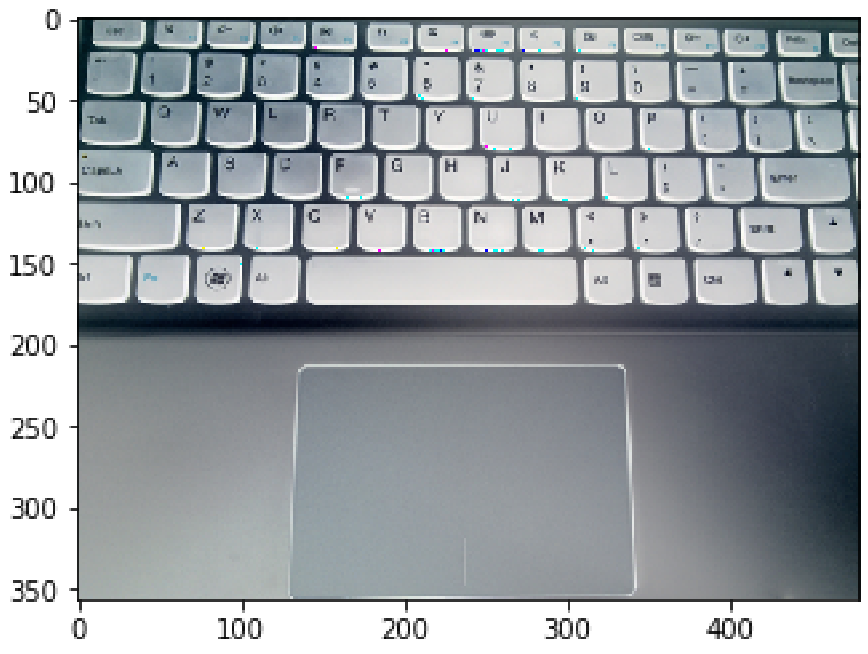
1.6 Problem 3 Keyboard Conundrum

In problem, you will create a function merge(img1, img2, ncols) that horizontally concatenates two perfectly aligned images. (laptop_left.png and laptop_right.png). The third argument ncols specifies the number of columns that must be deleted before the images are merged.

```
In [5]: import numpy as np
        from scipy.misc import imread,imsave
        import matplotlib.pyplot as plt

        def merge(img1,img2,ncols):
            rows=img1.shape[0]
            columns=img1.shape[1]+img2.shape[1]-ncols
            img=np.zeros([rows,columns,3])
            for i in range (0,rows):
                for j in range (0,img1.shape[1]):
```

```
                img[i,j]=img1[i,j]
            for j in range (img1.shape[1],columns):
                img[i,j]=img2[i,j-img1.shape[1]+ncols]
        return img

    left=imread('laptop_left.png')
    right=imread('laptop_right.png')
    print(left.shape)
    print(right.shape)
    merge_pic=merge(left,right,15)#ncols=15 is attained by attempting several different me
    print(merge_pic.shape)
    imsave('laptop.png',merge_pic)
    plt.imshow(merge_pic)
    plt.show()

    #another coding method using slice
```

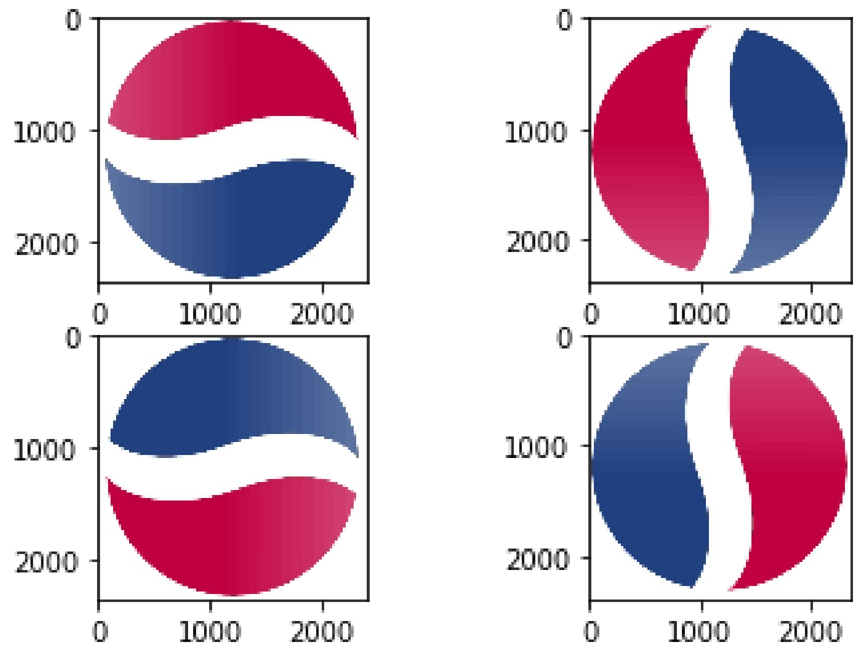
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:16: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  app.launch_new_instance()
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.


(356, 215, 3)
(356, 280, 3)
(356, 480, 3)
```



```
In [ ]: ## Problem 4 Image Manipulation

        In the assignment folder, you will find an image "pepsi.jpg". Import this image and wr

        You should write the rest of the code to print these results in a 2X2 grid using the su
        (You may not use OpenCV function for this part.)
```

```
In [2]: import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt

        # Rotate image (img) by 90 anticlockwise
        def rotate90(img):
            rows=img.shape[0]
            columns=img.shape[1]
            img_res=np.zeros([columns,rows,3], dtype=np.uint8)
            for i in range (0,rows):
                for j in range (0,columns):
                    img_res[columns-j-1][i]=img[i][j]
            return img_res

        # Roate image (img) by an angle (ang) in anticlockwise direction
        # Angle is assumed to be divisible by 90 but may be negative
        def rotate(img, ang):
            assert ang%90==0
            num=ang/90%4
            if num==1:
                return rotate90(img)
            if num==2:
                return rotate90(rotate90(img))
            if num==3:
                return rotate90(rotate90(rotate90(img)))
            if num==4:
                return img


        #Import image here
        pic=imread('pepsi.jpg')
        pic_rot90=rotate90(pic)
        pic_180=rotate(pic,180)
        pic_270=rotate(pic,270)

        #Sample call
        pic_rot90=rotate90(pic)
        pic_180=rotate(pic,180)
        pic_270=rotate(pic,270)
        #Plotting code below
        plt.subplot(2,2,1)
        plt.imshow(pic)


        plt.subplot(2,2,2)
        plt.imshow(pic_rot90)


        plt.subplot(2,2,3)
        plt.imshow(pic_180)


        plt.subplot(2,2,4)
        plt.imshow(pic_270)
        plt.show()
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:31: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

hw2

March 24, 2018

1 Computer Vision

1.1 Assigment 1

1.2 Feburary 4, 2018

This assignment contains 3 programming exercises.

2 Problem 1: Sampling and Quantization

In this problem, we intend to study the effects of sampling and quantization on digital images. Your job is to write a function with the following specifications (you may use loops if necessary):

(i) The function takes one input: the image file name, 'peppers.png'.

(ii) The input image is assumed to be grayscale.

(iii) Sample the image in spatial domain with a sampling rate of 10 (your image should be approximately 10 times smaller along width and height, do not use any numpy functions).

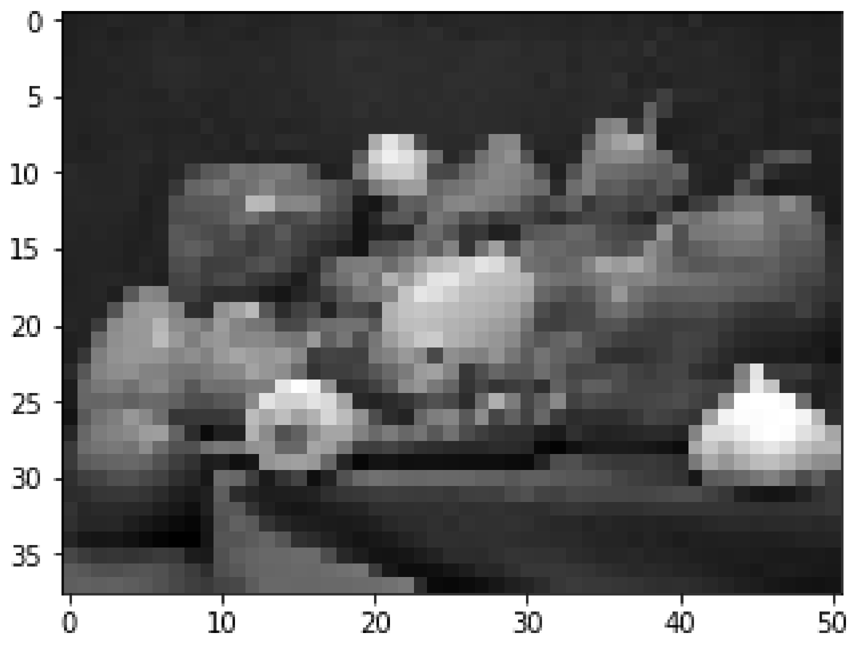
(iv) Do a 5-level uniform quantization of the sampled image so that the bins cover the whole range of grayscale values (0 to 255). You should not use any numpy functions for this.

(v) The function returns one output: the sampled and quantized image.

```
In [2]: import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt
        import math

        def sampling_quantization(img):
            widths=math.floor(img.shape[0]/10)
            height=math.floor(img.shape[1]/10)
            img_res=np.zeros([widths,height])
            sum=0
            for i in range(0, widths):
                for j in range(0, height):
```

```
                img_res[i,j]=img[10*i, 10*j]
                for m in range (10*i,10*i+10):
                    for n in range (10*j,10*j+10):
                        sum=sum+img[m,n]
                    sum=sum/100     #measure the average of per 100 elements
                    img_res[i,j]=sum

        return img_res

    pic=imread('peppers.png')
    pic_res=sampling_quantization(pic)
    plt.imshow(pic_res,cmap='gray')
    plt.show()
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



2.1 Problem 2 Image shift

Shifting an image x of size (n1, n2) in a direction (k, l) consists in creating a new image xshifted of size (n1, n2) such that

In practice, boundary conditions should be considered for pixels (i, j) such that (i + k, j + l) not equal to [0, n1-1] x [0, n2-1].

A typical example is to consider periodical boundary conditions such that

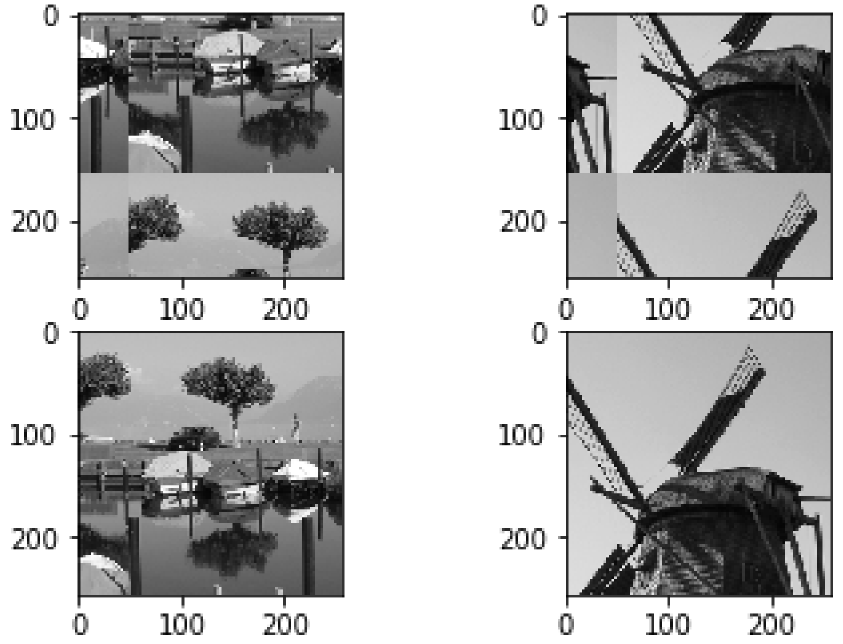
Create in imshift function implementing the shifting of an image x in periodical boundary, such as the following image(b) Shifted in the direction (k,l) by (+100,-50):

Hint: First write it using loops, and next try to get rid of the loops.

```
In [3]: import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt

        def imshift(x, p, q):
            n1=x.shape[1]#columns
            n2=x.shape[0]#rows
            k=p%n2
            l=q%n1
            xshifted=np.zeros([n2,n1])
            xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
            xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
            xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
            xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
            return xshifted

        #Sample call and Plotting code
        #lake.png and "windmill.png"
        pic1=imread('lake.png')
        pic2=imread('windmill.png')
        new_pic1=imshift(pic1,356,306)
        new_pic2=imshift(pic2,356,306)
                            #356=256+100 306=256+50 prove that periodical boundary conditions
        plt.subplot(2,2,1)
        plt.imshow(new_pic1,cmap='gray')
        plt.subplot(2,2,2)
        plt.imshow(new_pic2,cmap='gray')
        back_pic1=imshift(new_pic1,-356,-306)
        back_pic2=imshift(new_pic2,-356,-306)
        plt.subplot(2,2,3)
        plt.imshow(back_pic1,cmap='gray')
        plt.subplot(2,2,4)
        plt.imshow(back_pic2,cmap='gray')
        plt.show()



        imshift(1*pic1+1*pic2,356,306)==1*imshift(pic1,356,306)+1*imshift(pic2,356,306)

        #I thought it should be linear, but the result showed 'false', and I don't know why...
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:19: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:20: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



```
Out[3]: array([[False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False],
               ...,
               [False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False]])
```

Check on x = windmill.png and y = lake.png, if this operation is linear, i.e., After shifting the image in the direction (k, l), shift it back in the direction (k, l). Interpret the results. Which shift is one-to-one?

2.2 Problem 3 Convolution

In this problem, we intend to explore and implement 2D convolution.

First, Create imkernel function that produces a function handle nu implementing a convolution kernel functions on the finite support (-s1, s1)x(-s2, s2). In this case, we specifies the 'gaussian' kernel as following.

Create imconvolve_naive function that performs(except around boundaries) the convolution between x and v with four loops.

Create imconvolve_spatial function that performs the convolution between x and v including around boundaries. The idea is to switch the k, l loops with the i, j loops, and then make use of imshift. The final code should read with only two loops.

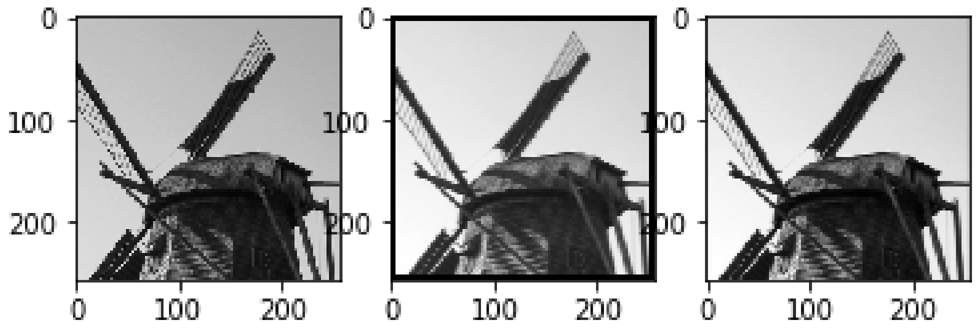
Write a script test_imconvolve function that compares the results and the execution times of imconvolve_naive and imconvolve_spatial, give comment on the execution times of two methods. You should have similar result like:

```
In [5]: import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt
        import time

        def imkernel(tau, s1, s2):
            w = lambda i,j:np.exp(-(i**2+j**2)/(2*tau**2))
            # normalization
            i,j=np.mgrid[-s1:s1,-s2:s2]
            Z = np.sum(w(i,j))
            nu = lambda i,j: w(i,j)/Z*(np.absolute(i)<= s1&np.absolute(j)<=s2)

            return nu

        # Create imconvolve_naive function,
        def imconvolve_naive(im, nu, s1, s2):
            (n1,n2)=im.shape
            win_size=(s1,s2)
            xconv = np.zeros((n1,n2))
            '''
            Your code here
            '''
            for i in range (s1,n1-s1):
                for j in range(s2,n2-s2):
                    sum=0
                    for k in range(-s1,s1+1):
                        for l in range(-s2,s2+1):
                            sum=sum+nu(k,l)*im[i-k,j-l]
                    xconv[i,j]=sum
            return xconv

        #Create imconvolve_spatial function
        #def imconvolve_spatial(x, nu,s1,s2):
            '''
            Your code here
            '''

            return xconv

        #Create imconvolve_spatial function
        def imconvolve_spatial(im,nu, s1,s2):
```

```
    '''
    Your code here
    '''
    (n1,n2)=im.shape
    win_size=(s1,s2)
    xconv = np.zeros((n1,n2))
    pading= np.zeros((n1+2*s1,n2+2*s2))
    pading[s1:n1+s1,s2:n2+s2]=im[0:n1,0:n2]
    for i in range (s1,n1+s1):
        for j in range(s2,n2+s2):
            sum=0
            for k in range (-s1,s1+1):
                for l in range (-s2,s2+1):
                    sum=sum+pading[i-k,j-l]*nu(k,l)
            pading[i,j]=sum
    xconv[0:n1,0:n2]=pading[s1:n1+s1,s2:n2+s2]

    return xconv

#Sample call and Plotting code

tau = 1
s1 = 4
s2 = 4
'''
Your code here
'''
nu=imkernel(tau, s1, s2)
im=imread('windmill.png')
original=im
plt.subplot(1,3,1)
plt.imshow(original,cmap='gray')
Naive=imconvolve_naive(im, nu, s1, s2)
plt.subplot(1,3,2)
plt.imshow(Naive,cmap='gray')
Spatial_ZP=imconvolve_spatial(im,nu,s1,s2)
plt.subplot(1,3,3)
plt.imshow(Spatial_ZP,cmap='gray')
plt.show()


start1=time.time()
imconvolve_naive(im, nu, s1, s2)
end1=time.time()
print(str(end1))


start2=time.time()
```

```
imconvolve_spatial(im,nu, s1,s2)
end2=time.time()
print(str(end2))

str(end2)>str(end1)
#the result shows that imconvolve_spatial function spends more time because it conside
#I didn't use the method as what the question requires to build imconvolve_spatial fun
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:70: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



```
1521821878.5247755
1521821922.5748947
```

```
Out[5]: True
```

** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

hw3

March 31, 2018

1 Computer Vision

1.1 Assigment 3

1.2 March 26, 2018

This assignment contains 2 programming exercises.

1.3 Problem 1: Order-statistic filtering

Order-statistic filters (OSF) are local filters that are only based on the ranking of pixel values inside a sliding window. 1. Create in imstack(img,s1,s2) function that creates a stack xstack of size n1 Œn2 Œs, which s = (2s1 +1)(2s2 +1) from the n1 Œn2 image x, such that xstack(i,j,:) contains all the values of x in the neighborhood (s1, s1) Œ (s2, s2). This function should take into account the four possible boundary conditions.

```
Hint: you can use imshift, which we implemented in assignment 1, and only two loops for s1 <= 1
```

2. Create in imosf() function function imosf(x, type, s1, s2) that implements order-statistic filters, returns xosf. imosf should first call imstack, next sort the entries of the stack with respect to the third dimension, and create the suitable output xosf according to the string type as follows:

   'median': select the median value,

   'erode': select the min value,

   'dilate': select the max value,

   'trimmed': take the mean after excluding at least 25% of the extreme values on each side.

3. Create in imopening() and imclosing() function that performs the opening and closing by the means of OSF filters.

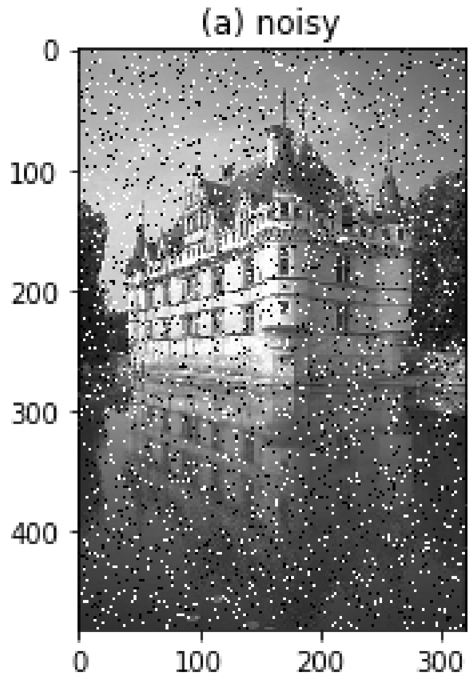
4. Load castle.png. Write a script to test imosf() that loads the image x = castle and create a corrupted version of image x with 10% of impulse noise (salt and pepper)

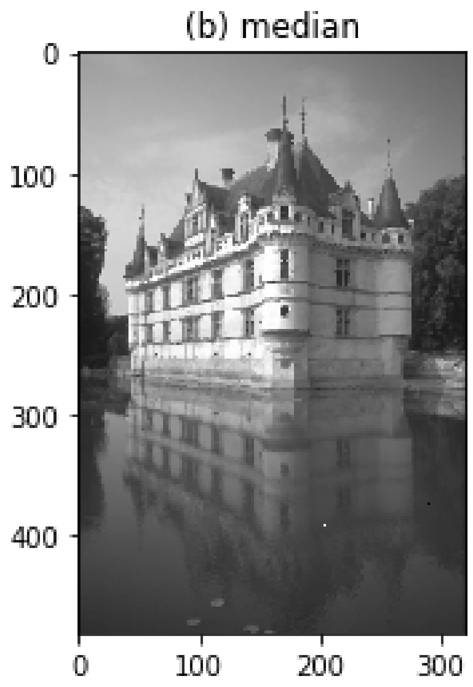
Apply your OSF filters and zoom on the results to check that your results are consistent with the following ones:

```
In [2]: import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt
        import math
        import random

        def imshift(x, p, q):
            n1=x.shape[1]#columns
            n2=x.shape[0]#rows
            k=p%n2
            l=q%n1
            xshifted=np.zeros([n2,n1])
            xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
            xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
            xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
            xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
            return xshifted

        #question(1)
        def imstack(img,s1,s2):
            (n1,n2)=img.shape
            s=(2*s1+1)*(2*s2+1)
            xstack=np.zeros((n1,n2,s),dtype=np.uint8)
            i=0
            for l in range (-s2,s2+1):
                for k in range (-s1,s1+1):
                    perimg=imshift(img,-k,l)
                    xstack[:,:,i]=perimg[:,:]
                    i=i+1
            return xstack
        #question(2)
        def imosf(x, type, s1, s2):
            (n1,n2)=x.shape
            s=(2*s1+1)*(2*s2+1)
            xosf=np.zeros((n1,n2))
            stack=imstack(x,s1,s2)
            for i in range (0,n1):
                for j in range (0,n2):
                    amount=sorted(stack[i,j])
                    amount=np.array(amount)
                    if type=='median':
                        xosf[i,j]=amount[int(amount.shape[0]/2)]
                    elif type=='erode':
                        xosf[i,j]=np.max(amount)
                    elif type=='dilate':
                        xosf[i,j]=np.min(amount)
                    else:
                        xosf[i,j]=np.mean(amount[math.ceil(amount.shape[0]/4):math.floor(amoun
            return xosf


        #question(3)
        def imclosing(img,s1,s2):
            (n1,n2)=img.shape
            imgclose=np.zeros((n1,n2))
            imgclose=imosf(img,'dilate', s1, s2)
            i=0
            while i<3:
                imgclose=imosf(imgclose,'erode',s1,s2)
                i=i+1
            return imgclose

        def imopening(img,s1,s2):
            (n1,n2)=img.shape
            imgopen=np.zeros((n1,n2))
            imgopen=imosf(img, 'erode', s1, s2)
            i=0
            while i<3:
                imgopen=imosf(imgopen,'dilate',s1,s2)
                i=i+1
            return imgopen

        #question4
        def imnoising(img):
            p=0
            (n1,n2)=img.shape
            n=n1*n2
            while p<n//10:
                i=np.random.randint(0,n1)
                j=np.random.randint(0,n2)
                noise=random.random()
                if noise>0.5:
                    noise=255
                else:
                    noise=0
                img[i,j]=noise
                p=p+1
            return img


        s1=1
        s2=1
        pic=imread('castle.png')
        pic_noise=imnoising(pic)
        plt.imshow(pic_noise,cmap='gray')
        plt.title('(a) noisy')
```

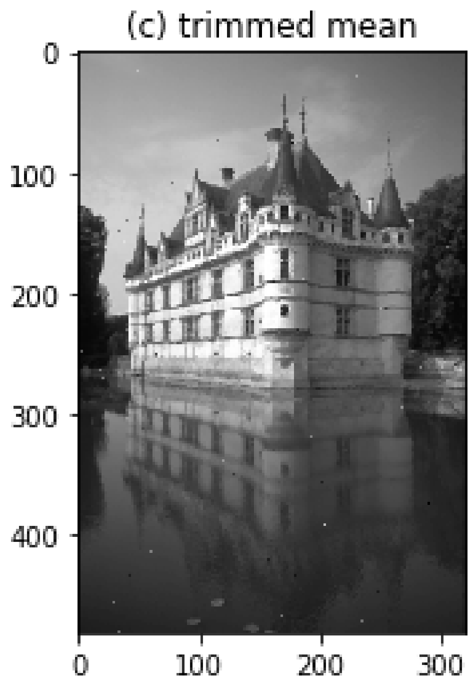
```
        plt.show()
        pic_median=imosf(pic_noise, 'median', s1, s2)
        plt.imshow(pic_median,cmap='gray')
        plt.title('(b) median')
        plt.show()
        pic_trimmed=imosf(pic_noise, 'trimmed', s1, s2)
        plt.imshow(pic_trimmed,cmap='gray')
        plt.title('(c) trimmed mean')
        plt.show()
        pic_close=imclosing(pic,s1,s2)
        plt.imshow(pic_close,cmap='gray')
        plt.title('(d) closing')
        plt.show()
        pic_open=imopening(pic,s1,s2)
        plt.imshow(pic_open,cmap='gray')
        plt.title('(e) opening')
        plt.show()
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:93: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

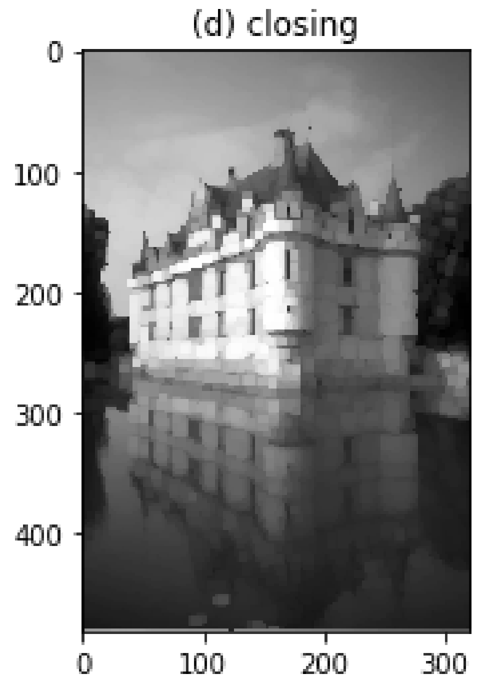


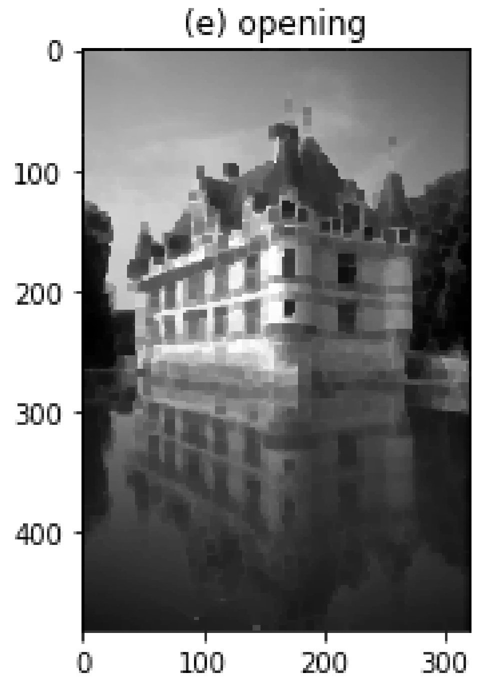


(b) median
0
100
200
300
400
0
100
200
300



(c) trimmed mean
0
100
200
300
400
0
100
200
300


(d) closing
0
100
200
300
400
0
100
200
300


(e) opening
0
100
200
300
400
0
100
200
300

1.4 Problem 2: Bilateral filter

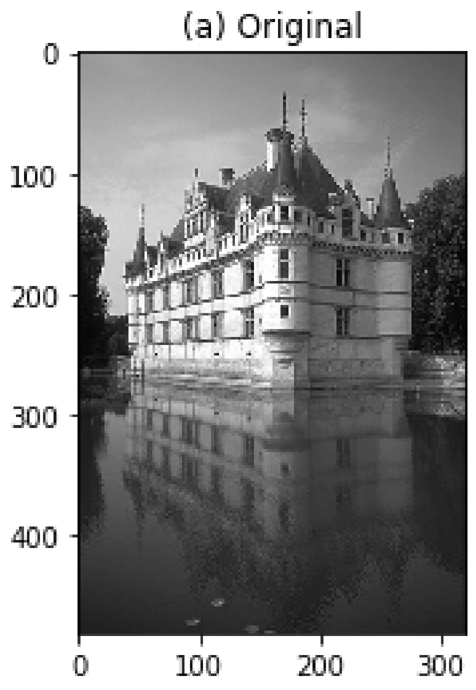
Now, we will discuss a non-local filter, Bilateral filter.
The bilateral filter is a denoising algorithm that reads as:

1. Create a test_imbilateral(img, sigma) function that loads the image x = castle and adds additive white Gaussian noise of standard deviation = 10
2. Create in imbilateral_naive(img, sigma, s1, s2, h), a function that implements the bilateral filter (except around boundaries) with four loops.
3. test your function on y with s1 = s2 = 10 and h = 1. Zoom on the results to check that your functions are consistent with the following ones:
4. Create function imbilateral(y, sigma, s1, s2, h)that implements the bilateral filter including around boundaries. The idea is again to switch the k, l loops with the i, j loops, and then make use of imshift. The final code should read with only two loops and deal with boundary conditions.
5. Compare the computation times.
6. Increase the noise level, and play with the search window sizes s1 and s2 and filtering parameter h.

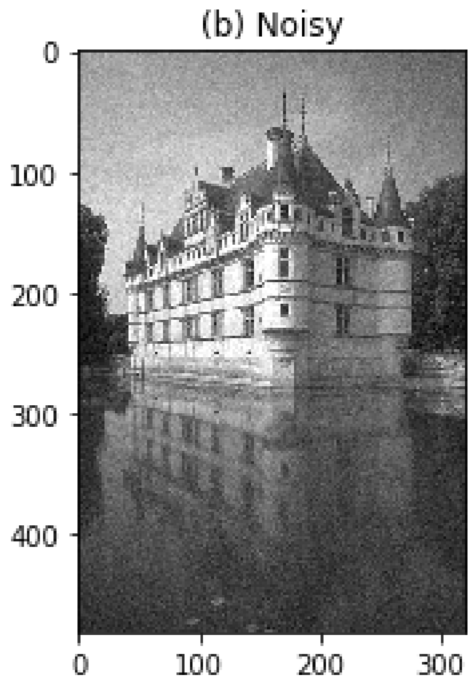
```
In [3]: import numpy as np
        from scipy.misc import imread, imsave
        import matplotlib.pyplot as plt
        import math

        #question (1)
        def test_imbilateral(img, sigma0):
            (n1,n2)=img.shape
            img_test=img+sigma0*np.random.randn(n1,n2)
            for i in range (0,n1):
                for j in range (0,n2):
                    if img_test[i,j]>255:
                        img_test[i,j]=255
                    if img_test[i,j]<0:
                        img_test[i,j]=0
            return img_test

        #question(2),(3)
        def imbilateral_naive(img, sigma, s1, s2, h):
            (n1,n2)=img.shape
            naive=np.zeros((n1,n2), dtype=np.uint8)
            dominator=16*h*sigma**2
            for i in range (s1,n1-s1):
                for j in range (s2,n2-s2):
                    sum=0
                    Z=0
            for k in range (-s1,s1+1):
                for l in range (-s2,s2+1):
                    phi=math.exp(max(2*sigma**2-(img[i+k,j+l]-img[i,j])**2,0)/dominato
                    sum=sum+phi*img[i,j]
                    Z=Z+phi
            naive[i,j]=sum/Z
    return naive

#question(4),(5),(6)
def imshift(x, p, q):
    n1=x.shape[1]#columns
    n2=x.shape[0]#rows
    k=p%n2
    l=q%n1
    xshifted=np.zeros([n2,n1])
    xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
    xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
    xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
    xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
    return xshifted

def imbilateral(img, sigma, s1, s2, h):
    (n1,n2)=img.shape
    naive=np.zeros((n1,n2), dtype=np.uint8)
    dominator=16*h*sigma**2
    sum=np.zeros((n1,n2))
    bilateral=np.zeros((n1,n2))
    numerator=np.zeros((n1,n2))
    z=np.zeros((n1,n2))
    phi=np.zeros((n1,n2))
    for k in range (-s1,s1+1):
        for l in range (-s2,s2+1):
            numerator=2*sigma**2-np.power(imshift(img,k,l)-img,2)
            numerator[numerator>0]=0
            phi=np.exp(numerator/dominator)
            z=z+phi
            sum=sum+phi*imshift(img,k,l)
    numerator=sum/z
    return numerator




sigma0=10
h=1
s1=10
s2=10
```

```
        sigma=3
        pic=imread('castle.png')
        plt.imshow(pic,cmap='gray')
        plt.title('(a) Original')
        imsave('Original.png',pic)
        plt.show()
        pic_noise=test_imbilateral(pic, sigma0)
        plt.imshow(pic_noise,cmap='gray')
        plt.title('(b) Noisy')
        imsave('Noisy.png',pic_noise)
        plt.show()
        pic_naive=imbilateral_naive(pic_noise, sigma, s1, s2, h)
        plt.imshow(pic_naive,cmap='gray')
        plt.title('(c) naive bilateral')
        imsave('naive bilateral.png',pic_naive)
        plt.show()
        pic_bilateral=imbilateral(pic_noise, sigma, s1,s2, h)
        plt.imshow(pic_bilateral,cmap='gray')
        plt.title('(d) bilateral')
        imsave('bilateral.png',pic_bilateral)
        plt.show()
```

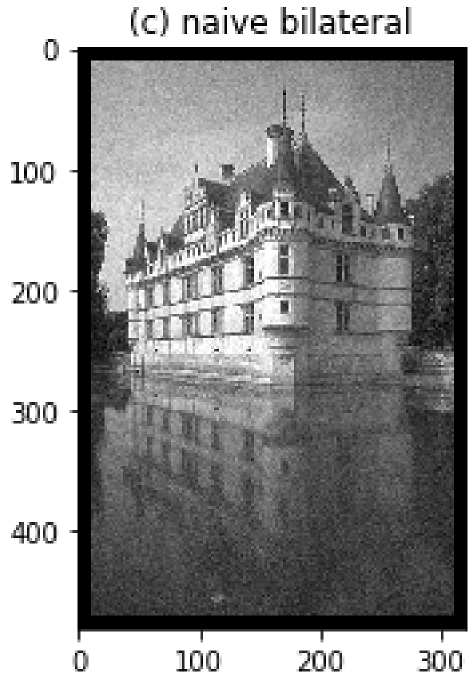
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:76: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:79: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



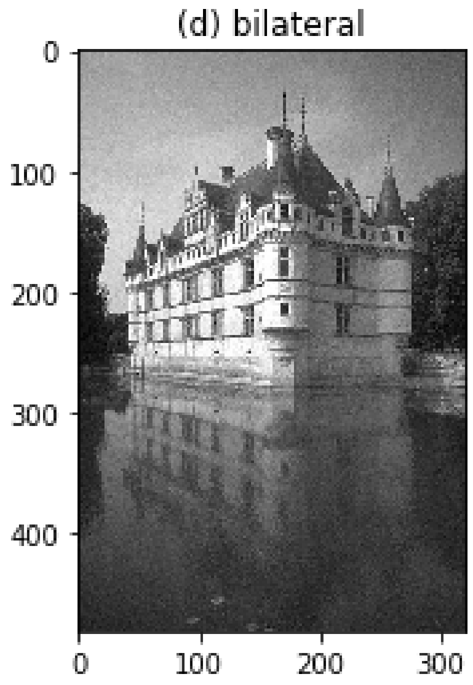
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:84: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:89: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:94: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

hw4

April 5, 2018

1 Computer Vision

1.1 Assigment 4

1.2 April 1, 2018

This assignment contains 2 programming exercises.

1.3 Problem 1: Canny Edge Detection

In this problem, you are required to write a function that performs Canny Edge Detection1. The function has the following specifications:
 It takes in two inputs: a grayscale image (geisel.jpg), and a threshold te.
 It returns the edge image.
 You are allowed the use of loops.
A brief description of the algorithm is given below. Make sure your function reproduces each step as given.

(i) Smoothing:

It is inevitable that all images taken from a camera will contain some amount of noise. To prevent noise from being mistaken for edges, noise must be reduced. Therefore the image is first smoothed by applying a Gaussian filter. A Gaussian kernel with standard deviation = 1.4 (shown below) is to be used.
You can perform this filtering step by using the scipy.signal.convolve2d( ) function.

(ii) Finding Gradients:

The next step is to find the horizontal and vertical gradients of the smoothed image using the Sobel operators. The gradient images in the x and y-direction, Gx and Gy are found by applying the kernels kx and ky given below. These operations can be performed using scipy.signal.convolve2d() in the same manner as before.
The corresponding gradient magnitude image is computed using:
and the edge direction image is calculated as follows:

(iii) Non-maximum Suppression (NMS):

The purpose of this step is to convert the thick edges in the gradient magnitude image to ”sharp” edges. This is done by preserving all local maxima in the gradient image, and deleting everything else. This is carried out by recursively performing the following steps for each pixel in the gradient image:

Round the gradient direction to nearest 45, corresponding to the use of an 8-connected neighbourhood.

Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction i.e. if the gradient direction is north (= 90), then compare with the pixels to the north and south.

If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (remove) the value.

(iv) Thresholding:

The edge-pixels remaining after the NMS step are (still) marked with their strength. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to remove these would be to use a threshold, so that only edges stronger that a certain value would be preserved. Use the input te to perform thresholding on the non-maximum suppressed magnitude image.

Evaluate your canny edge detection function on geisel.jpg for a suitable value of te that retains the structural edges, and removes the noisy ones.

Things to turn in:

Image afyer smoothing, the original gradient magnitude image, the image after NMS, and the final edge image after thresholding.

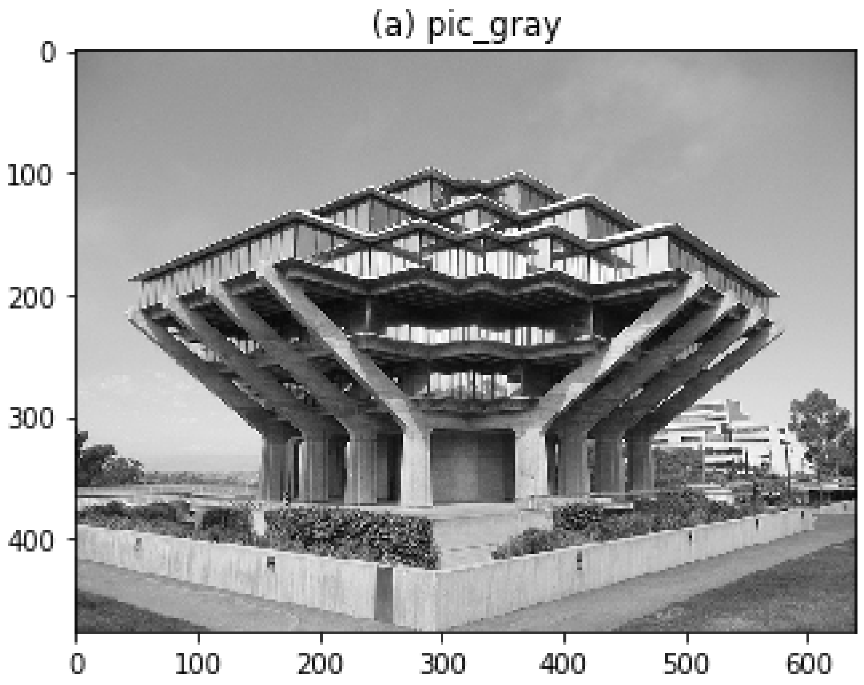
The value for te that you used to produce the final edge image.

```
In [2]: import numpy as np
        from scipy.misc import *
        import matplotlib.pyplot as plt
        import math
        from scipy import signal

        def rgb2gray(rgb):
            grayimg=np.sum(rgb,axis=2)/3
            return grayimg

        #question(1):
        def smoothing(img):
            kernel=np.array([[2,4,5,4,2],[4,9,12,9,4],[5,12,15,12,5],[4,9,12,9,4],[2,4,5,4,2]]
            smoothed_img=signal.convolve2d(img, kernel, boundary="symm")
            return smoothed_img

        #question(2):
        def Gradients(img):
            kx=np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
            ky=np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
            (n1,n2)=img.shape
            Gx=signal.convolve2d(img, kx, boundary="symm")
```

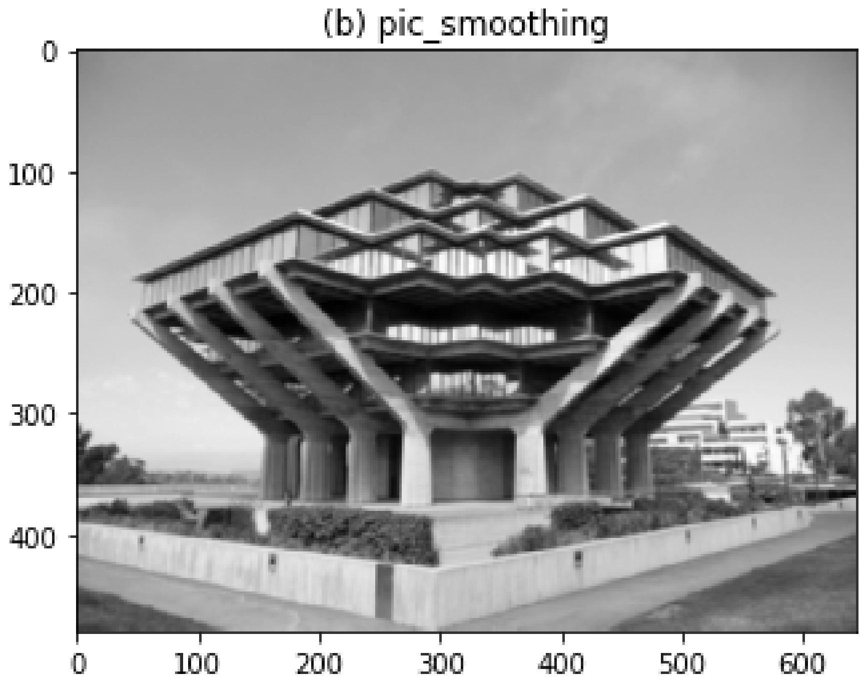
```
    Gy=signal.convolve2d(img, ky, boundary="symm")
    G=np.sqrt(np.power(Gx,2)+np.power(Gy,2))
    return G/G.max()*255


def Gradients_theta(img):
    kx=np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    ky=np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    (n1,n2)=img.shape
    Gx=signal.convolve2d(img, kx, boundary="symm")
    Gy=signal.convolve2d(img, ky, boundary="symm")
    G_theta=np.arctan2(Gy,Gx)
    return G_theta

#question(3):
def NMS(img):
    (n1,n2)=img.shape
    remainder=Gradients_theta(img)%(math.pi/2)
    for i in range (0,n1):
        for j in range (0,n2):
            if 0<remainder[i,j] and remainder[i,j]<=math.pi/8:
                remainder[i,j]=math.pi/4
            if math.pi/8<remainder[i,j] and remainder[i,j]<=math.pi*3/8:
                remainder[i,j]=math.pi/4
            if math.pi*3/8<remainder[i,j] and remainder[i,j]<=math.pi/2:
                remainder[i,j]=math.pi/2
    theta=(math.pi/2)*(Gradients_theta(img)//(math.pi/2))+remainder
    for i in range (1,n1-1):
        for j in range (1,n2-1):
            if theta[i,j]%(math.pi)==0:
                if img[i,j]<img[i,j-1] or img[i,j]<img[i,j+1]:
                    img[i,j]=0
            elif theta[i,j]%(math.pi)==math.pi/2:
                if img[i,j]<img[i-1,j] or img[i,j]<img[i+1,j]:
                    img[i,j]=0
            elif theta[i,j]%(math.pi)==math.pi/4:
                if img[i,j]<img[i+1,j-1] or img[i,j]<img[i-1,j+1]:
                    img[i,j]=0
            elif theta[i,j]%(math.pi)==math.pi*3/4:
                if img[i,j]<img[i-1,j-1] or img[i,j]<img[i+1,j+1]:
                    img[i,j]=0
    return img

#question(4)
def canny_edge(img, te):
    #img[img>=te]=255
    img[img < te] = 0
    return img
```

```
        pic_color=imread('geisel.jpg')
        pic_gray=rgb2gray(pic_color)
        plt.imshow(pic_gray,cmap='gray')
        plt.title('(a) pic_gray')
        plt.show()
        pic_smoothing=smoothing(pic_gray)
        plt.imshow(pic_smoothing,cmap='gray')
        imsave('pic_smoothing.png',pic_smoothing)
        plt.title('(b) pic_smoothing')
        plt.show()
        pic_gradient=Gradients(pic_smoothing)
        plt.imshow(pic_gradient,cmap='gray')
        imsave('pic_gradient.png',pic_gradient)
        plt.title('(c) pic_gradient')
        plt.show()
        pic_NMS=NMS(pic_gradient)
        imsave('pic_NMS.png',pic_NMS)
        plt.imshow(pic_NMS,cmap='gray')
        plt.title('(d) pic_NMS')
        plt.show()
        pic_canny=canny_edge(pic_NMS,50)
        plt.imshow(pic_canny, cmap='gray')
        imsave('pic_canny.png',pic_canny)
        plt.title('(e) pic_canny')
        plt.show()
```

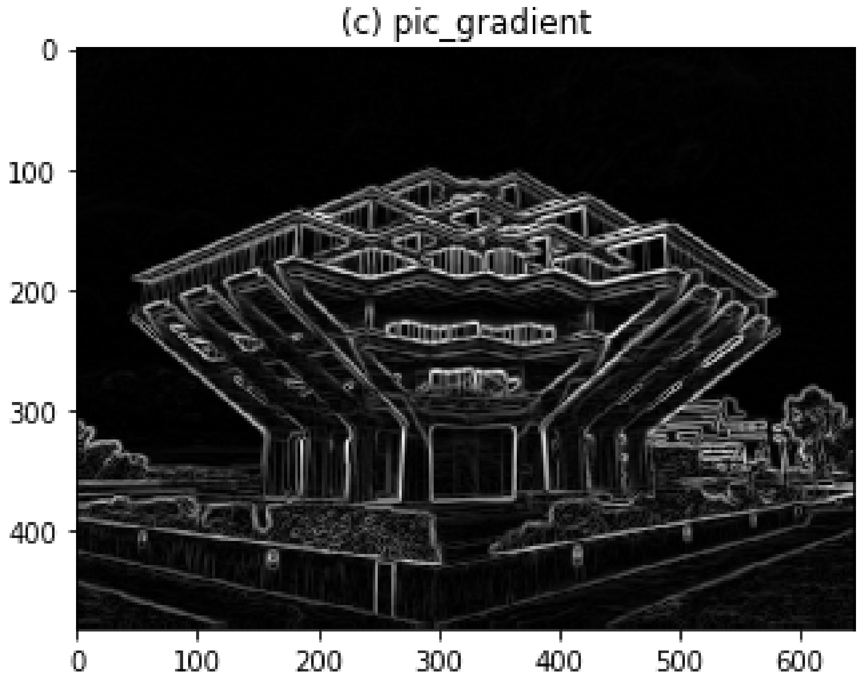
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:74: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



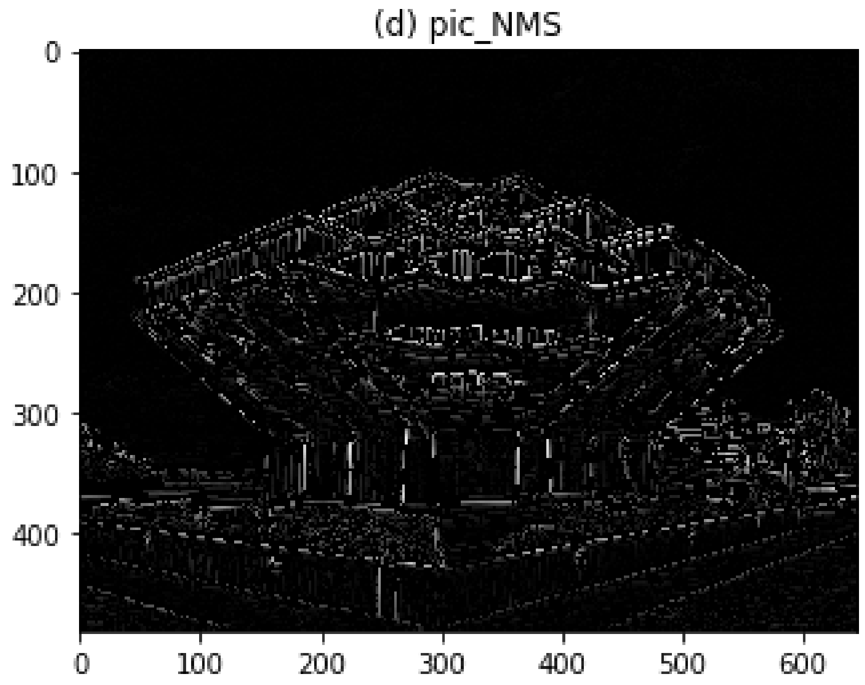
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:81: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



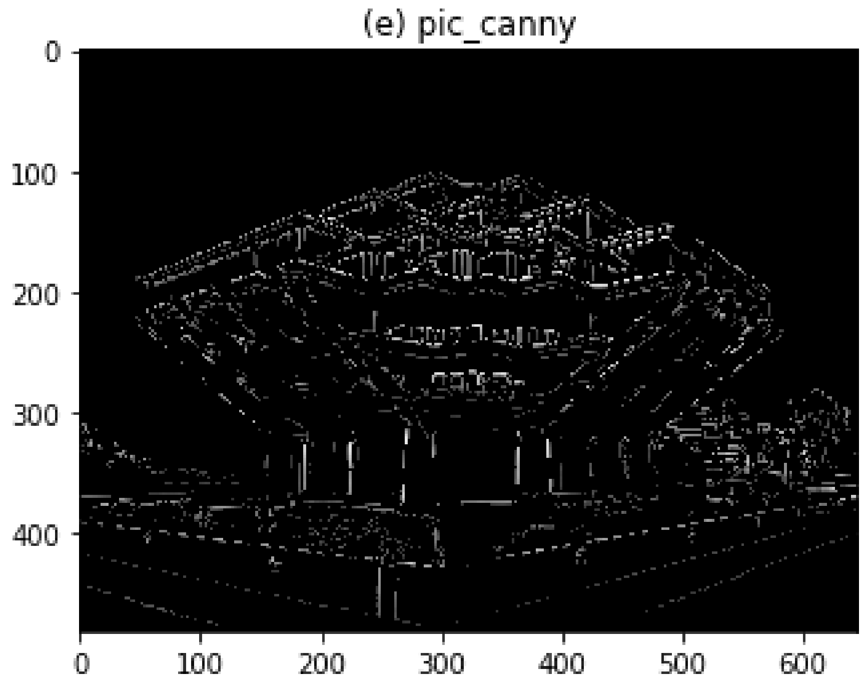
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:86: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:90: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:96: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



1.4 Problem 2 Adaptive Histogram Equalization

It is often found in image processing and related fields that real world data is unsuitable for direct use. This warrants the inclusion of pre-processing steps before any other operations are performed. An example of this is histogram equalization (HE) and its extension adaptive histogram equalization (AHE). The goal of this problem is to implement a function for AHE as described in Chapter 1 of Adaptive Histogram Equalization - A Parallel Implementation2. The function has the following specifications:

(i) The desired function AHE() takes two inputs: the image ("beach.png") im and the contextual region size win_size.

(ii) Using the pseudocode in Algorithm as a reference, compute the enhanced image after AHE.

(iii) You may use loops if necessary. You should not make use of any inbuilt functions for AHE or HE.

(iv) The function returns one output: the enhanced image after AHE.

Evaluate your function on the image beach.png for win size = 33, 65 and 129. In your report, include the original image, the 3 images after AHE. Make sure to resize all images to ensure they do not take up too much space. Additionally, include your answers (no more than three sentences each) to the following questions:
How does the original image qualitatively compare to the images after AHE?

```
In [2]: def AHE(im, win_size):
            '''
            Your code here
            '''
            return output


        #Import image here
        #Sample call
        #Plotting code below
        '''
        Your code here
        '''

Out[2]: '\nYour code here\n'
```

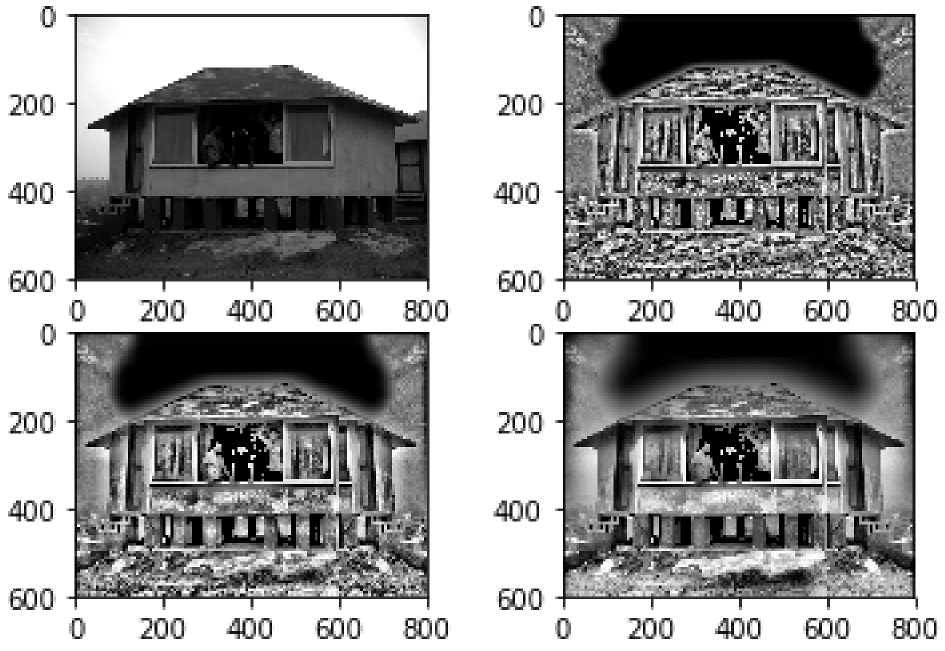
** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

question 2-two loops

April 5, 2018

```
In [7]: import numpy as np
        from scipy.misc import *
        import matplotlib.pyplot as plt
        import math

        def AHE(img, win_size):

            # (1) pre-requirement Padding
            s=int((win_size-1)/2)
            (n1,n2)=img.shape
            img_pad=np.zeros((n1+2*s,n2+2*s))
            img_pad[s:s+n1, s:s+n2]=img[:,:]
            for i in range (0,s-1):
                img_pad[i,s:s+n2]=img[s-i-1,:]
            for i in range (s+n1+1,2*s+n1):
                img_pad[i,s:s+n2]=img[n1-1-i, :]
            for j in range (0,s-1):
                img_pad[0:2*s+n1,j]=img_pad[0:2*s+n1,2*s-j-1]
            for j in range (s+n1+1,2*s+n2):
                img_pad[0:2*s+n1,j]=img_pad[0:2*s+n1,2*s+n2-j]
            #return img_pad

            # (2) AHE operation
            output=np.zeros((n1,n2))
            rank=np.zeros((n1,n2))
            for x in range (s,s+n1):
                for y in range (s,s+n2):
                    window=img_pad[x-s:x+s+1,y-s:y+s+1].reshape(-1)
                    rank[x - s, y - s] = np.sum(window < img_pad[x, y])
            output=(rank * 255 / (win_size * win_size))
            return output


            output=np.zeros((n1,n2))
            for x in range (s,s+n1):
                for y in range (s,s+n2):
                    rank=0
                for i in range (-s,s+1):
                    for j in range (-s,s+1):
                        if img_pad[x,y]>img_pad[x+i,y+j]:
                            rank=rank+1
                img_pad[x,y]=(rank*255)/(win_size**2)
        output[:,:]=img_pad[s:s+n1, s:s+n2]
        return output


#Import image here
#Sample call
#Plotting code below
'''
Your code here
'''
pic=imread('beach.png')
plt.subplot(2,2,1)
plt.imshow(pic, cmap='gray')
win_size=33
pic_AHE_33=AHE(pic,win_size)
plt.subplot(2,2,2)
plt.imshow(pic_AHE_33, cmap='gray')
imsave('beach_AHE_33.png',pic_AHE_33)
win_size=65
pic_AHE_65=AHE(pic,win_size)
plt.subplot(2,2,3)
plt.imshow(pic_AHE_65, cmap='gray')
imsave('beach_AHE_65.png',pic_AHE_33)
win_size=129
pic_AHE_129=AHE(pic,win_size)
plt.subplot(2,2,4)
plt.imshow(pic_AHE_129, cmap='gray')
imsave('beach_AHE_129.png',pic_AHE_129)
plt.show()
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:41: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:48: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:53: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:58: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



0
200
400
600
0
200
400
600
800

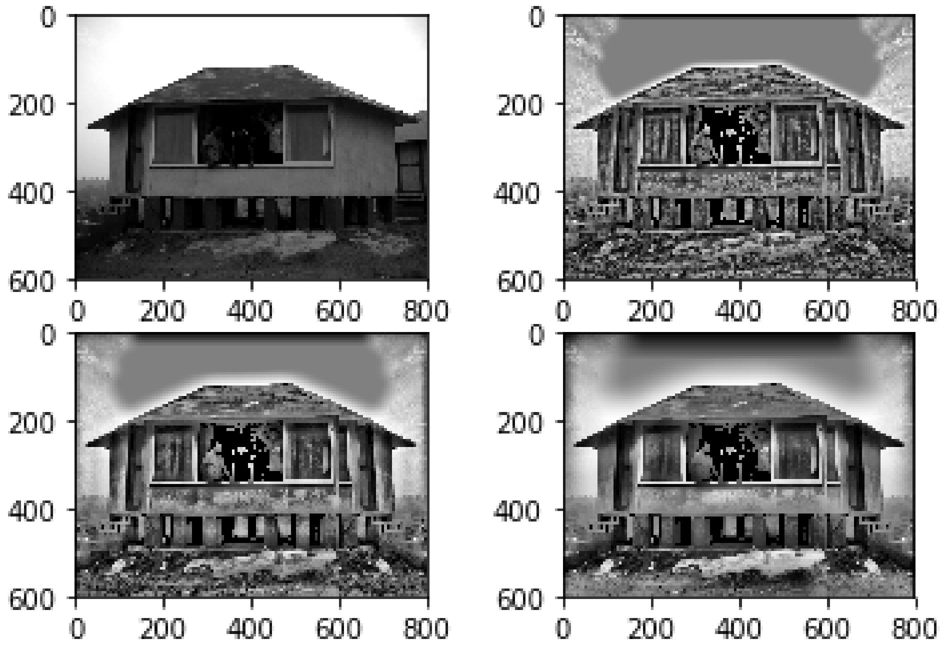
question 2-four loops

April 5, 2018

```
In [56]: import numpy as np
         from scipy.misc import *
         import matplotlib.pyplot as plt
         import math

         def AHE(img, win_size):

             # (1) pre-requirement Padding
             s=int((win_size-1)/2)
             (n1,n2)=img.shape
             img_pad=np.zeros((n1+2*s,n2+2*s))
             img_pad[s:s+n1, s:s+n2]=img[:,:]
             for i in range (0,s-1):
                 img_pad[i,s:s+n2]=img[s-i-1,:]
             for i in range (s+n1+1,2*s+n1):
                 img_pad[i,s:s+n2]=img[n1-1-i, :]
             for j in range (0,s-1):
                 img_pad[0:2*s+n1,j]=img_pad[0:2*s+n1,2*s-j-1]
             for j in range (s+n1+1,2*s+n2):
                 img_pad[0:2*s+n1,j]=img_pad[0:2*s+n1,2*s+n2-j]
             #return img_pad

             # (2) AHE operation
             output=np.zeros((n1,n2))
             for x in range (s,s+n1):
                 for y in range (s,s+n2):
                     rank=0
                     for i in range (-s,s+1):
                         for j in range (-s,s+1):
                             if img_pad[x,y]>img_pad[x+i,y+j]:
                                 rank=rank+1
                     img_pad[x,y]=(rank*255)/(win_size**2)
             output[:,:]=img_pad[s:s+n1, s:s+n2]
             return output
```

```
#Import image here
#Sample call
#Plotting code below
'''
Your code here
'''
pic=imread('beach.png')
plt.subplot(2,2,1)
plt.imshow(pic, cmap='gray')
win_size=33
pic_AHE_33=AHE(pic,win_size)
plt.subplot(2,2,2)
plt.imshow(pic_AHE_33, cmap='gray')
imsave('beach_AHE_33.png',pic_AHE_33)
win_size=65
pic_AHE_65=AHE(pic,win_size)
plt.subplot(2,2,3)
plt.imshow(pic_AHE_65, cmap='gray')
imsave('beach_AHE_65.png',pic_AHE_33)
win_size=129
pic_AHE_129=AHE(pic,win_size)
plt.subplot(2,2,4)
plt.imshow(pic_AHE_129, cmap='gray')
imsave('beach_AHE_129.png',pic_AHE_129)
plt.show()
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:44: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:51: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:56: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:61: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```


0
200
400
600
0
200
400
600
800

hw5

April 15, 2018

1 Computer Vision

1.1 Assigment 5

1.2 April 8, 2018

This assignment contains 2 programming exercises.

1.3 Problem 1: NL-means filter

The NL-means filter is an extension of the bilateral filter, where instead of averaging values of pixels with similar values, the values of pixels centered on similar patches are averaged. The NL-means reads as:

1. Create in imnlmeans_naive(img, sigma, s1, s2, p1,p2, h) function that implements the NL-means filter with six loops.
2. Create a script test_imnlmeans function that loads the image x = zebra.png and creates y by adding additive white Gaussian noise. Test your function on y with s1 = s2 = 10, p1 = p2 = 3 and h = 1.
3. implement the NL-means filter (including around boundaries) with less loops to reduce computaional time. (Hint: use imshift function and seperable property for spatial image convolve.)
4. Complete test_imnlmeans function to test your new function. Compare the results and check that your new function is about 7 times faster

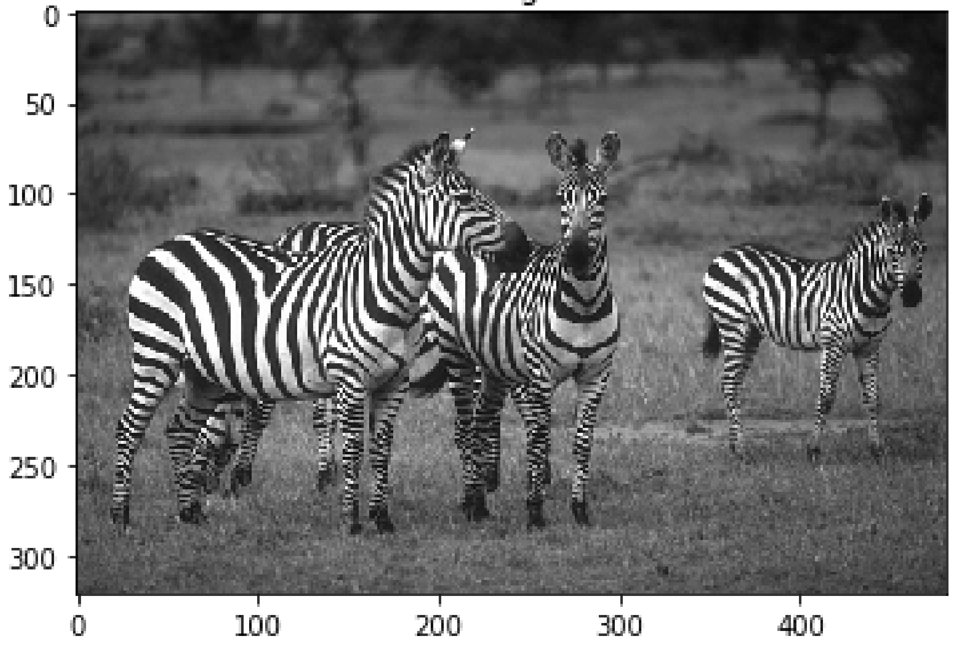
```
In [3]: import numpy as np
        from scipy.misc import imread, imsave
        import matplotlib.pyplot as plt
        import math

        #question(1),(2)
        def imnlmeans_naive(img, sigma, s1, s2, p1,p2, h):
            n1,n2=img.shape
            p=(2*p1+1)*(2*p2+1)
            for i in range (p1+s1,n1-s1-p1):
```

```
        for j in range (p2+s2,n2-s2-p2):
            z=0
            sum=0
            for k in range (-s1,s1+1):
                for l in range (-s2,s2+1):
                    alpha=0
                    for u in range (-p1,p1+1):
                        for v in range (-p2,p2+1):
                            temp=(img[i+k+u,j+l+v]-img[i+u,j+v])**2
                            alpha=alpha+temp
                    phi=math.exp(max(alpha/p-2*sigma**2,0)/((-16*h*sigma**2)/p))
                    z=z+phi
                    sum=sum+phi*img[i+k,j+l]
            img[i,j]=sum/z
    return img

def  noise(img, sigma0):
    n1,n2=img.shape
    img_test=img+sigma0*np.random.randn(n1,n2)
    for i in range (n1):
        for j in range (n2):
            if img_test[i,j]>255:
                img_test[i,j]=255
            if img_test[i,j]<0:
                img_test[i,j]=0
    return img_test

#question(3),(4):
def imshift(x, p, q):
    n1=x.shape[1]#columns
    n2=x.shape[0]#rows
    k=p%n2
    l=q%n1
    xshifted=np.zeros([n2,n1])
    xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
    xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
    xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
    xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
    return xshifted

def imnlmeans(img, sigma, s1, s2, p1,p2, h):
    n1,n2=img.shape
    p=(2*p1+1)*(2*p2+1)
    z=np.zeros((n1,n2))
    sum=np.zeros((n1,n2))
    for k in range (-s1,s1+1):
        for l in range (-s2,s2+1):
            alpha=np.zeros((n1,n2))
```

```
                for u in range (-p1,p1+1):
                    for v in range (-p2,p2+1):
                        temp=np.power(imshift(img, k+u, l+v)-imshift(img, u, v),2)
                        alpha=alpha+temp
                numerator=alpha/p-2*sigma**2
                numerator[numerator<0]=0
                phi=np.exp(numerator/((-16*h*sigma**2)/p))
                z=z+phi
                sum=sum+phi*imshift(img,k,l)
        x=sum/z
        return x


    sigma0=10
    sigma=10
    s1=s2=10
    p1=p2=3
    h=1
    pic=imread('zebra.png')
    plt.imshow(pic, cmap='gray')
    plt.title('(a) original')
    plt.show()
    pic_noise=noise(pic, sigma0)
    plt.imshow(pic_noise, cmap='gray')
    plt.title('(b) noised')
    imsave('noise.png',pic_noise)
    plt.show()
    NL=imnlmeans(pic_noise, sigma, s1, s2, p1,p2, h)
    plt.imshow(NL, cmap='gray')
    plt.title('(c) NL')
    imsave('NL.png',NL)
    plt.show()
    NL_naive=imnlmeans_naive(pic_noise, sigma, s1, s2, p1,p2, h)
    plt.imshow(NL_naive, cmap='gray')
    plt.title('(d) NL_naive')
    imsave('NL_naive.png',NL_naive)
    plt.show()
```

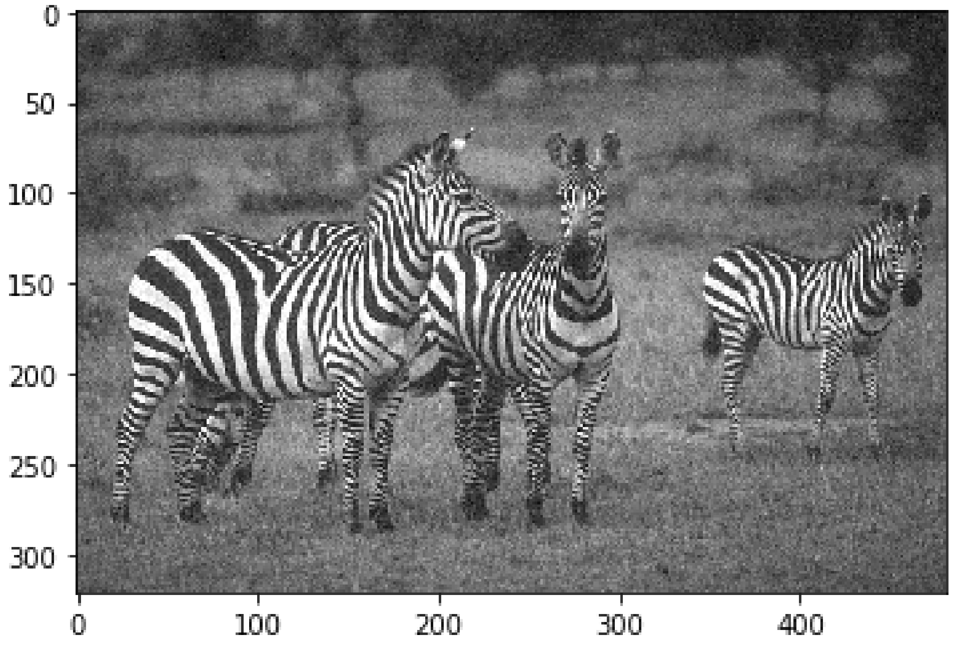
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:77: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

(a) original

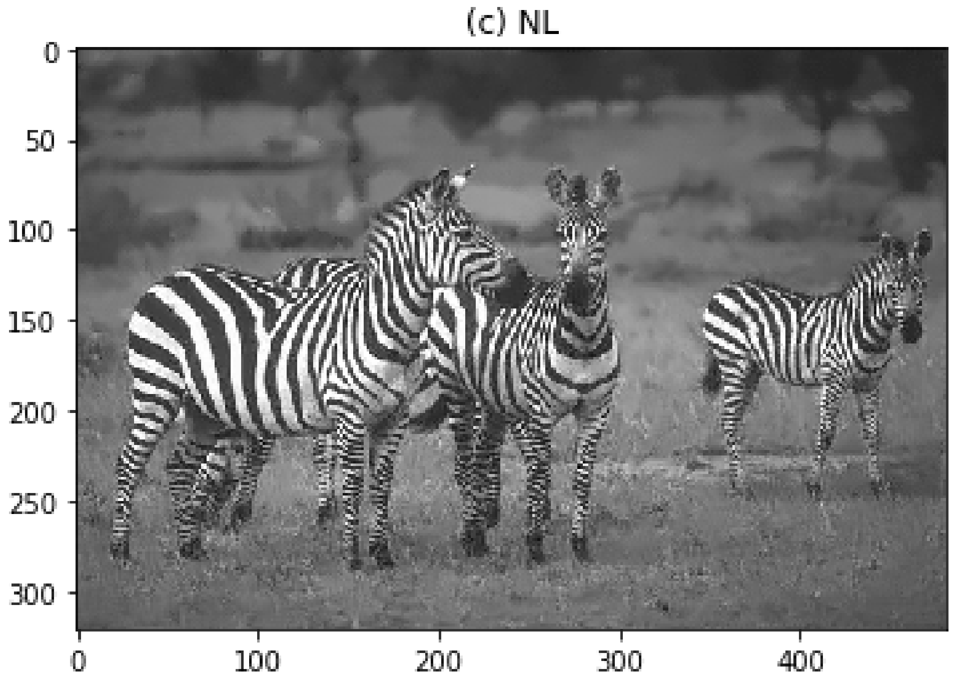


```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:84: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

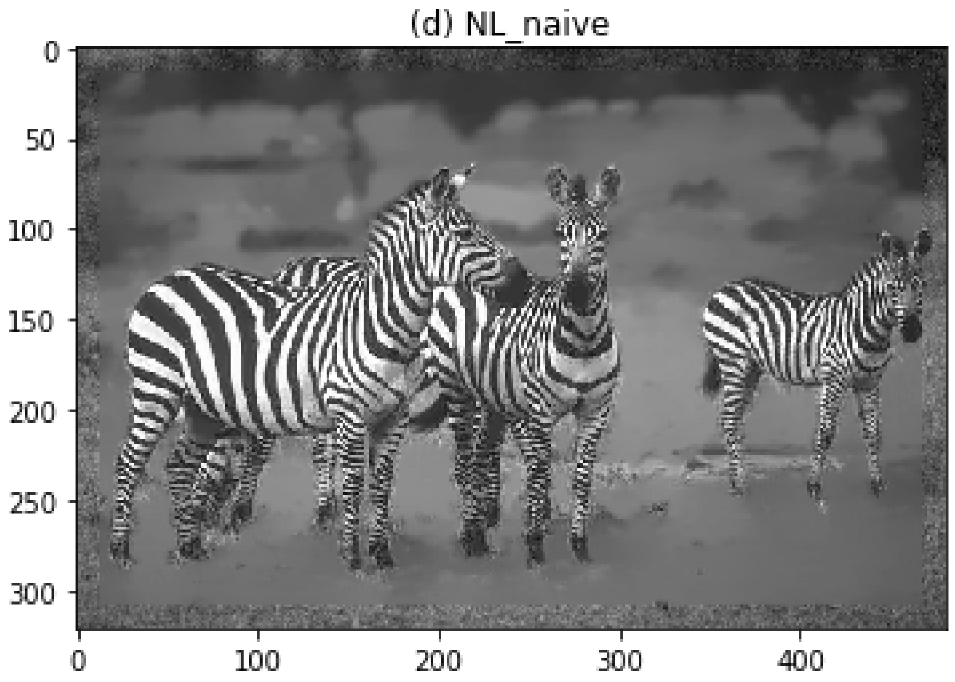
(b) noised



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:89: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:94: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



1.4 Problem 2: Block-wise NL-means filter

1, The block-wise NL-means is a variant of NL-means, where instead of averaging values of pixels with similar patches, similar patches are first averaged together:

2.Modify the imnlmean function to implement the block-wise NL-means imblocknlmeans(y, sig, s1, s2, p1, p2, h), time the function and compare the Problem1.

3.A very classical (but controversial) way to compare the quality of restoration techniques is to use the PSNR defined for images with values ranging in [0, 255] as where x is the noise-free image and x′ the estimate obtained from y. The PSNR measures in decibels (dB) the quality of the restoration: the higher the better. Implement it in impsnr function: impsnr(x,y)

4.Create a script test_imblocknlmeans function to test your new function on the corrupted version y of x = zebra with additive white Gaussian noise. Choose s1 = s2 = 10, p1 = p2 = 3 and h = 1, and compare the quality (PSNR) and the execution times of imbilateral, imnlmeans and imblocknlmeans. Check that your results are consistent with the following ones:

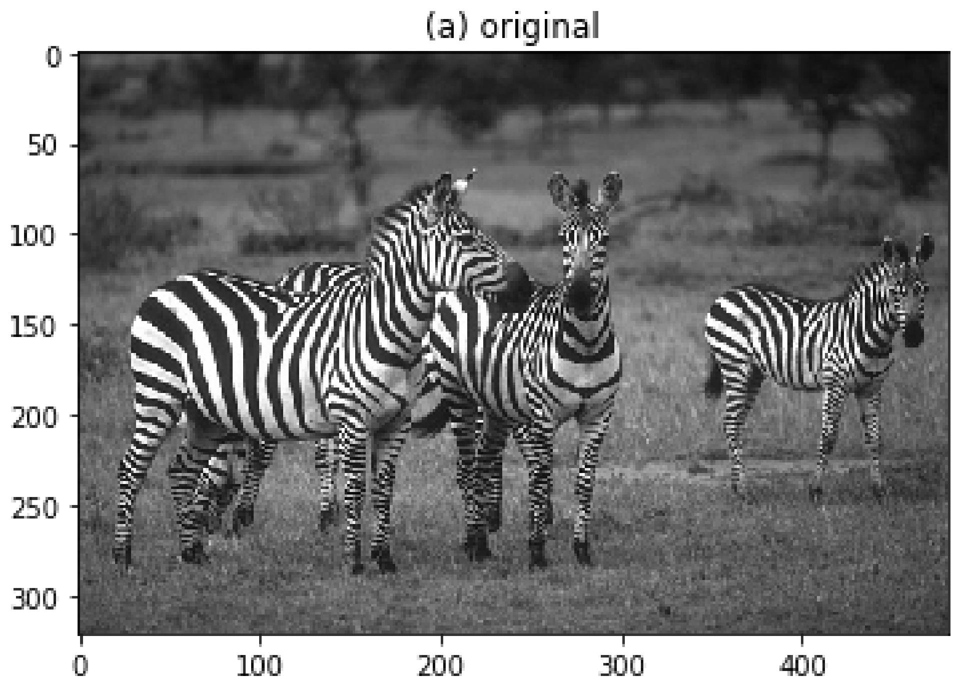
5.What is the advantage of the block-wise NL-means compared to the standard NL-means.

```
In [2]: import numpy as np
        from scipy.misc import imread, imsave
        import matplotlib.pyplot as plt
        import math

        def  noise(img, sigma0):
            n1,n2=img.shape
            img_test=img+sigma0*np.random.randn(n1,n2)
            for i in range (n1):
```

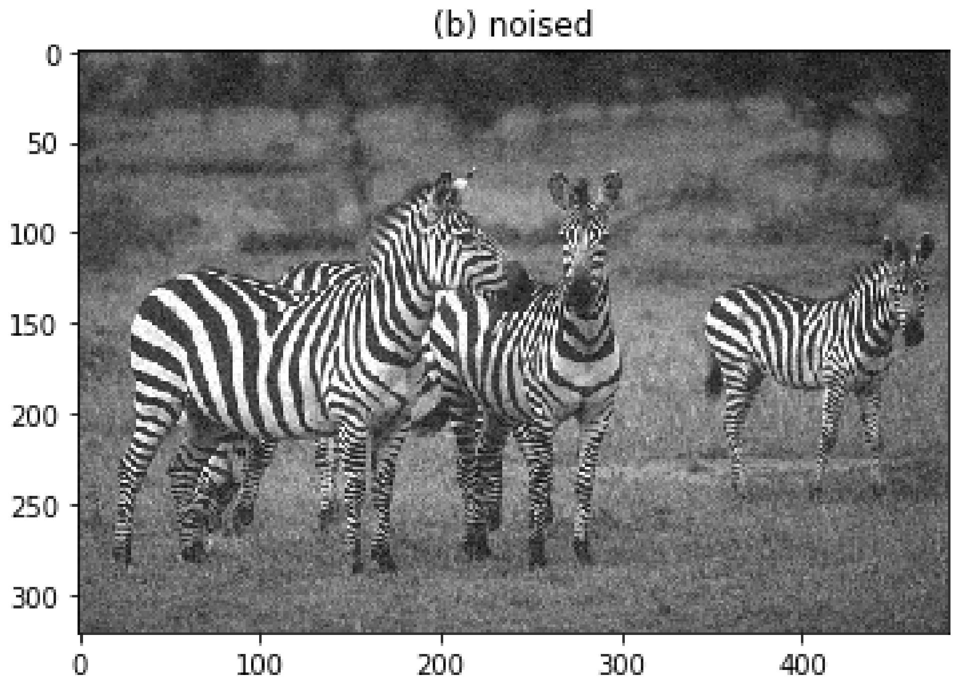
```
        for j in range (n2):
            if img_test[i,j]>255:
                img_test[i,j]=255
            if img_test[i,j]<0:
                img_test[i,j]=0
    return img_test

def imshift(x, p, q):
    n1=x.shape[1]#columns
    n2=x.shape[0]#rows
    k=p%n2
    l=q%n1
    xshifted=np.zeros([n2,n1])
    xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
    xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
    xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
    xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
    return xshifted

#bilateral filter
def imbilateral(img, sigma, s1, s2, h):
    (n1,n2)=img.shape
    naive=np.zeros((n1,n2), dtype=np.uint8)
    dominator=16*h*sigma**2
    sum=np.zeros((n1,n2))
    bilateral=np.zeros((n1,n2))
    numerator=np.zeros((n1,n2))
    z=np.zeros((n1,n2))
    phi=np.zeros((n1,n2))
    for k in range (-s1,s1+1):
        for l in range (-s2,s2+1):
            numerator=2*sigma**2-np.power(imshift(img,k,l)-img,2)
            numerator[numerator>0]=0
            phi=np.exp(numerator/dominator)
            z=z+phi
            sum=sum+phi*imshift(img,k,l)
    numerator=sum/z
    return numerator

#Non-local means filter
def imnlmeans(img, sigma, s1, s2, p1,p2, h):
    n1,n2=img.shape
    p=(2*p1+1)*(2*p2+1)
    z=np.zeros((n1,n2))
    sum=np.zeros((n1,n2))
    for k in range (-s1,s1+1):
        for l in range (-s2,s2+1):
            alpha=np.zeros((n1,n2))
            for u in range (-p1,p1+1):
                for v in range (-p2,p2+1):
                    temp=np.power(imshift(img, k+u, l+v)-imshift(img, u, v),2)
                    alpha=alpha+temp
            numerator=alpha/p-2*sigma**2
            numerator[numerator<0]=0
            phi=np.exp(numerator/((-16*h*sigma**2)/p))
            z=z+phi
            sum=sum+phi*imshift(img,k,l)
    x=sum/z
    return x

#BWNL-means filter
def imblocknlmeans(img, sigma, s1, s2, p1,p2, h):
    n1,n2=img.shape
    p=(2*p1+1)*(2*p2+1)
    z=np.zeros((n1,n2))
    sum=np.zeros((n1,n2))
    for k in range (-s1,s1+1):
        for l in range (-s2,s2+1):
            for a in range (-p1,p1+1):
                for b in range (-p2,p2+1):
                    alpha=np.zeros((n1,n2))
                    for u in range (-p1,p1+1):
                        for v in range (-p2,p2+1):
                            temp=np.power(imshift(img, k+u+a, l+v+b)-imshift(img, u+a,
                            alpha=alpha+temp
                    numerator=alpha/p-2*sigma**2
                    numerator[numerator<0]=0
                    phi=np.exp(numerator/((-16*h*sigma**2)/p))
                    z=z+phi
                    sum=sum+phi*imshift(img,k,l)
    x=sum/z
    return x

#peak-signal-to-noise r
def impsnr(x,y):
    n1,n2=x.shape
    n=(n1+1)*(n2+1)
    sum=0
    for i in range (n1):
        for j in range (n2):
            sum=sum+((x[i,j]-y[i,j])**2)/n
    psnr=10*math.log10((255**2)/sum)
    return psnr


sigma0=10
```

```
        sigma=10
        s1=s2=10
        p1=p2=1
        h=1
        pic=imread('zebra.png')
        plt.imshow(pic, cmap='gray')
        plt.title('(a) original')
        plt.show()

        pic_noise=noise(pic, sigma0)
        plt.imshow(pic_noise, cmap='gray')
        plt.title('(b) noised')
        imsave('noised.png',pic_noise)
        plt.show()

        pic_bilateral=imbilateral(pic_noise, sigma, s1, s2, h)
        plt.imshow(pic_bilateral, cmap='gray')
        plt.title('(c) bilateral')
        imsave('bilateral.png',pic_bilateral)
        plt.show()
        print(impsnr(pic,pic_bilateral))

        pic_NL=imnlmeans(pic_noise, sigma, s1, s2, p1,p2, h)
        plt.imshow(pic_NL, cmap='gray')
        plt.title('(d) pic_NL')
        imsave('NL.png',pic_NL)
        plt.show()
        print(impsnr(pic,pic_NL))

        BWNL=imblocknlmeans(pic_noise, sigma, s1, s2, p1,p2, h)
        plt.imshow(BWNL, cmap='gray')
        plt.title('(e) BWNL')
        imsave('BWNL.png',BWNL)
        plt.show()
        print(impsnr(pic,BWNL))
```

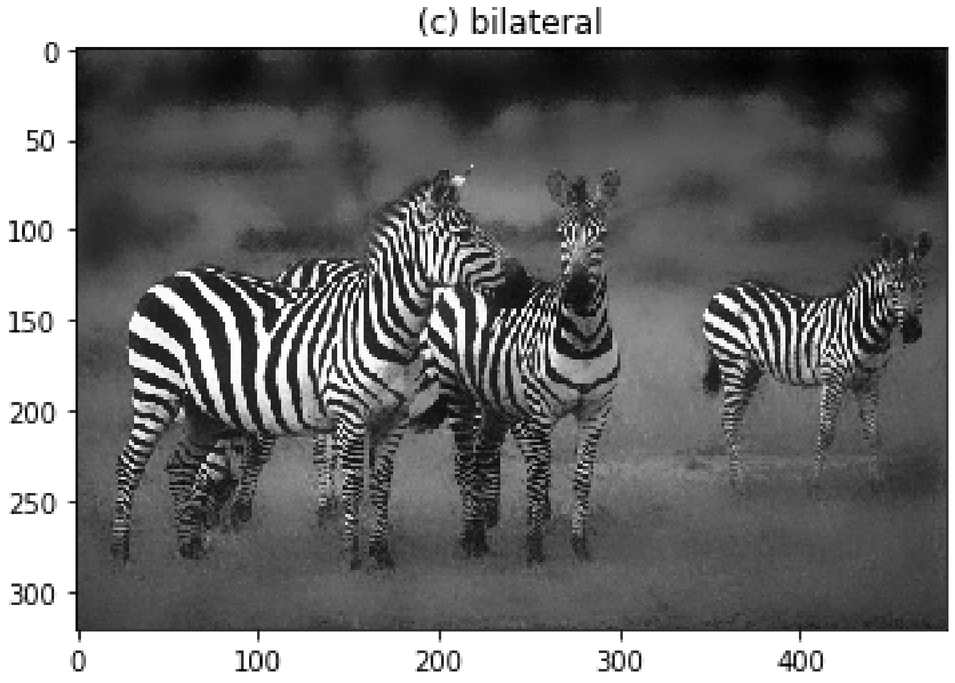
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:110: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:118: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

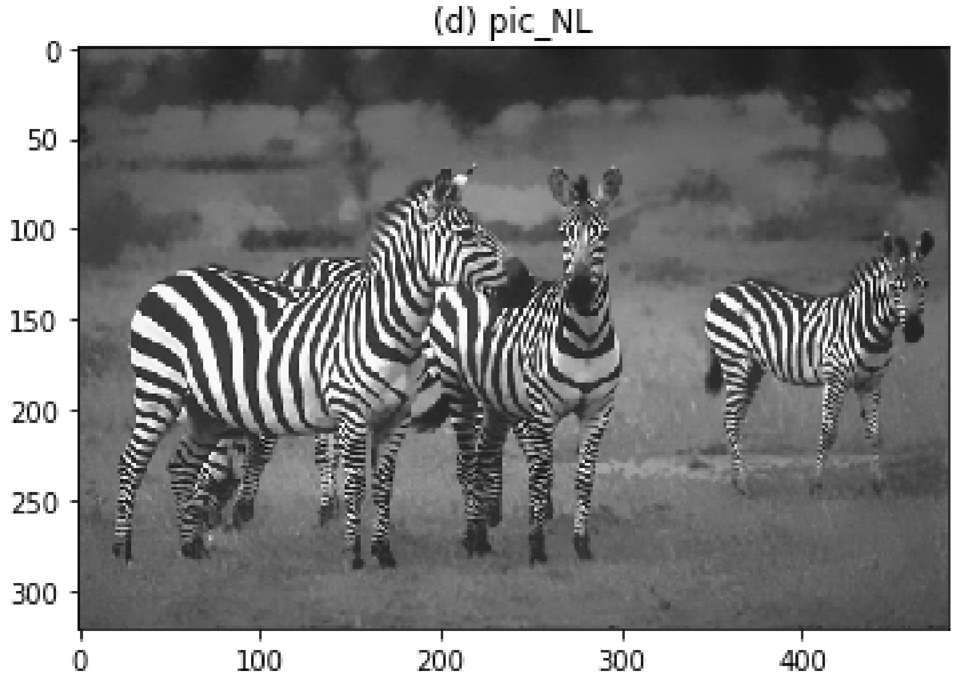


```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:124: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



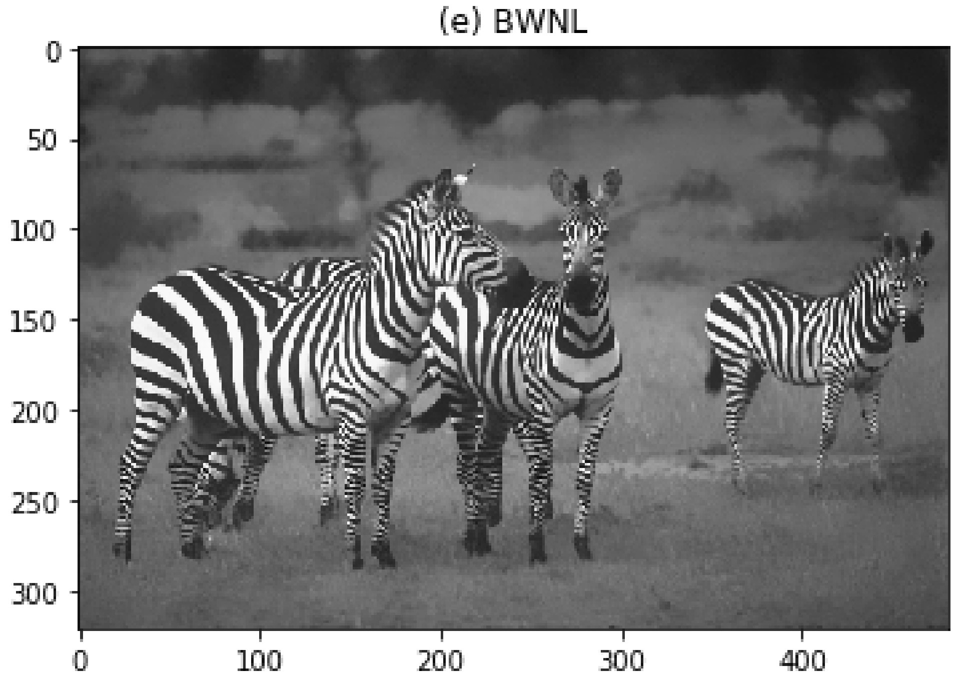
```
28.366511002938893
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:131: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
31.259399900846375
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:138: DeprecationWarning: `imsave` is deprec
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
30.679987187791063
```

** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

phi_matrix_table

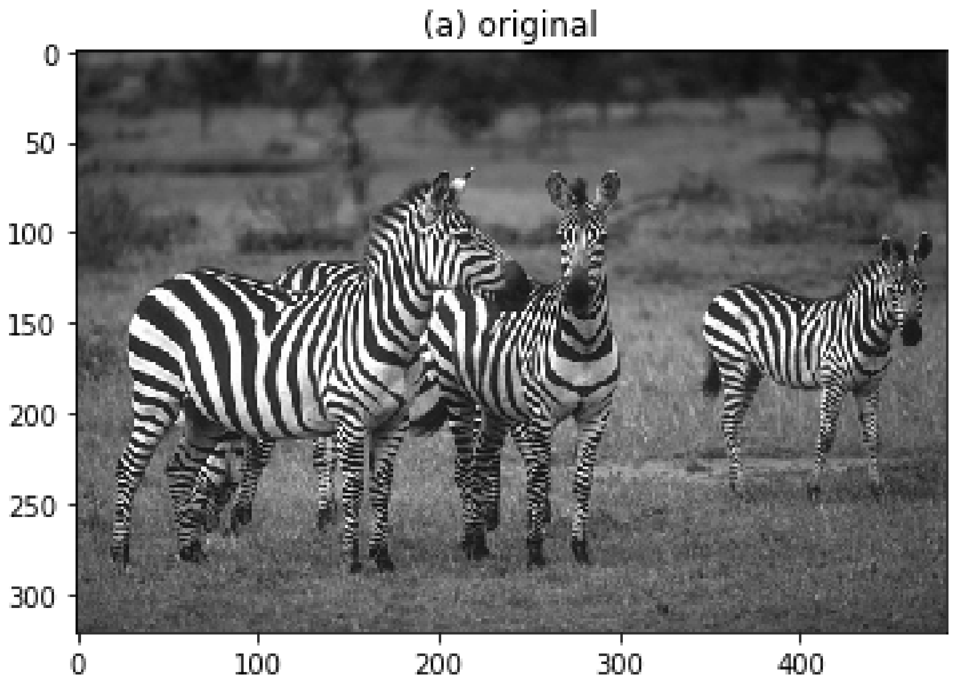
April 15, 2018

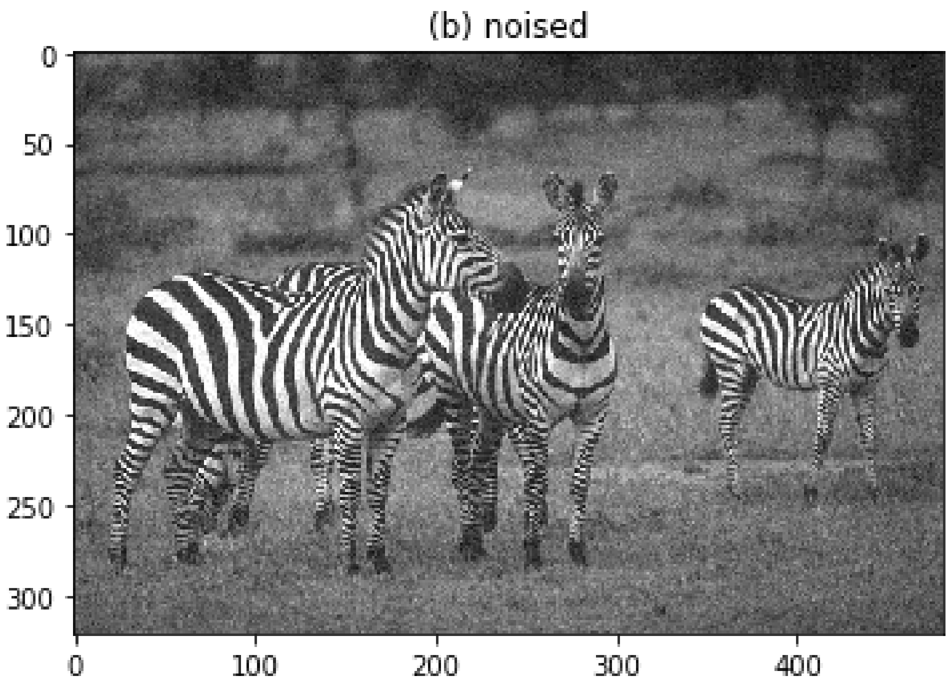
```
In [5]: import numpy as np
        from scipy.misc import imread, imsave
        import matplotlib.pyplot as plt
        import math
        import time

        def  noise(img, sigma0):
            n1,n2=img.shape
            img_test=img+sigma0*np.random.randn(n1,n2)
            for i in range (n1):
                for j in range (n2):
                    if img_test[i,j]>255:
                        img_test[i,j]=255
                    if img_test[i,j]<0:
                        img_test[i,j]=0
            return img_test

        def unfold(img, s1, s2):
            n1, n2 = img.shape
            dest = np.zeros((n1 + 2 * s1, n2 + 2 * s2), dtype=np.float64)
            dest[s1:n1 + s1, s2:n2 + s2] = img
            dest[:, 0:s2] = np.fliplr(dest[:, s2 + 1:2 * s2 + 1])
            dest[:, n2 + s2:n2 + 2 * s2] = np.fliplr(dest[:, n2:n2 + s2])
            dest[0:s1, :] = np.flipud(dest[s1 + 1:2 * s1 + 1, :])
            dest[n1 + s1:n1 + 2 * s1, :] = np.flipud(dest[n1:n1 + s1, :])
            return dest

        def imshift(x, k, l):
            n1, n2 = x.shape[0], x.shape[1]
            k, l = k % n2, l % n1
            xshifted = np.zeros(x.shape, dtype=np.float64)
            xshifted[l:n1, k:n2] = x[0:n1 - l, 0:n2 - k]
            xshifted[0:l, k:n2] = x[n1 - l:n1, 0:n2 - k]
            xshifted[0:l, 0:k] = x[n1 - l:n1, n2 - k:n2]
            xshifted[l:n1, 0:k] = x[0:n1 - l, n2 - k:n2]
            return xshifted
```

```
def imblocknlmeans_table(y, sigma, s1, s2, p1, p2, h):
    n1, n2 = y.shape
    patches = np.empty((n1, n2, 2 * p1 + 1, 2 * p2 + 1))
    mirror = unfold(y, p1, p2)
    for i in range(n1):
        #print("Build table:", i)
        for j in range(n2):
            patches[i, j] = mirror[i:i + 2 * p1 + 1, j:j + 2 * p2 + 1]
    p = (2 * p1 + 1) * (2 * p2 + 1)
    div = -16.0 * h * (sigma ** 2) / p
    boldx = np.zeros(patches.shape)
    z = np.zeros(patches.shape)
    for k in range(-s1, s1 + 1):
        #print("K =", k)
        for l in range(-s2, s2 + 1):
            shifted = imshift(patches, k, l)
            alpha = np.power(shifted - patches, 2) / p - (2 * sigma ** 2)
            alpha[alpha < 0] = 0
            phi = np.exp(alpha / div)
            boldx += phi * shifted
            z += phi
    boldx /= z
    result = np.zeros(y.shape)

    for a in range(-p1, p1 + 1):
        #print("A =", a)
        for b in range(-p2, p2 + 1):
            shifted = imshift(boldx, b, a)
            result += shifted[:, :, p1 + a, p2 + b]
    return result / p

def impsnr(x,y):
    n1,n2=x.shape
    n=(n1+1)*(n2+1)
    sum=0
    for i in range (n1):
        for j in range (n2):
            sum=sum+((x[i,j]-y[i,j])**2)/n
    psnr=10*math.log10((255**2)/sum)
    return psnr


sigma0=10
sigma=3
s1=s2=2
p1=p2=1
h=1
```

```
pic=imread('zebra.png')
plt.imshow(pic, cmap='gray')
plt.title('(a) original')
plt.show()

pic_noise=noise(pic, sigma0)
plt.imshow(pic_noise, cmap='gray')
plt.title('(b) noised')
#imsave('noised.png',pic_noise)
plt.show()
startime=time.time()
pic_BWNL_table=imblocknlmeans_table(pic_noise, sigma, s1, s2, p1,p2, h)
time_1=time.time()-startime
print('time=',time_1, 's')
plt.imshow(pic_BWNL_table, cmap='gray')
plt.title('(c) pic_BWNL_table')
imsave('pic_BWNL_table.png',pic_BWNL_table)
plt.show()
print('PNSR=',impsnr(pic,pic_BWNL_table))
```

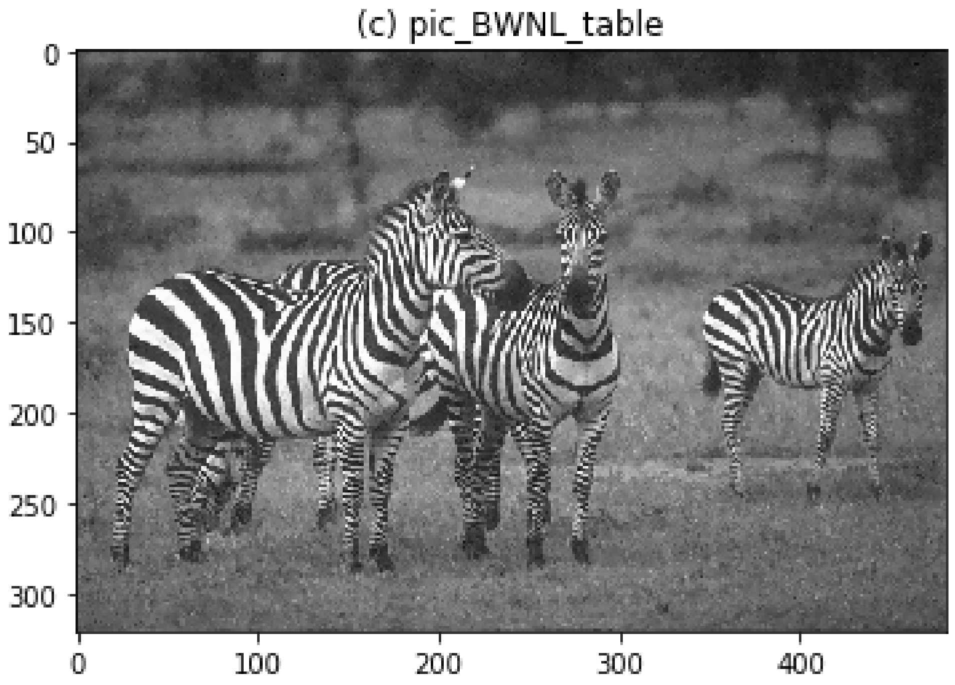
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:87: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```





```
time= 3.6968650817871094 s
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:103: DeprecationWarning: `imsave` is deprec
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
PNSR= 30.59108360898503
```

Phi_matrix_four loops imshift

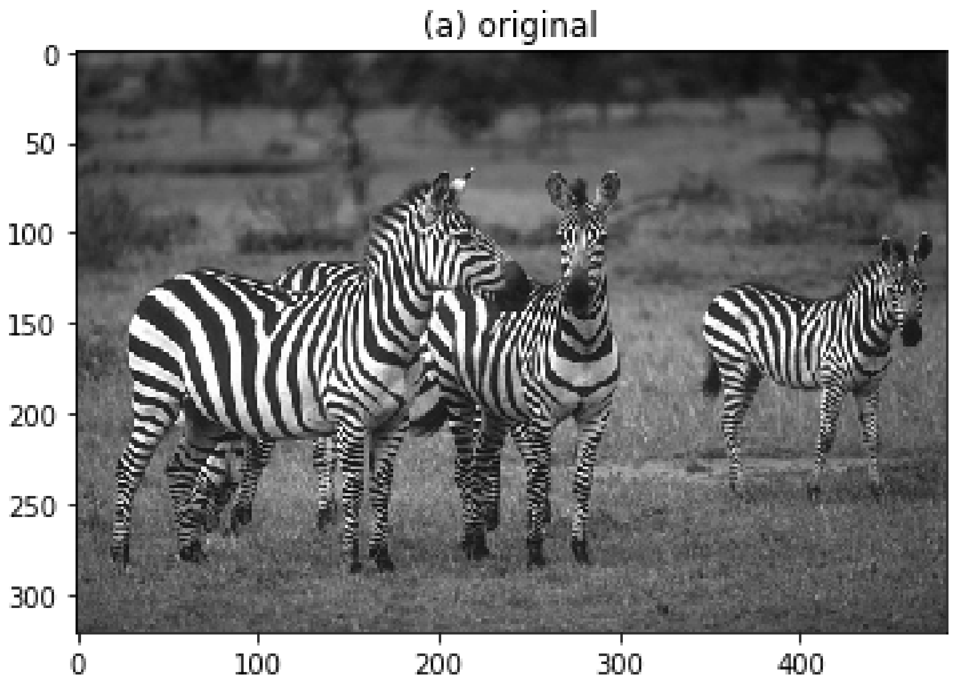
April 15, 2018

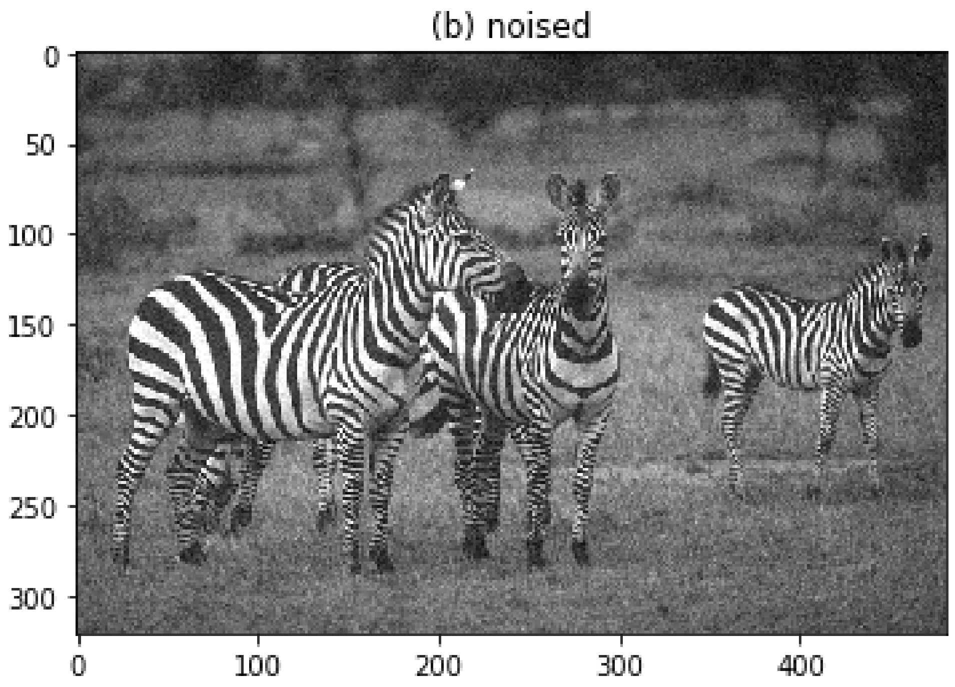
```
In [13]: import numpy as np
         from scipy.misc import imread, imsave
         import matplotlib.pyplot as plt
         import math
         import time

         def  noise(img, sigma0):
             n1,n2=img.shape
             img_test=img+sigma0*np.random.randn(n1,n2)
             for i in range (n1):
                 for j in range (n2):
                     if img_test[i,j]>255:
                         img_test[i,j]=255
                     if img_test[i,j]<0:
                         img_test[i,j]=0
             return img_test

         def imshift(x, k, l):
             n1, n2 = x.shape[0], x.shape[1]
             k, l = k % n2, l % n1
             xshifted = np.zeros(x.shape, dtype=np.float64)
             xshifted[l:n1, k:n2] = x[0:n1 - l, 0:n2 - k]
             xshifted[0:l, k:n2] = x[n1 - l:n1, 0:n2 - k]
             xshifted[0:l, 0:k] = x[n1 - l:n1, n2 - k:n2]
             xshifted[l:n1, 0:k] = x[0:n1 - l, n2 - k:n2]
             return xshifted


         def imBWNL(img, sigma, s1, s2, p1,p2, h):
             n1,n2=img.shape
             p=(2*p1+1)*(2*p2+1)
             x_BWNL=temp2=np.zeros((n1,n2))
             img_patch=np.zeros((n1,n2,2*p1+1,2*p2+1))
             table=np.zeros((2*s1+1,2*s2+1,n1,n2))
             for k in range (-s1,s1+1):
                 for l in range (-s2,s2+1):
                     table[k+s1,l+s2]=imshift(img,l,k)
```

```
    dominator=((-16*h*sigma**2)/p)
    for i in range (s1,n1-s1):
        for j in range (s2,n2-s2):
            temp1=img[i-p1:i+p1+1,j-p2:j+p2+1]
            sum=np.zeros((2*p1+1,2*p2+1))
            z=np.zeros((2*p1+1,2*p2+1))
            for k in range (-s1,s1+1):
                for l in range (-s2,s2+1):
                    temp0=table[k+s1,l+s2][i-p1:i+p1+1,j-p2:j+p2+1]
                    alpha=np.power(temp0-temp1,2)
                    alpha=alpha/p
                    numerator=alpha-2*sigma**2
                    numerator[numerator<0]=0
                    phi=np.exp(numerator/dominator)
                    temp=phi*temp0
                    z=z+phi
                    sum=sum+temp
                    img_patch[i-k,j-l]=sum/z
    for i in range (p1,n1-p1):
        #print(i)
        for j in range (p2,n2-p2):
            for u in range (-p1,p1+1):
                for v in range (-p2,p2+1):
                    x_BWNL[i,j]=x_BWNL[i,j]+img_patch[i-u,j-v,p1+u,p2+v]
    return imshift(x_BWNL/p,s2,s1)


#peak-signal-to-noise r
def impsnr(x,y,p1,p2):
    n1,n2=x.shape
    n=n1*n2
    sum=0
    for i in range (p1,n1-p1):
        for j in range (p2,n2-p2):
            sum=sum+((x[i,j]-y[i,j])**2)/n
    psnr=10*math.log10((255**2)/sum)
    return psnr

sigma0=10
sigma=3
s1=s2=2
p1=p2=1
h=1

pic=imread('zebra.png')
plt.imshow(pic, cmap='gray')
plt.title('(a) original')
plt.show()
```

```
        pic_noise=noise(pic, sigma0)
        plt.imshow(pic_noise, cmap='gray')
        plt.title('(b) noised')
        plt.show()
        startime=time.time()
        pic_BWNL=imBWNL(pic_noise, sigma, s1, s2, p1,p2, h)
        time_1=time.time()-startime
        print('time=',time_1, 's')
        plt.imshow(pic_BWNL, cmap='gray')
        plt.title('(d) pic_BWNL')
        imsave('pic_BWNL.png',pic_BWNL)
        plt.show()
        print('PNSR=',impsnr(pic,pic_BWNL,10,10))
```

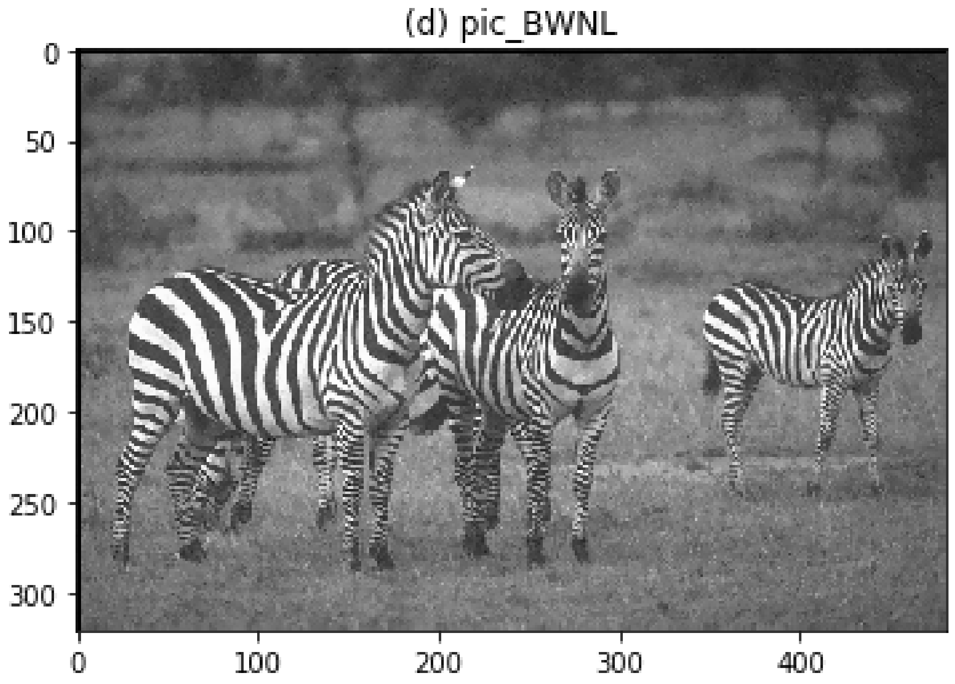
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:82: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```





```
time= 67.08843922615051 s
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:97: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
PNSR= 31.00495650976352
```

Phi_matrix_slicing

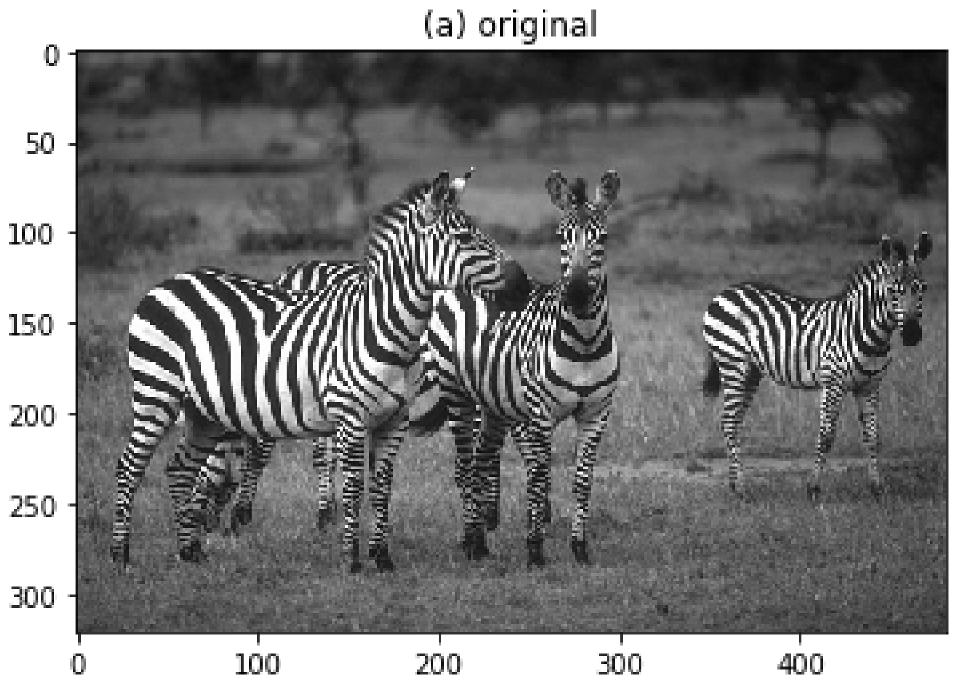
April 15, 2018

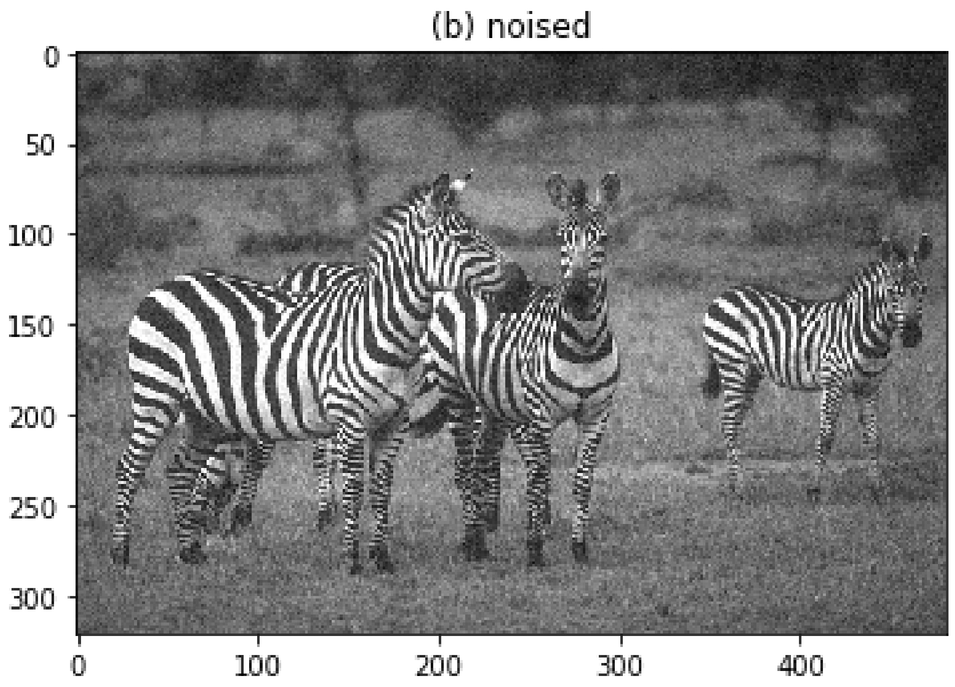
```
In [13]: import numpy as np
         from scipy.misc import imread, imsave
         import matplotlib.pyplot as plt
         import math
         import time

         def  noise(img, sigma0):
             n1,n2=img.shape
             img_test=img+sigma0*np.random.randn(n1,n2)
             for i in range (n1):
                 for j in range (n2):
                     if img_test[i,j]>255:
                         img_test[i,j]=255
                     if img_test[i,j]<0:
                         img_test[i,j]=0
             return img_test

         def unfold(img, s1, s2):
             n1, n2 = img.shape
             dest = np.zeros((n1 + 2 * s1, n2 + 2 * s2), dtype=np.float64)
             dest[s1:n1 + s1, s2:n2 + s2] = img
             dest[:, 0:s2] = np.fliplr(dest[:, s2 + 1:2 * s2 + 1])
             dest[:, n2 + s2:n2 + 2 * s2] = np.fliplr(dest[:, n2:n2 + s2])
             dest[0:s1, :] = np.flipud(dest[s1 + 1:2 * s1 + 1, :])
             dest[n1 + s1:n1 + 2 * s1, :] = np.flipud(dest[n1:n1 + s1, :])
             return dest

         def imshift(x, k, l):
             n1, n2 = x.shape[0], x.shape[1]
             k, l = k % n2, l % n1
             xshifted = np.zeros(x.shape, dtype=np.float64)
             xshifted[l:n1, k:n2] = x[0:n1 - l, 0:n2 - k]
             xshifted[0:l, k:n2] = x[n1 - l:n1, 0:n2 - k]
             xshifted[0:l, 0:k] = x[n1 - l:n1, n2 - k:n2]
             xshifted[l:n1, 0:k] = x[0:n1 - l, n2 - k:n2]
             return xshifted
```

```
def imblocknlmeans_slicing(y, sigma, s1, s2, p1, p2, h):
    n1, n2 = y.shape
    p = (2 * p1 + 1) * (2 * p2 + 1)
    sub = 2 * sigma ** 2
    div = -16.0 * h * (sigma ** 2) / p
    mirror = unfold(y, s1 + p1, s2 + p2)
    boldx = np.empty((n1, n2, 2 * p1 + 1, 2 * p2 + 1))
    for i in range(n1):
        #print("In line:", i)
        for j in range(n2):
            patch = mirror[i + s1:i + s1 + 2 * p1 + 1, j + s2:j + s2 + 2 * p2 + 1]
            temp = np.zeros((2 * p1 + 1, 2 * p2 + 1))
            z = np.zeros((2 * p1 + 1, 2 * p2 + 1))
            z = 0
            for k in range(-s1, s1 + 1):
                for l in range(-s2, s2 + 1):
                    shifted = mirror[i + s1 + k:i + s1 + 2 * p1 + 1 + k, j + s2 + l:j
                    alpha = np.mean(np.power(shifted - patch, 2))
                    phi = math.exp(max(alpha - sub, 0) / div)
                    alpha = np.power(shifted - patch, 2) / p - sub
                    alpha[alpha < 0] = 0
                    phi = np.exp(alpha / div)
                    z += phi
                    temp += shifted * phi
            boldx[i, j] = temp / z
    result = np.zeros(y.shape)
    for a in range(-p1, p1 + 1):
        #print("A =", a)
        for b in range(-p2, p2 + 1):
            shifted = imshift(boldx, b, a)
            result += shifted[:, :, p1 + a, p2 + b]
    return result / p

def impsnr(x,y):
    n1,n2=x.shape
    n=(n1+1)*(n2+1)
    sum=0
    for i in range (n1):
        for j in range (n2):
            sum=sum+((x[i,j]-y[i,j])**2)/n
    psnr=10*math.log10((255**2)/sum)
    return psnr

sigma0=10
sigma=3
s1=s2=2
p1=p2=1
h=1
```

```
pic=imread('zebra.png')
plt.imshow(pic, cmap='gray')
plt.title('(a) original')
plt.show()

pic_noise=noise(pic, sigma0)
plt.imshow(pic_noise, cmap='gray')
plt.title('(b) noised')
#imsave('noised.png',pic_noise)
plt.show()
startime=time.time()
pic_BWNL_slicing=imblocknlmeans_slicing(pic_noise, sigma, s1, s2, p1,p2, h)
time_1=time.time()-startime
print('time=',time_1, 's')
plt.imshow(pic_BWNL_slicing, cmap='gray')
plt.title('(c) pic_BWNL_slicing')
imsave('pic_BWNL.png',pic_BWNL_slicing)
plt.show()
print('PNSR=',impsnr(pic,pic_BWNL_slicing))
```

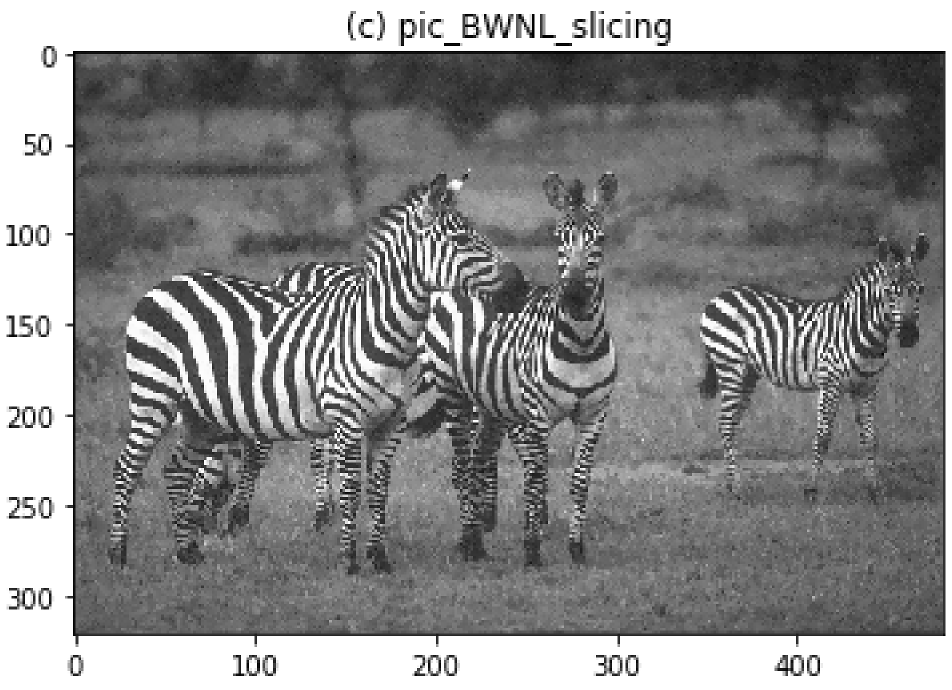
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:87: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```





```
time= 131.4691035747528 s
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:103: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
PNSR= 30.58966698303419
```

hw6

April 18, 2018

1 Computer Vision

1.1 Assigment 6

1.2 April 15, 2018

This assignment contains 1 programming exercises with 2 sections.

1.3 Problem 1: Hough Transform

This problem we will introduce Hough Transform. The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

(i) Implement the Hough Transform (HT) using the (phi, theta) parameterization as described in GW Third Edition p. 733-738 (please see 'HoughTransform.pdf' provided in the folder). Use accumulator cells with a resolution of 1 degree in theta and 1 pixel in phi.

(ii) Produce a simple 11 x 11 test image made up of zeros with 5 ones in it, arranged like the 5 points in GW Third Edition Figure 10.33(a).

Compute and display its Hough Transform; the result should look like GW Third Edition Figure 10.33(b). Threshold the HT by looking for any (phi, theta) cells that contains more than 2 votes then plot the corresponding lines in (x,y)-space on top of the original image.

(iii) Load in the image 'lane.png'.

Compute and display its edges using the edge detector, you can use canny edge detector, which you have implemented in Problem 1, or use OpenCV edge detection operator, such as Sobel, etc.

Now compute and display the Hough Transform of the binary edge image. As before, threshold the HT and plot the corresponding lines atop the original image; this time, use a threshold of 75% maximum accumulator count over the entire HT, i.e. 0.75*max(HT(:)).

(iv) We would like to only show line detections in the driver's lane and ignore any other line detections such as the lines resulting from the neighboring lane closest to the bus, light pole, and sidewalks. Using the thresholded HT from the 'lanes.png' image in the previous part, show only the lines corresponding to the line detections from the driver's lane by thresholding the HT again using a specified range of theta this time. What are the approximate theta values for the two lines in the driver's lane?

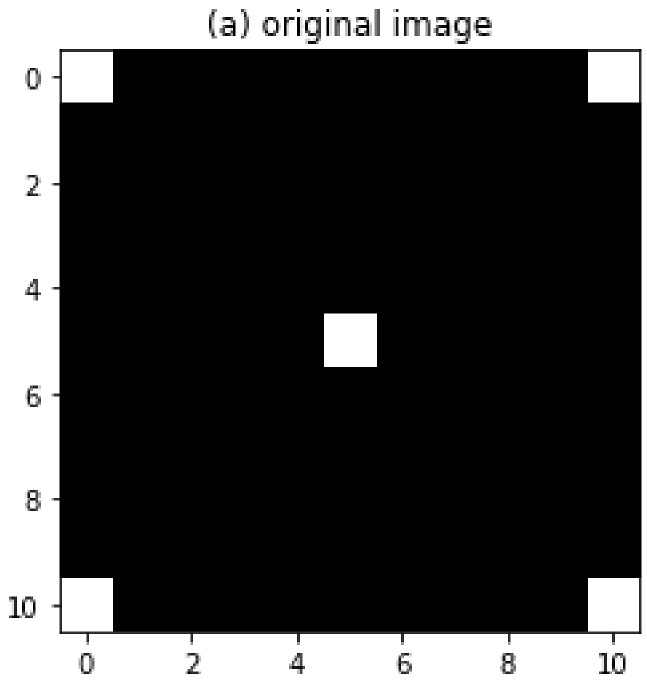
Things to turn in:
Hough Transform plot should have colorbars next to them
Line overlays should be clearly visible (adjust line width if needed)
HT image axes should be properly labeled with name and values (see Figure 10.33(b) in the HoughTransform PDF for example)
3 images from 2(ii): original image, Hough Transform plot, original image with detected lines
4 images from 2(iii): original image, binary edge image, Hough Transform plot, original image with detected lines
1 image from 2(iv): original image with detected lines
theta values from 2(iv)

```
In [2]: import numpy as np
        from scipy.misc import *
        import matplotlib.pyplot as plt
        import math
        from scipy import signal

        def Hough_transform(img):
            n1,n2=img.shape
            D=round(math.sqrt(n1**2+n2**2))
            A=np.zeros((2*D+1,181),dtype=np.uint8)
            for i in range (n1):
                for j in range (n2):
                    if img[i,j]!=0:
                        for theta in range (-90,91):
                            rho=round(i*math.cos((theta*math.pi)/180)+j*math.sin((theta*math.p
                            A[rho+D,theta+90]+=1
            return A

        def thresholding_naive(img, HT_img):
            s1,s2=HT_img.shape
            n1,n2=img.shape
            D=round(math.sqrt(n1**2+n2**2))
            xy_plane=np.zeros((n1,n2),dtype=np.uint8)
            threshold=0.75*np.max(HT_img)
            for i in range (s1):
                for j in range (s2):
                    if HT_img[i,j]>threshold:
                        rho=i-D
                        theta=j-90
                        for y in range (n1):
```

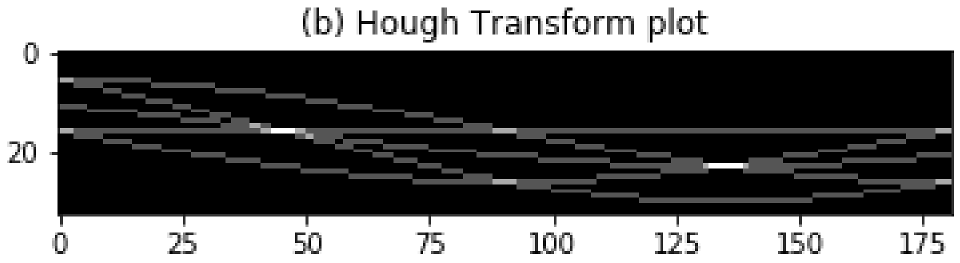
```
                    for x in range(n2):
                        a=math.ceil(x*math.sin((theta*math.pi)/180)+y*math.cos((theta*m
                        b=a-1
                        if a==rho or b==rho:
                            xy_plane[y,x]=255
    return xy_plane


def thresholding(img, HT_img):
    s1,s2=HT_img.shape
    n1,n2=img.shape
    D=round(math.sqrt(n1**2+n2**2))
    xy_plane=np.zeros((n1,n2),dtype=np.uint8)
    threshold=0.75*np.max(HT_img)
    for j in range (s2):
        for i in range (s1):
            if HT_img[i,j]>threshold:
                rho=i-D
                theta=j-90
                if -45<theta<-30 or 30<theta<45:
                    for y in range (n1):
                        for x in range(n2):
                            a=math.ceil(x*math.sin((theta*math.pi)/180)+y*math.cos((the
                            b=a-1
                            if a==rho or b==rho:
                                xy_plane[y,x]=255
    return xy_plane

#canny process:::::

def rgb2gray(rgb):
    grayimg = np.mean(rgb, axis=2)
    return grayimg.astype(np.uint8)

def canny_Smooth(img):
    kernel = np.array([[2, 4, 5, 4, 2], [4, 9, 12, 9, 4], [5, 12, 15, 12, 5], [4, 9, 1
    return signal.convolve2d(img, kernel, boundary="symm")
def canny_Gradients(img):
    xKernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    yKernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    Gx = signal.convolve2d(img, xKernel, boundary="symm")
    Gy = signal.convolve2d(img, yKernel, boundary="symm")
    G = np.sqrt(np.power(Gx, 2) + np.power(Gy, 2))
    Gtheta, gmax = np.arctan2(Gx, Gy), G.max()
    return G * 255 / gmax, Gtheta
def canny_NMS(G, Gtheta):
    table = np.array([[1, 0], [1, 1], [0, 1], [-1, 1], [-1, 0], [-1, -1], [0, -1], [1,
    Gtheta[Gtheta < 0] = Gtheta[Gtheta < 0] + 2 * math.pi
    gNMS = np.zeros(G.shape, dtype=np.uint8)
    for i in range(1, G.shape[0] - 1):
        #print("In line:", i)
        for j in range(1, G.shape[1] - 1):
            key = int(round(Gtheta[i, j] * 4 / math.pi)) % 8
            if math.fabs(G[i, j] - max(G[i, j], G[i + table[key, 0], j + table[key, 1]
                                      G[i - table[key, 0], j - table[key, 1]])) < 1e-
                gNMS[i, j] = G[i, j]
    return gNMS
def canny_Threshold(img, te):
    img[img < te] = 0
    img[img >= te] = 255
    return img

#Canny Edge Detection Function
def canny_edge(img, te):
    img_smooth = canny_Smooth(img)
    #imsave("geisel_smooth.jpg", img_smooth)
    #plt.subplot(2, 2, 1)
    #plt.imshow(img_smooth, cmap="gray")
    G, Gt = canny_Gradients(img_smooth)
    #imsave("geisel_G.jpg", G)
    #plt.subplot(2, 2, 2)
    #plt.imshow(G, cmap="gray")
    img_NMS = canny_NMS(G, Gt)
    #imsave("geisel_NMS.jpg", img_NMS)
    #plt.subplot(2, 2, 3)
    #plt.imshow(img_NMS, cmap="gray")
    detected_img = canny_Threshold(img_NMS, te)
    imsave("canny.png", detected_img)
    #plt.subplot(2, 2, 4)
    plt.imshow(detected_img, cmap="gray")
    return detected_img



#question(2)
print('question(2):')
pic_test=np.zeros((11,11))
pic_test[0,0]=pic_test[0,10]=pic_test[10,0]=pic_test[10,10]=pic_test[5,5]=255
plt.imshow(pic_test, cmap='gray')
plt.title('(a) original image')
imsave('pic_test.png',pic_test)
plt.show()
HT_test=Hough_transform(pic_test)
plt.imshow(HT_test, cmap='gray')
plt.title('(b) Hough Transform plot')
imsave('HT_test.png',HT_test)
```

```
    plt.show()
    threshold_test=thresholding(pic_test, HT_test)
    plt.imshow(threshold_test, cmap='gray')
    plt.title('(c) original image with detected lines')
    imsave('threshold_test.png',threshold_test)
    plt.show()

    #question(3)
    print('question(3):')
    img=imread("lane.png")
    plt.imshow(img, cmap='gray')
    plt.title('(a) original image')
    plt.show()

    img_gray = rgb2gray(img)
    plt.imshow(img_gray, cmap="gray")
    canny_edge(img_gray, 40)
    plt.title('(b) binary edge image')
    plt.show()

    pic_canny=canny_edge(img_gray, 40)
    HT_canny_naive=Hough_transform(pic_canny)
    plt.imshow(HT_canny_naive, cmap='gray')
    plt.title('(c) Hough Transform plot')
    imsave('HT_canny_naive.png',HT_canny_naive)
    plt.show()
    threshold_canny_naive=thresholding_naive(pic_canny, HT_canny_naive)
    plt.imshow(threshold_canny_naive, cmap='gray')
    plt.title('(d) original image with detected lines')
    imsave('threshold_canny_naive.png',threshold_canny_naive)
    plt.show()

    #question(4)
    print('question(4):')
    HT_canny=Hough_transform(pic_canny)
    threshold_canny=thresholding(pic_canny, HT_canny)
    plt.imshow(threshold_canny, cmap='gray')
    plt.title('original image with detected lines')
    imsave('threshold_canny.png',threshold_canny)
    plt.show()
```

```
question(2):


C:\Anaconda\lib\site-packages\ipykernel_launcher.py:120: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

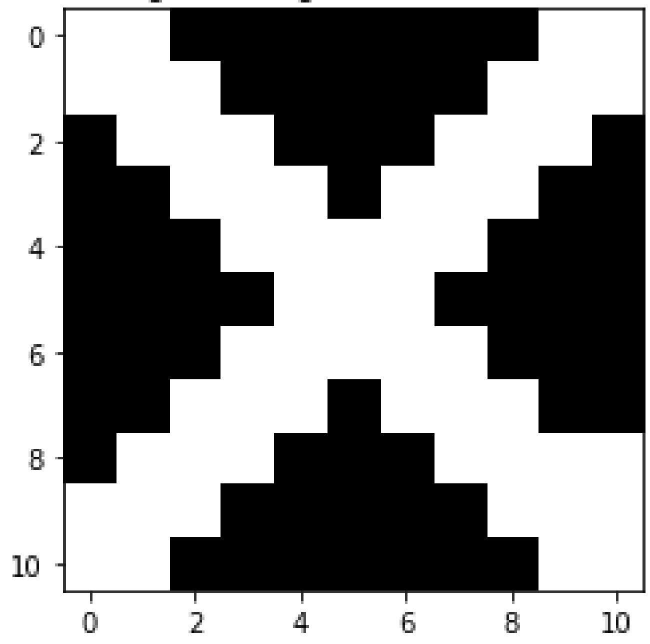


```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:125: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:130: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

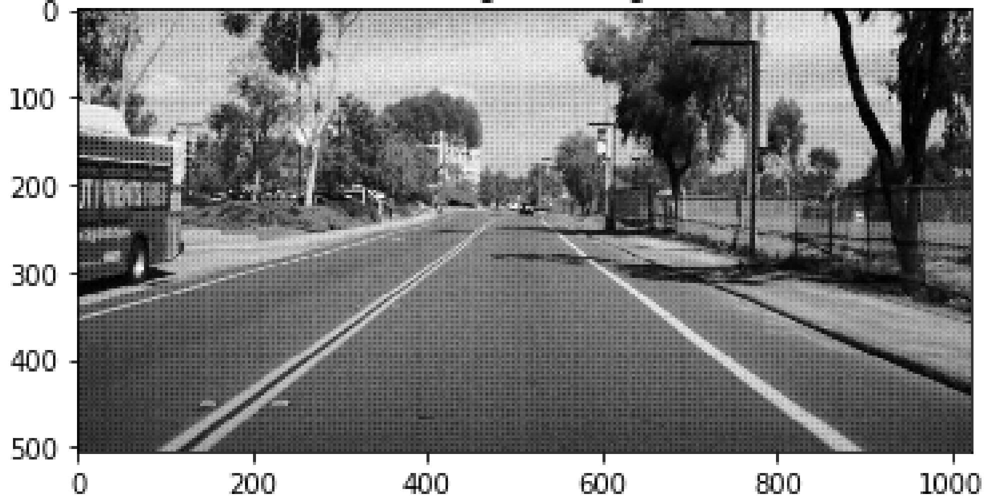
(c) original image with detected lines



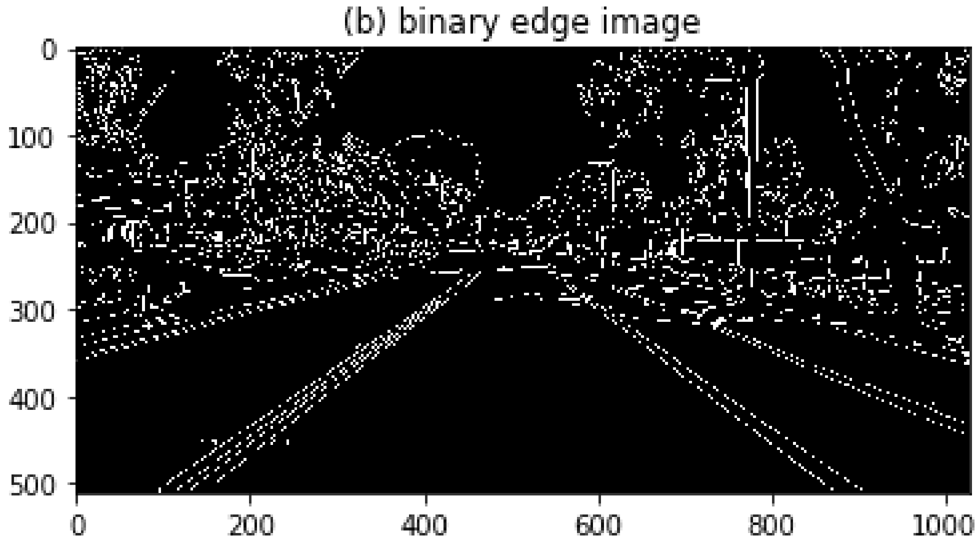
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:135: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

question(3):

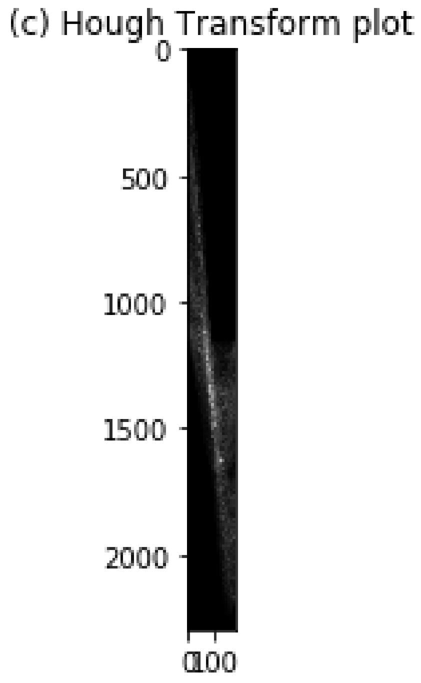
(a) original image



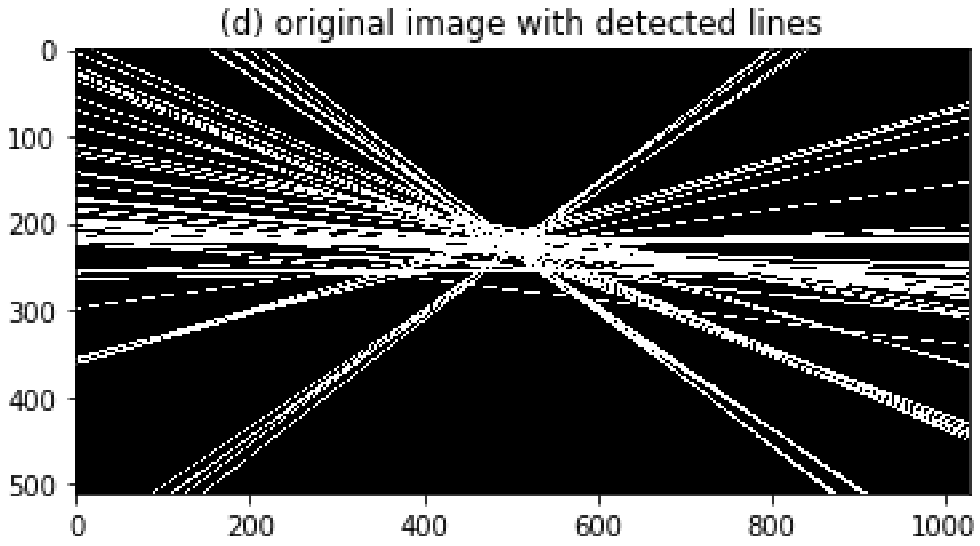
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:107: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:150: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

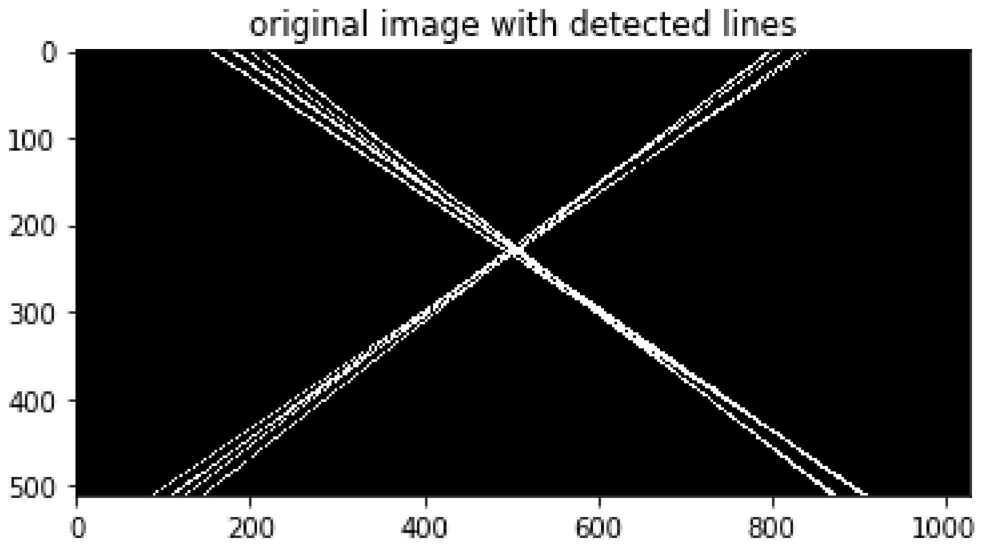


```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:155: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



question(4):

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:164: DeprecationWarning: `imsave` is depre
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```



** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File $\rightarrow$ Download as $\rightarrow$ PDF via LaTeX

hw7

April 26, 2018

1 Computer Vision

1.1 Assigment 7

1.2 April 22, 2018

This assignment contains 2 programming exercises.

1.3 Problem 1: Template Matching

Template matching is a naive technique for object detection and recognition in an image. It makes use of normalized cross correlation (NCC), which is a very similar operation to convolution. The differences between NCC and convolution are subtle3; you can think of it as a modified convolution. Follow the four steps to perform template matching on two different images:

(i) Correlation by Convolution: Using the Letter.jpg and LetterTemplate.jpg perform the following steps to find the template in the original image.

Convert both image and template to grayscale, double type.
Flip LetterTemplate.jpg vertically and horizontally.
Convolve the Letter.jpg and LetterTemplate.jpg.
Find the maximum point of the resulting convolved image. This is the location of the template match in the original image.

Draw a rectangle (using matplotlib.patches.rectangle() or cv2.rectangle()) over the original image centered at the location where the template match occurred. The size of this rectangle is the same as the template.

(ii) Normalized Cross Correlation: try to use functions like skimage.feature.match_template(), or scipy.signal.correlate2d() to performs a normalized cross correlation. It returns a correlation matrix that can be interpreted as an image. The maximum value of this image represents the location of the largest correlation between the image and the template. Repeat the process in (i) using Normalized Cross Correlation instead of convolution, and compare their result.

(iii) Multiple Matches: Now use Cross Correlation to detect faces in crowd.jpg using face1.jpeg as a template. This time instead of finding the maximum value in the correlation matrix, use a percentage of the maximum as the threshold to detect multiple faces instead of just the best match. Draw rectangles over the original image to indicate where the template matches occurred.

(iv) Multiple Templates: Repeat part (iii), but this time use all three templates, face1.jpeg, face2.jpeg and face3.jpeg, to generate three separate correlation matrices. Use each of the three to detect multiple face orientations in the crowd. Draw rectangles over the original image to indicate where the template matches occur (for all three templates). Make sure you use rectangles of different colors to represent matches corresponding to different templates.

Things to turn in:

Part (i): image after convolution (use imagesc()) and original image with rectangle depicting template match.

Part (ii): image after Cross Correlation (use imagesc()) and original image with rectangle depicting template match.

Part (iii): image after Cross Correlation (use imagesc()) and original image with rectangles depicting template matches. Also mention NCC threshold used to detect faces.

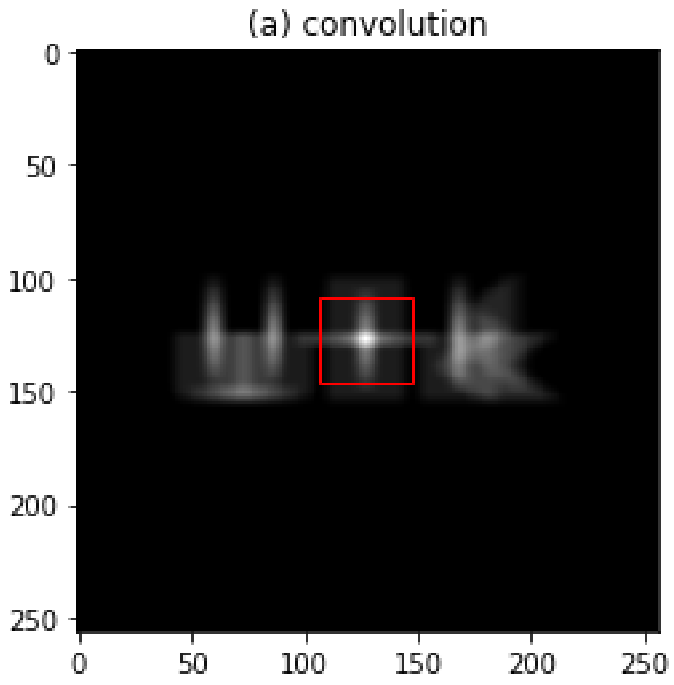
Part (iv): images after Cross Correlation (use imagesc()) and original image with rectangles depicting template matches (color coded for each template).

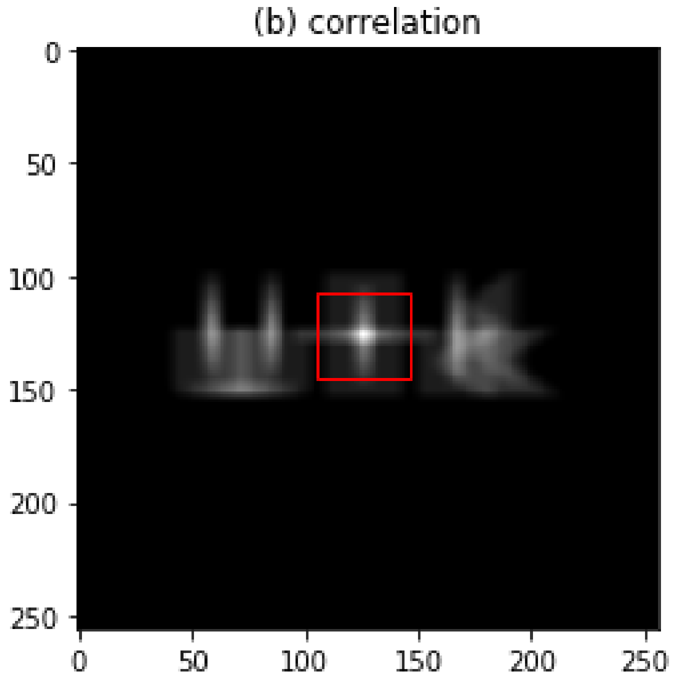
```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.misc import *
        import math
        import matplotlib.patches as mpatches
        from scipy import signal


        def rgb(img):
            gray=np.sum(img,axis=2)/3
            gray=gray.astype(np.float64)
            return gray

        def imshift(x, p, q): #P,q
            p=-p
            n1=x.shape[1]#columns
            n2=x.shape[0]#rows
            k=p%n2
            l=q%n1
            xshifted=np.zeros([n2,n1])
            xshifted[n2-k:n2,0:l]=x[0:k,n1-l:n1]
            xshifted[0:n2-k,l:n1]=x[k:n2,0:n1-l]
            xshifted[0:n2-k,0:l]=x[k:n2,n1-l:n1]
            xshifted[n2-k:n2,l:n1]=x[0:k,0:n1-l]
            return xshifted

        def pading(img, tpl):
            s1,s2=tpl.shape
            s1=int((s1)/2)
            s2=int((s2)/2)
            (n1,n2)=img.shape
            img_pad=np.zeros((n1+2*s1,n2+2*s2))
```

```
    img_pad[s1:s1+n1, s2:s2+n2]=img[:,:]
    for i in range (0,s1-1):
        img_pad[i,s2:s2+n2]=img[s1-i-1,:]
    for i in range (s1+n1+1,2*s1+n1):
        img_pad[i,s2:s2+n2]=img[n1-1-i, :]
    for j in range (0,s2-1):
        img_pad[0:2*s1+n1,j]=img_pad[0:2*s1+n1,2*s2-j-1]
    for j in range (s1+n1+1,2*s2+n2):
        img_pad[0:2*s1+n1,j]=img_pad[0:2*s1+n1,2*s2+n2-j]
    return img_pad

#question 1

def Convolution(img, tpl):
    n1,n2=img.shape
    p1,p2=tpl.shape
    s1,s2=(int(p1//2),int(p2//2))
    conv=np.zeros(img.shape, dtype=np.float64)
    for i in range (n1-p1):
        for j in range (n2-p2):
            temp=img[i:i+p1,j:j+p2]
            conv[i+s1,j+s2]=(np.sum(temp*tpl))
    return conv

def draw_conv(img,tpl):
    conv=Convolution(img, tpl)
    s1,s2=tpl.shape
    location=np.where(conv==np.max(conv))
    x=location[0][0]
    y=location[1][0]
    [i,j]=np.array((x,y))-np.array((int((s1)/2),int((s2)/2)))
    rect = mpatches.Rectangle([j,i], s2, s1, color="r", fill=False)
    plt.subplot().add_patch(rect)
    return rect


#question(2)
def Correlation(img, tpl):
    return signal.correlate2d(img, tpl,mode="same", boundary="fill")


def draw_corr(img,tpl):
    corr=Correlation(img, tpl)
    s1,s2=tpl.shape
    location=np.where(corr==np.max(corr))
    x=location[0][0]
    y=location[1][0]
    [i,j]=np.array((x,y))-np.array((int((s1)/2),int((s2)/2)))
    rect = mpatches.Rectangle([j,i], s2, s1, color="r", fill=False)
    plt.subplot().add_patch(rect)
    return rect

#question(3)
def NCC(img,tpl):
    n1,n2=img.shape
    t1,t2=tpl.shape
    t=t1*t2
    s1,s2=int(t1//2),int(t2//2)
    t_mean=np.mean(tpl)
    t_sum=np.sum(tpl)
    result=np.zeros(img.shape,dtype=np.float64)
    diff_2=tpl-t_mean
    for i in range (n1-t1):
        for j in range (n2-t2):
            f_mean=(np.sum(img[i:i+t1,j:j+t2]))/t
            diff_1=img[i:i+t1,j:j+t2]-f_mean
            numerator=np.sum(diff_1*diff_2)
            diff_one=np.power(diff_1,2)
            sum1=np.sum(diff_one)
            sum2=np.sum(np.power(diff_2,2))
            dominator=math.sqrt(sum1*sum2)
            result[i+s1,j+s2]=numerator/dominator
    return result

def draw_NCC(img,tpl,color,threshold):
    haha=NCC(img,tpl)
    s1,s2=tpl.shape
    location=np.array(np.where(haha>=threshold*haha.max()), dtype=np.float64)
    for p in range (location.shape[1]):
        x=location[0][p]
        y=location[1][p]
        [i,j]=np.array((x,y))-np.array((int((s1)/2),int((s2)/2)))
        rect = mpatches.Rectangle([j,i], s2, s1, color=color, fill=False)
    plt.subplot().add_patch(rect)
    return rect

pic=imread('Letters.jpg').astype(np.float64)
template=imread('LettersTemplate.jpg').astype(np.float64)
conv=Convolution(pic,template)
plt.imshow(conv,cmap='gray')
draw_conv=draw_conv(pic,template)
plt.title('(a) convolution')
plt.show()

corr=Correlation(pic,template)
plt.imshow(corr,cmap='gray')
```

```
        draw_corr=draw_corr(pic,template)
        plt.title('(b) correlation')
        plt.show()


        pic_2=imread('crowd.jpg').astype(np.float64)
        template_1=imread('face1.jpeg').astype(np.float64)
        template_2=imread('face2.jpeg').astype(np.float64)
        template_3=imread('face3.jpeg').astype(np.float64)
        pic_gray=rgb(pic_2).astype(np.float64)
        template_1_gray=rgb(template_1).astype(np.float64)
        template_2_gray=rgb(template_2).astype(np.float64)
        template_3_gray=rgb(template_3).astype(np.float64)
        plt.imshow(pic_gray,cmap='gray')
        plt.title('(c) detect_face')
        draw_NCC(pic_gray,template_1_gray,'r', 0.8)
        draw_NCC(pic_gray,template_2_gray,'g', 0.8)
        detect_face=draw_NCC(pic_gray,template_3_gray,'y', 0.8)
        plt.show()
```

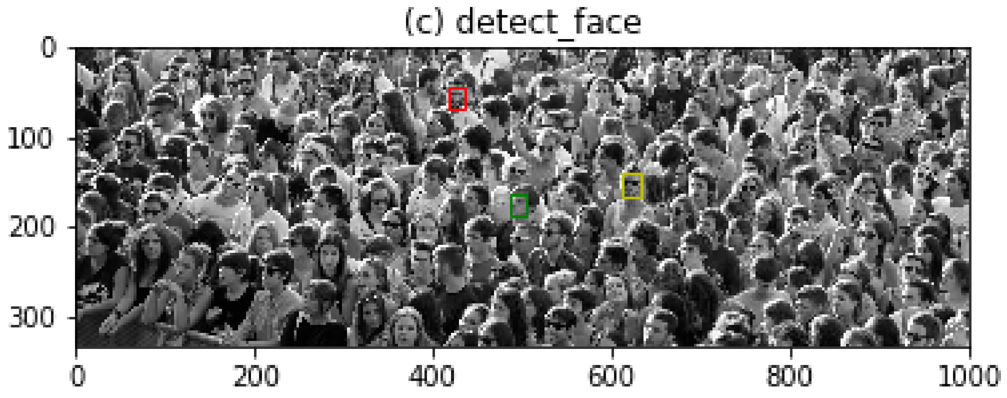
```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:119: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:120: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecationWarnin
  warnings.warn(message, mplDeprecation, stacklevel=1)
```





```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:134: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
```

```
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:135: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:136: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:137: DeprecationWarning: `imread` is depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



1.4 Problem 2 K-Means Segmentation

In this problem, we shall implement a K-Means based segmentation algorithm from scratch. To do this, you are required to implement the following three functions -

def createDataset(im): This function takes in an RGB image as input, and returns a dataset of features which are to be clustered. The function returns features, which is an N Œ M matrix where N is the number of pixels in the image im, and M = 3 (to store the RGB value of each pixel).

def kMeansCluster(features, centers): This function is intended to perform K-Means based clustering on the dataset features (of size N Œ M). The function returns a list [idx, centers]. Each row in features represents a data point, and each column represents a feature. centers is a k Œ M matrix, where each row is the initial value of a cluster center. The output idx is an N Œ 1 vector that stores the final cluster membership (1, 2, ů ů ů , k) of each data point. The output centers are the final cluster centers after K-Means. Note that you may need to set a maximum iteration count to exit K-Means in case the algorithm fails to converge. You may use loops in this function.

def mapValues(im, idx): This function takes in the cluster membership vector idx (N Œ 1), and returns the segmented image im_seg as the output. Each pixel in the segmented image must have the RGB value of the cluster center to which it belongs. You may use loops for this part.

With the above functions set up, perform image segmentation on the image white-tower.png, with the number of clusters, nclusters = 7. To maintain uniformity in the output image, please initialize clusters centers for K-Means as follows -

```
np.random.seed(5)

nclusters = 7

features = createDataset(im)

id = np.random.randint(np.shape(features)[0], size=(1, nclusters))
```

Cluster Center points will be the corresponding row of each id value in features.
Things to turn in:
The input image, and the image after segmentation.
The final cluster centers that you obtain after K-Means.

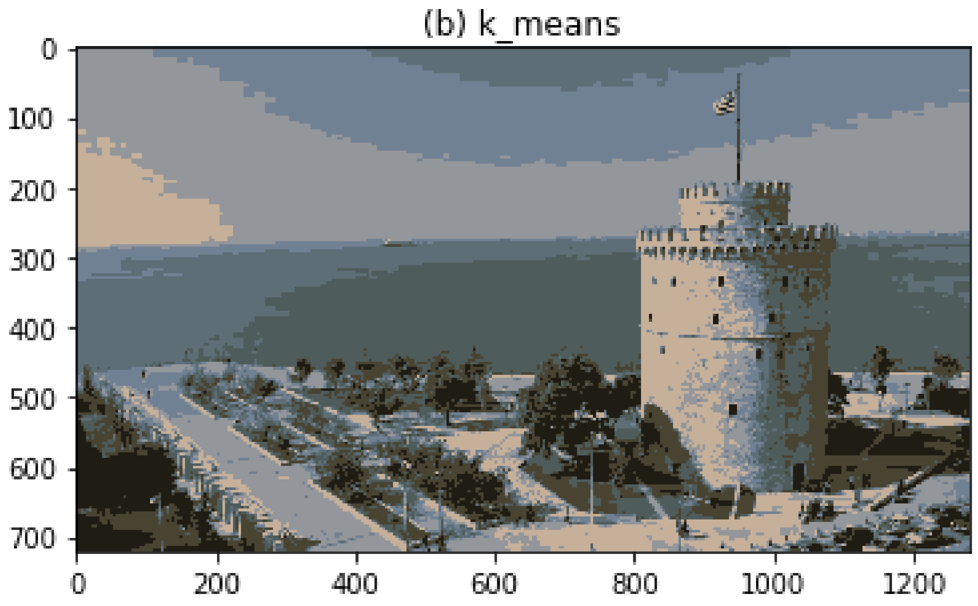
```
In [2]: #K mean segmentation Function
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.patches as mpatches
        import math
        from scipy.misc import *
        from scipy import signal


        def createDataset(im):
            img=im.reshape((im.shape[0]*im.shape[1], im.shape[2]))
            return img


        def kMeansCluster(features, centers):
            number=0
            while number<18:
                print('interation=',number)
                idx=[]
                for i in range (features.shape[0]):
                    intens_features=features[i][0]**2+features[i][1]**2+features[i][2]**2
                    dis_=[]
                    for j in range (centers.shape[0]):
                        intens_centers=centers[j][0]**2+centers[j][1]**2+centers[j][2]**2
                        distance=(intens_features-intens_centers)**2
                        dis_.append(distance)
                    category=dis_.index(min(dis_))
                    idx.append(category)
                center=np.zeros(centers.shape)
                count=np.zeros(centers.shape)
                for i in range (len(idx)):
                    center[idx[i]]=center[idx[i]]+features[i]
                    count[idx[i]]=count[idx[i]]+np.array([1,1,1])
                centers=center/count
                number=number+1
            return [idx, centers]

        def mapValues(img,centers,index,features):
            temp=np.zeros(features.shape)
            for i in range (features.shape[0]):
                temp[i]=centers[index[i]]
            img_seg=temp.reshape((img.shape[0],img.shape[1],3))
            return img_seg


        pic=imread('white-tower.png')
        plt.imshow(pic)
        plt.title('(a) original')
        plt.show

        np.random.seed(5)
        nclusters = 7
        features = createDataset(pic)
        id = np.random.randint(np.shape(features)[0], size=(1, nclusters))
        centers=features[id].reshape((nclusters,3))

        result=kMeansCluster(features, centers)
        centers_=result[1]
        index=result[0]
        im_seg=mapValues(pic,centers_,index,features)
        img_seg=im_seg.astype(np.uint8)
        plt.imshow(img_seg)
        imsave('k_means.png',img_seg)
        plt.title('(b) k_means')
        plt.show
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:46: DeprecationWarning: `imread` is depreca
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.


interation= 0
interation= 1
interation= 2
interation= 3
interation= 4
interation= 5
interation= 6
interation= 7
interation= 8
interation= 9
interation= 10
interation= 11
```

```
interation= 12
interation= 13
interation= 14
interation= 15
interation= 16
interation= 17
```

```
C:\Anaconda\lib\site-packages\ipykernel_launcher.py:63: DeprecationWarning: `imsave` is depreca
`imsave` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imwrite`` instead.
```

```
Out[2]: <function matplotlib.pyplot.show>
```



** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX

hw8

May 15, 2018

1 Computer Vision

1.1 Assigment 8

1.2 April 29, 2018

This assignment contains Tensorflow programming exercises.

1.3 Problem 1: Install Tensorflow

Follow the directions on https://www.tensorflow.org/install/ to install Tensorflow on your computer.

Note: You will not need GPU support for this assignment so don't worry if you don't have one. Furthermore, installing with GPU support is often more difficult to configure so it is suggested that you install the CPU only version. However, if you have a GPU and would like to install GPU support feel free to do so at your own risk :)

Note: On windows, Tensorflow is only supported in python3, so you will need to install python3 for this assignment.

Run the following cell to verify your instalation.

```
In [1]: import tensorflow as tf
        hello = tf.constant('Hello, TensorFlow!')
        sess = tf.Session()
        print(sess.run(hello))
```

```
C:\Anaconda\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argu
  from ._conv import register_converters as _register_converters


b'Hello, TensorFlow!'
```

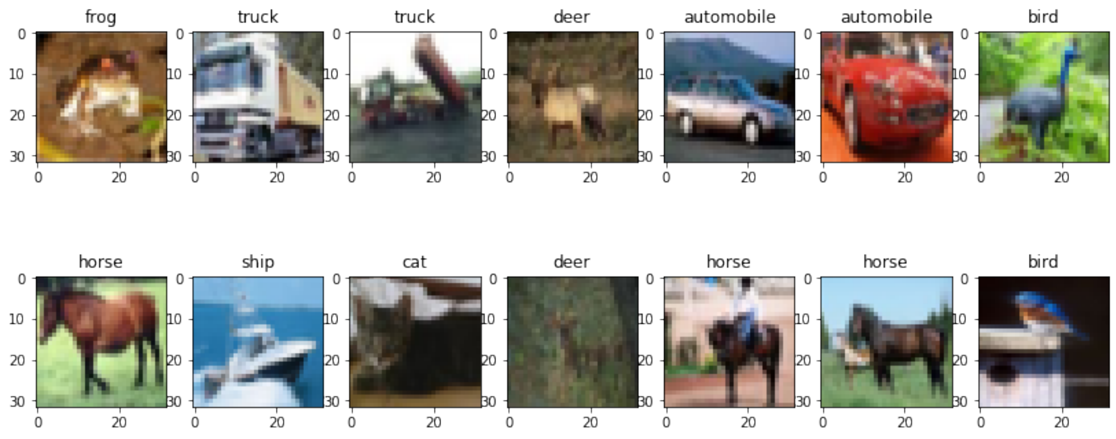
1.4 Problem 2: Downloading CIFAR10

Download the CIFAR10 dataset (http://www.cs.toronto.edu/~kriz/cifar.html). You will need the python version: http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

Extract the data to ./data Once extracted run the following cell to view a few example images.

```
In [2]: import numpy as np

        # unpickles raw data files
        def unpickle(file):
            import pickle
            import sys
            with open(file, 'rb') as fo:
                if sys.version_info[0] < 3:
                    dict = pickle.load(fo)
                else:
                   dict = pickle.load(fo, encoding='bytes')
            return dict

        # loads data from a single file
        def getBatch(file):
            dict = unpickle(file)
            data = dict[b'data'].reshape(-1,3,32,32).transpose(0,2,3,1)
            labels = np.asarray(dict[b'labels'], dtype=np.int64)
            return data,labels

        # loads all training and testing data
        def getData(path='./data'):
            classes = [s.decode('UTF-8') for s in unpickle(path+'/batches.meta')[b'label_names

            trainData, trainLabels = [], []
            for i in range(5):
                data, labels = getBatch(path+'/data_batch_%d'%(i+1))
                trainData.append(data)
                trainLabels.append(labels)
            trainData = np.concatenate(trainData)
            trainLabels = np.concatenate(trainLabels)

            testData, testLabels = getBatch(path+'/test_batch')
            return classes, trainData, trainLabels, testData, testLabels

        # training and testing data that will be used in the following problems
        classes, trainData, trainLabels, testData, testLabels = getData()

        # display some example images
        import matplotlib.pyplot as plt
        %matplotlib inline

        plt.figure(figsize=(14, 6))
        for i in range(14):
            plt.subplot(2,7,i+1)
            plt.imshow(trainData[i])
            plt.title(classes[trainLabels[i]])
        plt.show()
```

```
print ('train shape: ' + str(trainData.shape) + ', ' + str(trainLabels.shape))
print ('test shape : ' + str(testData.shape) + ', ' + str(testLabels.shape))
```



```
train shape: (50000, 32, 32, 3), (50000,)
test shape : (10000, 32, 32, 3), (10000,)
```

Below are some helper functions that will be used in the following problems.

```
In [51]: # a generator for batches of data
         # yields data (batchsize, 3, 32, 32) and labels (batchsize)
         # if shuffle, it will load batches in a random order
         def DataBatch(data, label, batchsize, shuffle=True):
             n = data.shape[0]
             #print('datashape=',data.shape)
             #print('n=',n)
             #print('haha=',data[10].shape)
             #print('original_label=',label)
             if shuffle:
                 index = np.random.permutation(n)
             else:
                 index = np.arange(n)
             for i in range(int(np.ceil(n/batchsize))):
                 inds = index[i*batchsize : min(n,(i+1)*batchsize)]
                 yield data[inds], label[inds]

         # tests the accuracy of a classifier
         def test(testData, testLabels, classifier):
             batchsize=50
             correct=0.
             i=0
```

```
        #print('testLabels=',testLabels.shape)
        for data,label in DataBatch(testData,testLabels,batchsize):
            #print('label=',label.shape)
            #print('label=',label)
            #print('data=',data.shape)
            prediction = classifier(data)
            #print ('prediction=',prediction)
            correct += np.sum(prediction==label)
        #print('data=',data)
        return correct/testData.shape[0]*100

    # a sample classifier
    # given an input it outputs a random class
    class RandomClassifier():
        def __init__(self, classes=10):
            self.classes=classes
        def __call__(self, x):
            return np.random.randint(self.classes, size=x.shape[0])

    randomClassifier = RandomClassifier()
    print ('Random classifier accuracy: %f'%test(testData, testLabels, randomClassifier))
```

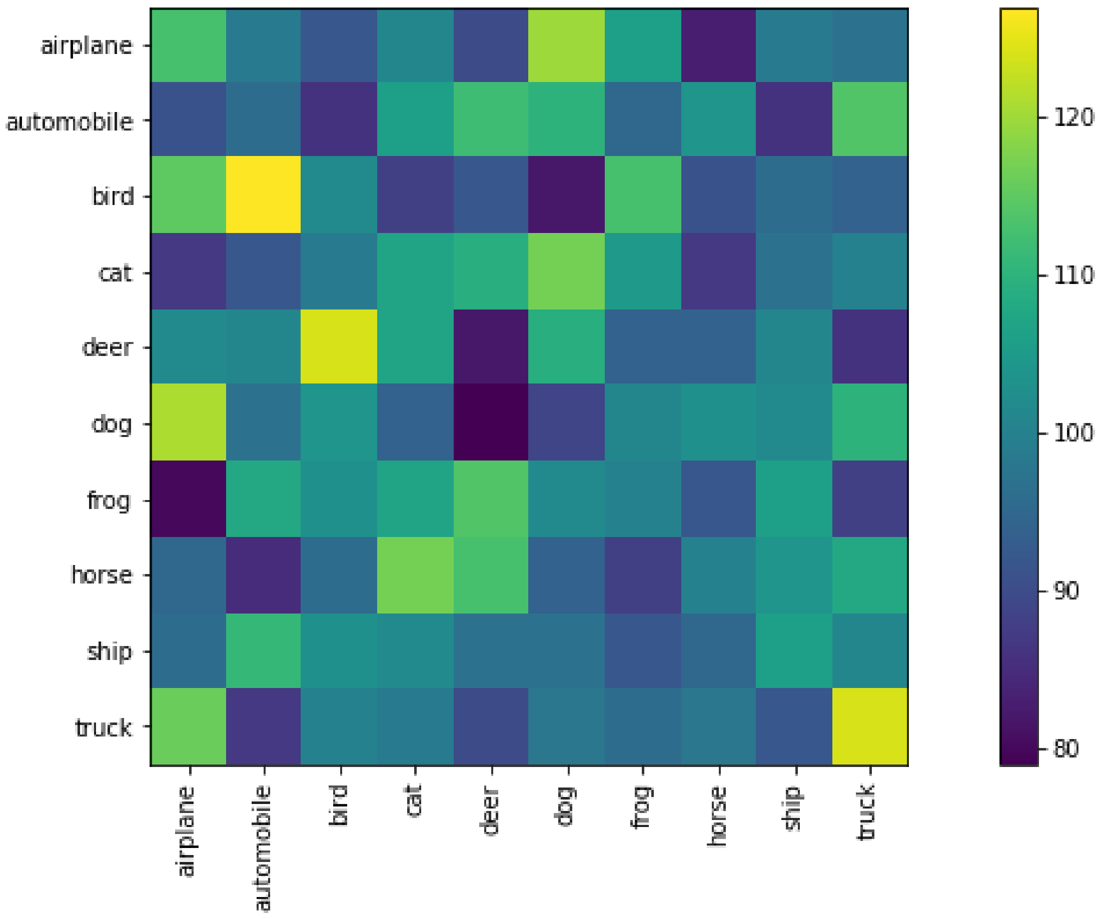
```
Random classifier accuracy: 10.350000
```

1.5 Problem 3: Confusion Matirx

Here you will implement a test script that computes the confussion matrix for a classifier. The matrix should be nxn where n is the number of classes. Entry M[i,j] should contain the number of times an image of class i was classified as class j. M should be normalized such that each row sums to 1.

Hint: see the function test() above for reference.

```
In [4]: def confusion(testData, testLabels, classifier):
            n=testData.shape[0]
            output=classifier(testData)
            M=np.zeros((10,10))
            for i in range (n):
                M[testLabels[i],output[i]]+=1
            return M


        def VisualizeConfussion(M):
            plt.figure(figsize=(14, 6))
            plt.imshow(M)#, vmin=0, vmax=1)
            plt.xticks(np.arange(len(classes)), classes, rotation='vertical')
            plt.yticks(np.arange(len(classes)), classes)
            plt.colorbar()
            plt.show()

        M = confusion(testData, testLabels, randomClassifier)
        VisualizeConfussion(M)
```



1.6 Problem 4: K-Nearest Neighbors (KNN)

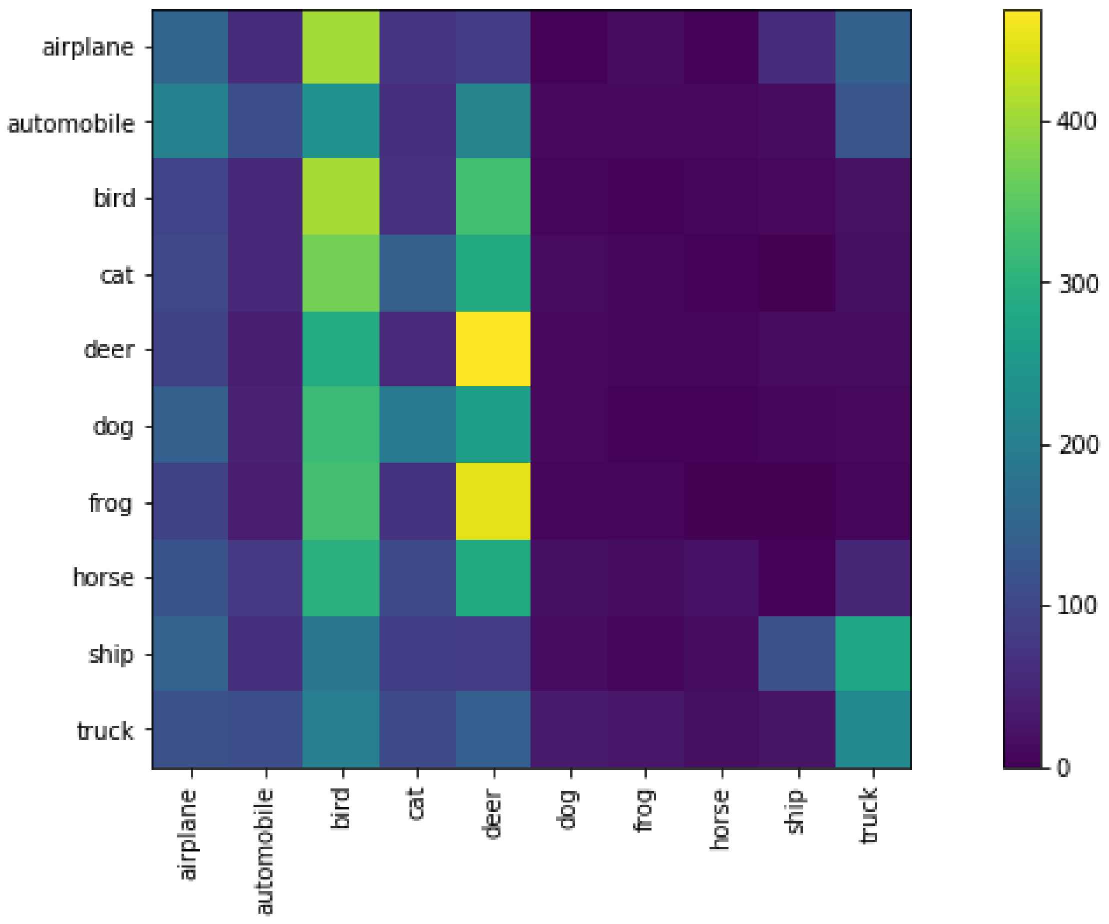
Here you will implemnet a simple knn classifer. The distance metric is euclidian in pixel space. k refers to the number of neighbors involved in voting on the class.

Hint: you may want to use: sklearn.neighbors.KNeighborsClassifier

```
In [54]: from sklearn.neighbors import KNeighborsClassifier
         class KNNClassifer():
             def __init__(self, k=3):
                 # k is the number of neighbors involved in voting
                 self.k=k
                 self.neigh = KNeighborsClassifier(n_neighbors=self.k)
```

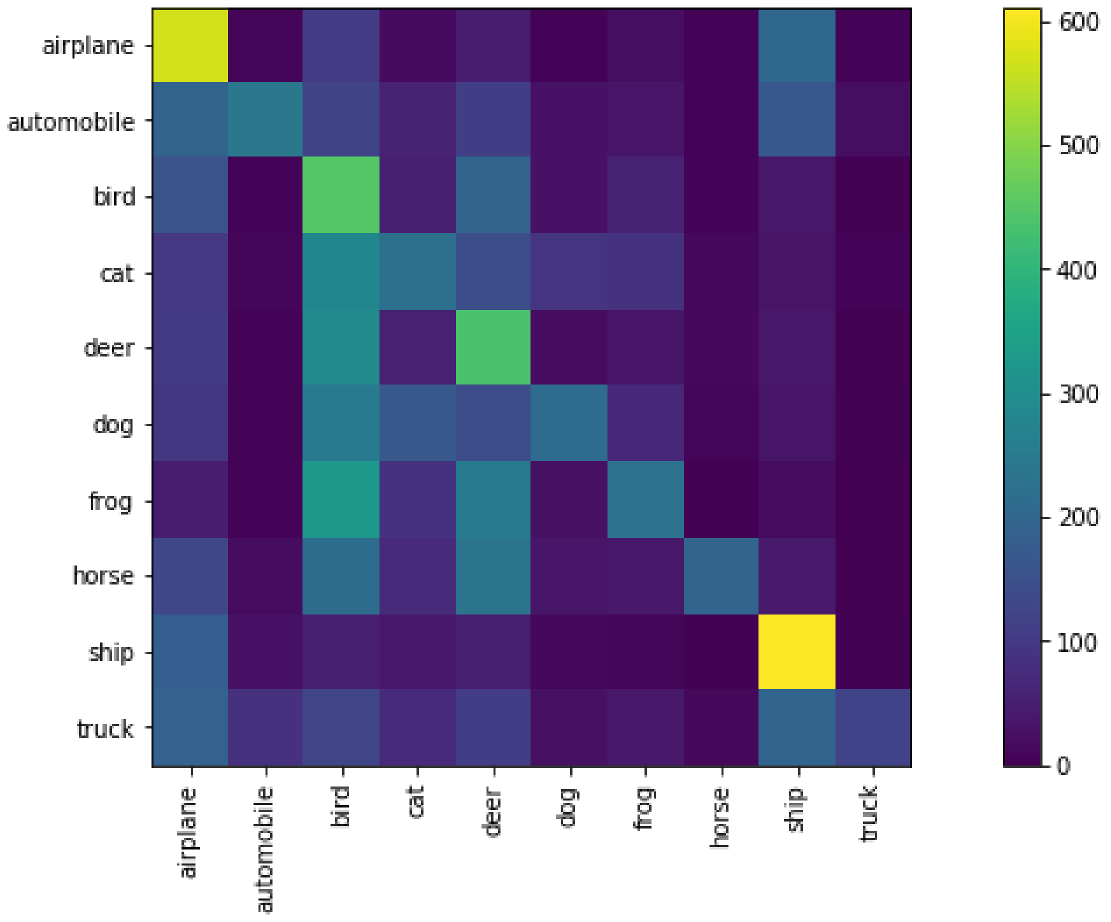
```
    def train(self, trainData, trainLabels):
        self.neigh.fit(trainData.reshape(trainData.shape[0], -1), trainLabels)




    def __call__(self, x):
        # this method should take a batch of images (batchsize, 32, 32, 3) and return
        # predictions should be int64 values in the range [0,9] corrisponding to the
        prediction=self.neigh.predict(x.reshape(x.shape[0],-1))
        return prediction



# test your classifier with only the first 100 training examples (use this while debu
# note you should get around 10-20% accuracy
knnClassiferX = KNNClassifer()
knnClassiferX.train(trainData[:100], trainLabels[:100])
print ('KNN classifier accuracy: %f'%test(testData, testLabels, knnClassiferX))

M = confusion(testData, testLabels, knnClassiferX)
VisualizeConfussion(M)
```

```
KNN classifier accuracy: 16.600000
```



```
In [22]: # test your classifier with all the training examples (This may take a while)
         # note you should get around 30% accuracy
         knnClassifer = KNNClassifer()
         knnClassifer.train(trainData, trainLabels)
         print ('KNN classifier accuracy: %f'%test(testData, testLabels, knnClassifer))

         # display confusion matrix for your KNN classifier with all the training examples
         M = confusion(testData, testLabels, knnClassifer)
         VisualizeConfussion(M)
```

```
KNN classifier accuracy: 33.030000
```



1.7 Problem 5: Principal Component Analysis (PCA) K-Nearest Neighbors (KNN)

Here you will implemnet a simple knn classifer in PCA space. You should implement PCA yourself using svd (you may not use sklearn.decomposition.PCA or any other package that directly implements PCA transofrmations

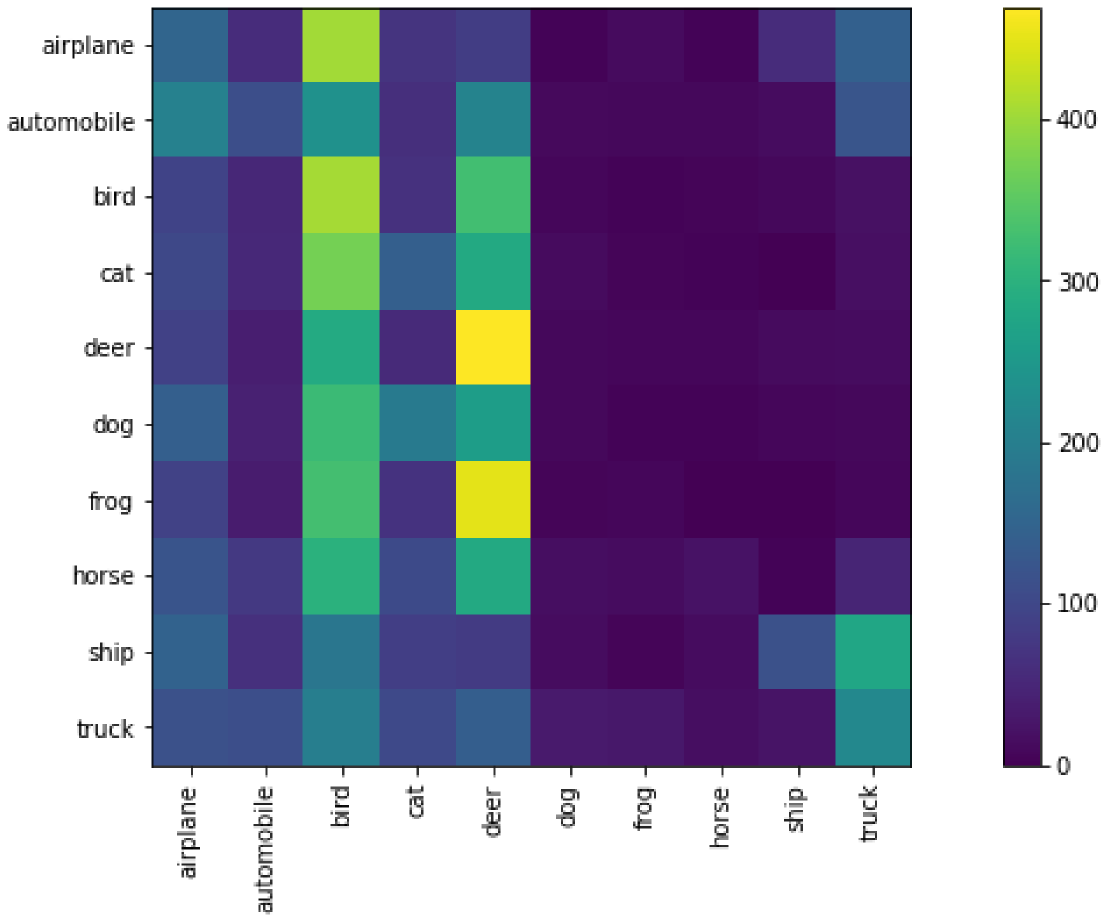
Hint: Don't forget to apply the same normalization at test time.

Note: you should get similar accuracy to above, but it should run faster.

```
In [58]: from sklearn.decomposition import PCA
         class PCAKNNClassifer():
             def __init__(self, components=25, k=3):
                 """your code here"""
                 self.k=k
                 self.components=components
                 self.neigh = KNeighborsClassifier(n_neighbors=self.k)
                 self.pca = PCA(n_components=self.components)
```

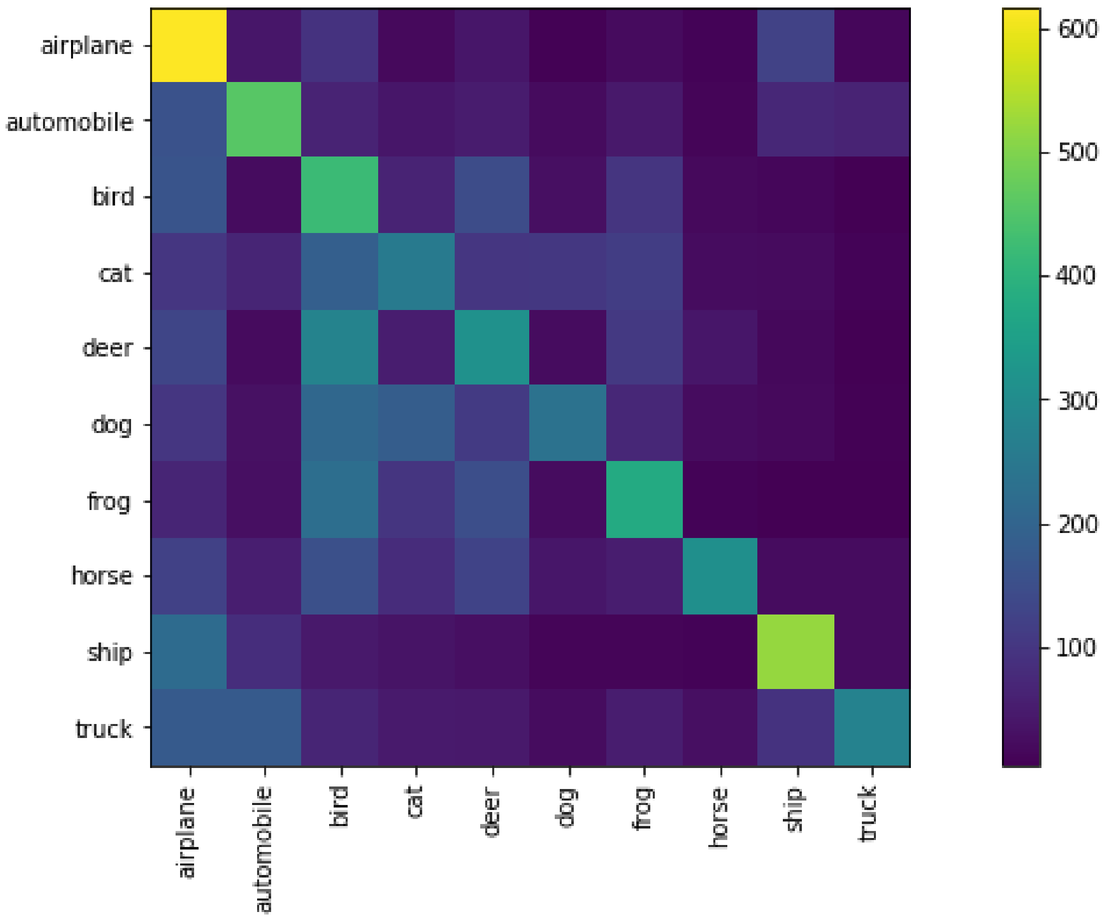
```
    def train(self, trainData, trainLabels):
        """your code here"""
        newX=self.pca.fit_transform(trainData.reshape(trainData.shape[0],-1))
        self.neigh.fit(newX, trainLabels)

    def __call__(self, x):
        """your code here"""
        test=self.pca.transform(x.reshape(x.shape[0],-1))
        prediction=self.neigh.predict(test)
        return prediction



# test your classifier with only the first 100 training examples (use this while debu
pcaknnClassiferX = PCAKNNClassifer()
pcaknnClassiferX.train(trainData[:100], trainLabels[:100])
print ('PCA-KNN classifier accuracy: %f'%test(testData, testLabels, pcaknnClassiferX)

M = confusion(testData, testLabels, knnClassiferX)
VisualizeConfussion(M)
```

```
PCA-KNN classifier accuracy: 16.450000
```



```
In [57]: # test your classifier with all the training examples (This may take a few minutes)
         pcaknnClassifer = PCAKNNClassifer()
         pcaknnClassifer.train(trainData, trainLabels)
         print ('KNN classifier accuracy: %f'%test(testData, testLabels, pcaknnClassifer))

         # display the confusion matrix
         M = confusion(testData, testLabels, pcaknnClassifer)
         VisualizeConfussion(M)
```

```
KNN classifier accuracy: 37.860000
```



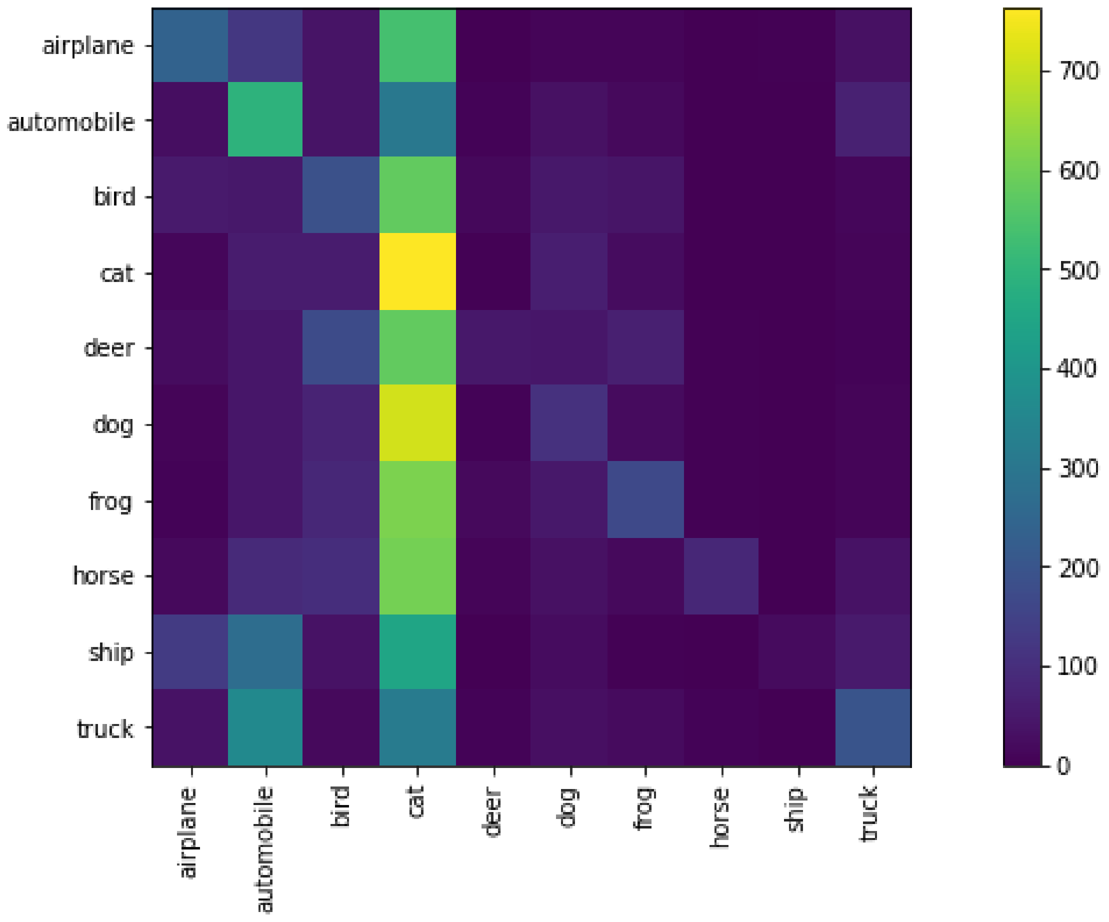
1.8 Deep learning

Below is some helper code to train your deep networks
Hint: see https://www.tensorflow.org/get_started/mnist/pros or https://www.tensorflow.org/get_started/mnist/beginners for reference

```
In [6]: # base class for your Tensorflow networks. It implements the training loop (train) and
        # You will need to implement the __init__ function to define the networks structures i
        class TFClassifier():
            def __init__(self):
                pass

            def train(self, trainData, trainLabels, epochs=1, batchsize=50):
                self.prediction = tf.argmax(self.y,1)
                self.cross_entropy = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_log
                self.train_step = tf.train.AdamOptimizer(1e-4).minimize(self.cross_entropy)
                self.correct_prediction = tf.equal(self.prediction, self.y_)
                self.accuracy = tf.reduce_mean(tf.cast(self.correct_prediction, tf.float32))
                self.sess.run(tf.global_variables_initializer())

                for epoch in range(epochs):
                    for i, (data,label) in enumerate(DataBatch(trainData, trainLabels, batchsi
                        _, acc = self.sess.run([self.train_step, self.accuracy], feed_dict={se
                        #if i%100==99:
                        #    print ('%d/%d %d %f'%(epoch, epochs, i, acc))

                    print ('testing epoch:%d accuracy: %f'%(epoch+1, test(testData, testLabels

            def __call__(self, x):
                return self.sess.run(self.prediction, feed_dict={self.x: x})

        # helper function to get weight variable
        def weight_variable(shape):
            initial = tf.truncated_normal(shape, stddev=0.01)
            return tf.Variable(initial)

        # helper function to get bias variable
        def bias_variable(shape):
            initial = tf.constant(0.1, shape=shape)
            return tf.Variable(initial)

        # example linear classifier
        class LinearClassifer(TFClassifier):
            def __init__(self, classes=10):
                self.sess = tf.Session()

                self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of ima
                print('self.x=',self.x.shape)
                self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels
                print('self.y_=',self.y_.shape)

                # model variables
                self.W = weight_variable([32*32*3,classes])
                print('self.W=',self.W.shape)
                self.b = bias_variable([classes])
                print('self.b=',self.b.shape)
                haha=tf.reshape(self.x,(-1,32*32*3))
                print('haha=',haha.shape)
                # linear operation
                hehe= tf.matmul(tf.reshape(self.x,(-1,32*32*3)),self.W)
                print('hehe=',hehe.shape)
                self.y = tf.matmul(tf.reshape(self.x,(-1,32*32*3)),self.W) + self.b
                print('self.y=',self.y.shape)
                print((self.y[0]-hehe[0]).shape)
        # test the example linear classifier (note you should get around 20-30% accuracy)
```

```
        linearClassifer = LinearClassifer()
        linearClassifer.train(trainData, trainLabels, epochs=20)

        # display confusion matrix
        M = confusion(testData, testLabels, linearClassifer)
        VisualizeConfussion(M)
```

```
self.x= (?, 32, 32, 3)
self.y_= (?,)
self.W= (3072, 10)
self.b= (10,)
haha= (?, 3072)
hehe= (?, 10)
self.y= (?, 10)
(10,)
testing epoch:1 accuracy: 25.290000
testing epoch:2 accuracy: 26.910000
testing epoch:3 accuracy: 23.680000
testing epoch:4 accuracy: 24.960000
testing epoch:5 accuracy: 25.980000
testing epoch:6 accuracy: 23.680000
testing epoch:7 accuracy: 28.540000
testing epoch:8 accuracy: 25.230000
testing epoch:9 accuracy: 27.870000
testing epoch:10 accuracy: 23.630000
testing epoch:11 accuracy: 31.490000
testing epoch:12 accuracy: 22.800000
testing epoch:13 accuracy: 25.650000
testing epoch:14 accuracy: 25.280000
testing epoch:15 accuracy: 23.600000
testing epoch:16 accuracy: 29.830000
testing epoch:17 accuracy: 27.190000
testing epoch:18 accuracy: 28.650000
testing epoch:19 accuracy: 25.610000
testing epoch:20 accuracy: 23.220000
```



1.9 Problem 6: Multi Layer Perceptron (MLP)

Here you will implement an MLP. The MLP shoud consist of 3 linear layers (matrix multiplcation and bias offset) that map to the following feature dimensions:

32x32x3 -> hidden

hidden -> hidden

hidden -> classes

The first two linear layers should be followed with a ReLU nonlinearity. The final layer should not have a nonlinearity applied as we desire the raw logits output (see: the documentation for tf.nn.sparse_softmax_cross_entropy_with_logits used in the training)

The final output of the computation graph should be stored in self.y as that will be used in the training.

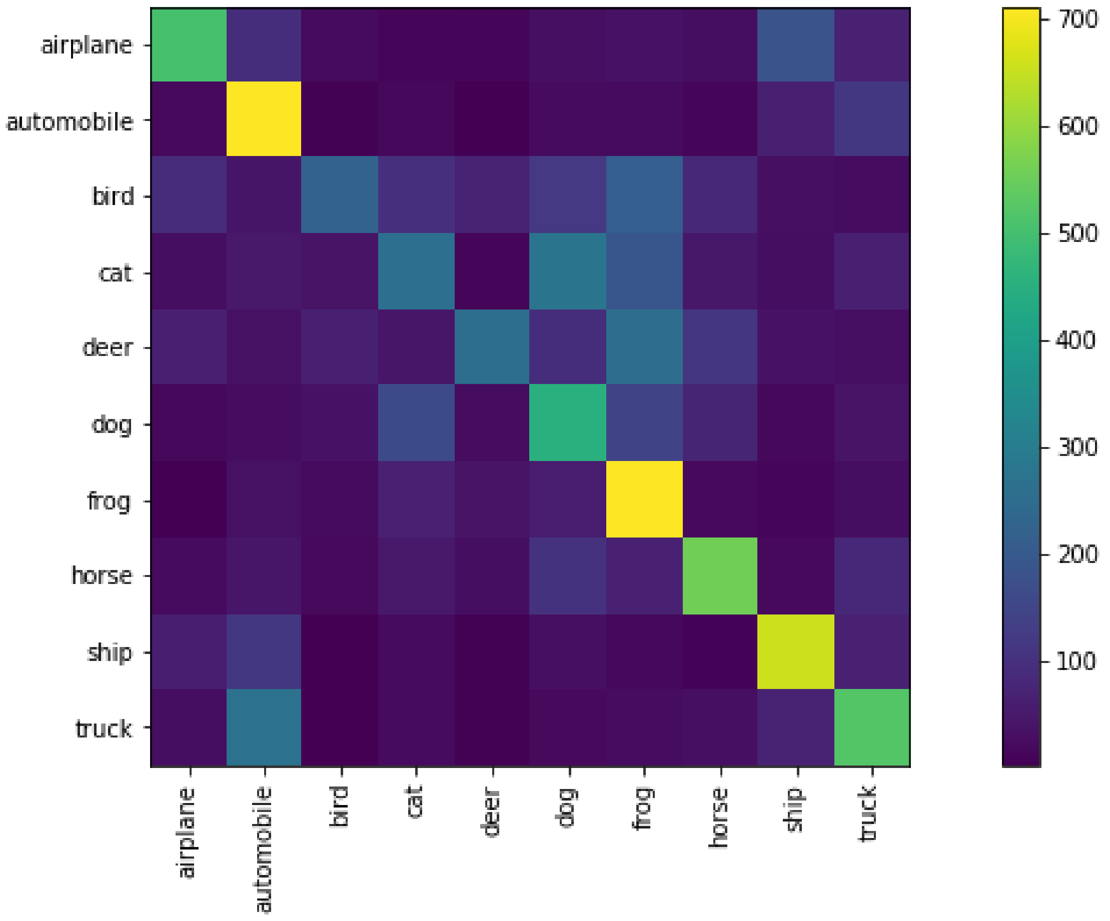
Hint: see the example linear classifier

Note: you should get around 50% accuracy

```
In [11]: class MLPClassifer(TFClassifier):
             def __init__(self, classes=10, hidden=100):
                 self.sess = tf.Session()
```

```
            self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of im
            self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels

            """your code here"""
            W_1 = weight_variable([32*32*3,hidden])
            b_1 = bias_variable([hidden])
            layer_0_flat=tf.reshape(self.x,[-1,32*32*3])
            layer_1 = tf.nn.relu(tf.matmul(layer_0_flat,W_1)+b_1)
            W_2=weight_variable([hidden,hidden])
            b_2=bias_variable([hidden])
            layer_1_flat=tf.reshape(layer_1,[-1,hidden])
            layer_2=tf.nn.relu(tf.matmul(layer_1_flat,W_2)+b_2)
            W_3 = weight_variable([hidden,classes])
            b_3 = bias_variable([classes])
            layer_2_flat=tf.reshape(layer_2,[-1,hidden])
            layer_3=tf.matmul(layer_2_flat, W_3)+b_3
            print('layer_3=',layer_3.shape)
            self.y=layer_3



    # test your MLP classifier (note you should get around 50% accuracy)
    mlpClassifer = MLPClassifer()
    mlpClassifer.train(trainData, trainLabels, epochs=20)

    # display confusion matrix
    M = confusion(testData, testLabels, mlpClassifer)
    VisualizeConfussion(M)
```

```
layer_3= (?, 10)
testing epoch:1 accuracy: 38.940000
testing epoch:2 accuracy: 42.820000
testing epoch:3 accuracy: 44.400000
testing epoch:4 accuracy: 45.440000
testing epoch:5 accuracy: 47.350000
testing epoch:6 accuracy: 45.790000
testing epoch:7 accuracy: 47.570000
testing epoch:8 accuracy: 45.560000
testing epoch:9 accuracy: 48.010000
testing epoch:10 accuracy: 48.310000
testing epoch:11 accuracy: 48.360000
testing epoch:12 accuracy: 48.890000
testing epoch:13 accuracy: 49.520000
testing epoch:14 accuracy: 48.100000
testing epoch:15 accuracy: 47.940000
testing epoch:16 accuracy: 48.760000
testing epoch:17 accuracy: 47.800000
testing epoch:18 accuracy: 49.840000
testing epoch:19 accuracy: 49.510000
testing epoch:20 accuracy: 48.650000
```



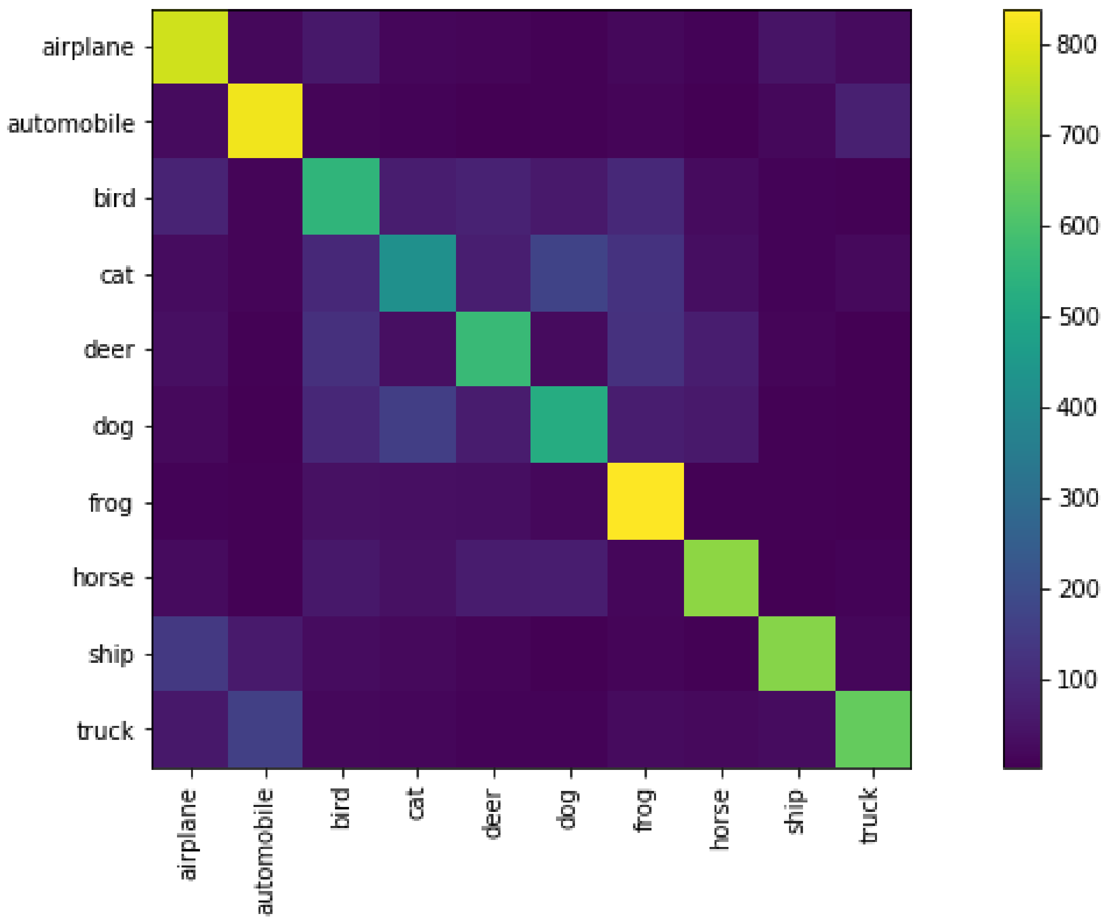
1.10 Problem 7: Convolutional Neural Netork (CNN)

Here you will implement a CNN with the following architecture:
    ReLU(Conv(kernel_size=4x4 stride=2, output_features=n))
    ReLU(Conv(kernel_size=4x4 stride=2, output_features=n*2))
    ReLU(Conv(kernel_size=4x4 stride=2, output_features=n*4))
    Linear(output_features=classes)

```
In [16]: def conv2d(x, W, stride=2):
             return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='SAME')

         class CNNClassifer(TFClassifier):
             def __init__(self, classes=10, n=16):
                 self.sess = tf.Session()
                 self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of im
                 self.y_ = tf.placeholder(tf.int64, shape=[None]) # input label
                 """your code here"""

                 #convolution process
                 W_conv1=weight_variable([4,4,3,n])
                 b_conv1=bias_variable([n])
                 h_conv1=tf.nn.relu(conv2d(self.x, W_conv1)+b_conv1)
                 print('h_conv1=',h_conv1.shape)
                 W_conv2=weight_variable([4,4,n,n*2])
                 b_conv2=bias_variable([n*2])
                 h_conv2=tf.nn.relu(conv2d(h_conv1, W_conv2)+b_conv2)
                 print('h_conv2=',h_conv2.shape)
                 W_conv3=weight_variable([4,4,n*2,n*4])
                 b_conv3=bias_variable([n*4])
                 h_conv3=tf.nn.relu(conv2d(h_conv2, W_conv3)+b_conv3)
                 #h_conv3_flat=tf.reshape(h_conv3, [-1,4,4,n**4])
                 print('h_conv3=',h_conv3.shape)

                 '''W_conv4=weight_variable([4,4,n*4,n*8])
                 b_conv4=bias_variable([n*8])
                 h_conv4=tf.nn.relu(conv2d(h_conv3, W_conv4)+b_conv4)
                 print('h_conv4=',h_conv4.shape)'''

                 #FC process
                 W_fc1=weight_variable([4*4*(n*4),classes])
                 b_fc1=bias_variable([classes])
                 h_conv3_flat=tf.reshape(h_conv3, [-1,4*4*(n*4)])
                 print('h_conv3_flat=',h_conv3_flat.shape)
                 print('W_fc1=',W_fc1.shape)
                 h_fc1=tf.nn.relu(tf.matmul(h_conv3_flat, W_fc1)+b_fc1)
                 print('h_fc1=',h_fc1.shape)
                 self.y=h_fc1

         # test your CNN classifier (note you should get around 65% accuracy)
         cnnClassifer = CNNClassifer()
         cnnClassifer.train(trainData, trainLabels, epochs=20)

         # display confusion matrix
         M = confusion(testData, testLabels, cnnClassifer)
         VisualizeConfussion(M)
```

```
h_conv1= (?, 16, 16, 16)
h_conv2= (?, 8, 8, 32)
h_conv3= (?, 4, 4, 64)
h_conv3_flat= (?, 1024)
W_fc1= (1024, 10)
```

```
h_fc1= (?, 10)
testing epoch:1 accuracy: 43.550000
testing epoch:2 accuracy: 49.310000
testing epoch:3 accuracy: 52.930000
testing epoch:4 accuracy: 55.570000
testing epoch:5 accuracy: 57.130000
testing epoch:6 accuracy: 59.660000
testing epoch:7 accuracy: 59.940000
testing epoch:8 accuracy: 60.890000
testing epoch:9 accuracy: 62.250000
testing epoch:10 accuracy: 60.820000
testing epoch:11 accuracy: 62.640000
testing epoch:12 accuracy: 64.470000
testing epoch:13 accuracy: 65.220000
testing epoch:14 accuracy: 64.520000
testing epoch:15 accuracy: 65.450000
testing epoch:16 accuracy: 64.630000
testing epoch:17 accuracy: 64.340000
testing epoch:18 accuracy: 66.080000
testing epoch:19 accuracy: 65.480000
testing epoch:20 accuracy: 65.320000
```



1.11 Further reference

To see how state of the art deep networks do on this dataset see: https://github.com/tensorflow/models/tree/master/research/resnet