# CSE527 Homework1

**Due date: 23:59 on Sep 24, 2019 (Tuesday)**

---

In this semester, we will use Google Colab for the assignments, which allows us to utilize resources that some of us might not have in their local machines such as GPUs. You will need to use your Stony Brook (*.stonybrook.edu) account for coding and Google Drive to save your results.

## Google Colab Tutorial

---

Go to https://colab.research.google.com/notebooks/ (https://colab.research.google.com/notebooks/), you will see a tutorial named "Welcome to Colaboratory" file, where you can learn the basics of using google colab.

Settings used for assignments: ***Edit -> Notebook Settings -> Runtime Type (Python 3)***.

## Local Machine Prerequisites

---

Since we are using Google Colab, all the code is run on the server environment where lots of libraries or packages have already been installed. In case of missing libraries or if you want to install them in your local machine, below are the links for installation.

- **Install Python 3.6**: https://www.python.org/downloads/ (https://www.python.org/downloads/) or use Anaconda (a Python distribution) at https://docs.continuum.io/anaconda/install/ (https://docs.continuum.io/anaconda/install/). Below are some materials and tutorials which you may find useful for learning Python if you are new to Python.
  - https://docs.python.org/3.6/tutorial/index.html (https://docs.python.org/3.6/tutorial/index.html)
  - https://www.learnpython.org/ (https://www.learnpython.org/)
  - http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html (http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html)
  - http://www.scipy-lectures.org/advanced/image_processing/index.html (http://www.scipy-lectures.org/advanced/image_processing/index.html)

- **Install Python packages**: install Python packages: `numpy` , `matplotlib` , `opencv-python` using pip, for example:

  ```
  pip install numpy matplotlib opencv-python
  ```

  Note that when using “pip install”, make sure that the version you are using is python3. Below are some commands to check which python version it uses in you machine. You can pick one to execute:

  ```
  pip show pip

  pip --version

  pip -V
  ```

Incase of wrong version, use pip3 for python3 explictly.

- **Install Jupyter Notebook**: follow the instructions at http://jupyter.org/install.html (http://jupyter.org/install.html) to install Jupyter Notebook and familiarize yourself with it. *After you have installed Python and Jupyter Notebook, please open the notebook file 'HW1.ipynb' with your Jupyter Notebook and do your homework there.*

# Example

Please read through the following examples where we apply image thresholding to an image. This example is desinged to help you get familiar with the basics of Python and routines of OpenCV. This part is for your exercises only, you do not need to submit anything from this part.

In [0]:

```
import sys
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage, misc
from IPython.display import display, Image
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

In [2]:

```
# Mount your google drive where you've saved your assignment folder
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2F
www.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fau
th%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=
code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:
```
# function for image thresholding
def imThreshold(img, threshold, maxVal):
    assert len(img.shape) == 2 # input image has to be gray

    height, width = img.shape
    bi_img = np.zeros((height, width), dtype=np.uint8)
    for x in range(height):
        for y in range(width):
            if img.item(x, y) > threshold:
                bi_img.itemset((x, y), maxVal)

    return bi_img
```

In [5]:
```
# read the image for local directory (same with this .ipynb)
img = cv2.imread('SourceImages/Snow.jpg')

# convert a color image to gray
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# image thresholding using global tresholder
img_bi = imThreshold(img_gray, 127, 255)

# Be sure to convert the color space of the image from
# BGR (Opencv) to RGB (Matplotlib) before you show a
# color image read from OpenCV
plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('original image')
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(img_gray, 'gray')
plt.title('gray image')
plt.axis("off")

plt.subplot(1, 3, 3)
plt.imshow(img_bi, 'gray')
plt.title('binarized image')
plt.axis("off")

plt.show()
```

# Description

There are five basic image processing problems in total with specific instructions for each of them. Be sure to read **Submission Guidelines** below. They are important.

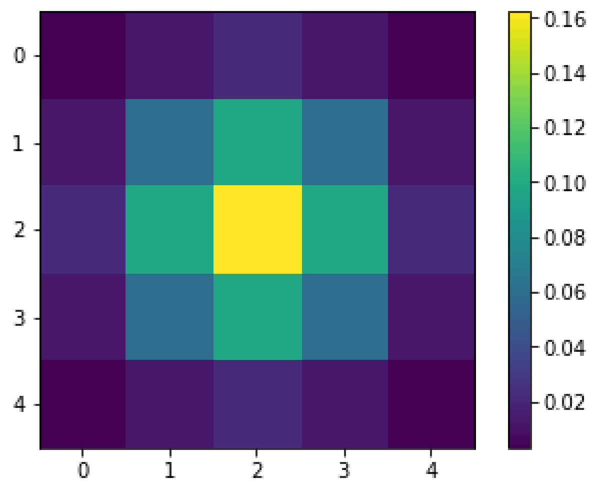
# Problems

- **Problem 1.a Gaussian convolution {10 pts}:** Write a function in Python that takes two arguments, a width parameter and a variance parameter, and returns a 2D array containing a Gaussian kernel of the desired dimension and variance. The peak of the Gaussian should be in the center of the array. Make sure to normalize the kernel such that the sum of all the elements in the array is 1. Use this function and the OpenCV's `filter2D` routine to convolve the image and noisy image arrays with a 5 by 5 Gaussian kernel with sigma of 1. Repeat with a 11 by 11 Gaussian kernel with a sigma of 3. There will be four output images from this problem, namely, image convolved with 3x3, 11x11, noisy image convolved with 3x3, and 11x11. Once you fill in and run the codes, the outputs will be saved under Results folder. These images will be graded based on the difference with ground truth images. You might want to try the same thing on other images but it is not required. Include your notebook and the saved state where the output is displayed in the notebook.

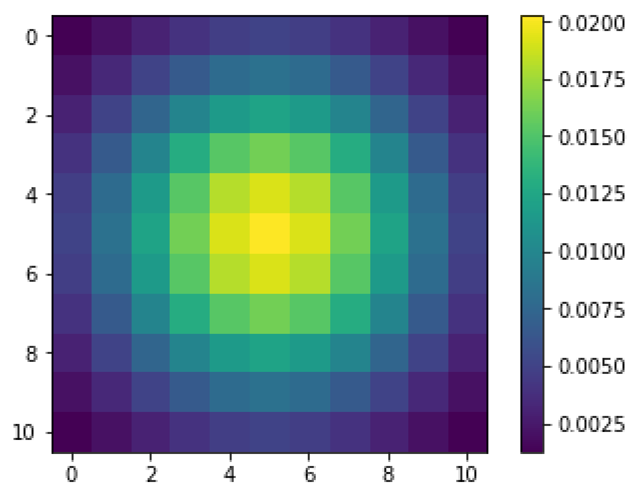
In [6]:
```
def genGaussianKernel(width, sigma):

    # define your 2d kernel here

    kernel_2d = np.zeros((width,width))

    diff = int((width - 1)/2)
    constant = 1 / (2 * np.pi * sigma**2)

    for i in range(0,width):
      for j in range(0,width):
        kernel_2d[i][j] = np.exp(-((i-diff)**2 + (j-diff)**2)/(2* (sigma**2)))

    kernel_2d *= constant

    kernel_2d /= np.sum(kernel_2d) # normalization
    print("Sum of the kernel elements:", np.sum(kernel_2d))

    plt.imshow(kernel_2d)
    plt.colorbar()
    plt.show()

    return kernel_2d

# Load images
img       = cv2.imread('SourceImages/pic.jpg', 0)
img_noise = cv2.imread('SourceImages/pic_noisy.jpg', 0)

# Generate Gaussian kernels
kernel_1 = genGaussianKernel(5,1) #Fill in your code here      # 5 by 5 kernel
with sigma of 1
kernel_2 = genGaussianKernel(11,3) #Fill in your code here      # 11 by 11 ker
nel with sigma of 3


# Convolve with image and noisy image
res_img_kernel1 = cv2.filter2D(img, -1, kernel_1) #Fill in your code here
res_img_kernel2 = cv2.filter2D(img, -1, kernel_2) #Fill in your code here
res_img_noise_kernel1 = cv2.filter2D(img_noise, -1, kernel_1) #Fill in your co
de here
res_img_noise_kernel2 = cv2.filter2D(img_noise, -1, kernel_2) #Fill in your co
de here

# Write out result images
cv2.imwrite("Results/P1_01.jpg", res_img_kernel1)
cv2.imwrite("Results/P1_02.jpg", res_img_kernel2)
cv2.imwrite("Results/P1_03.jpg", res_img_noise_kernel1)
cv2.imwrite("Results/P1_04.jpg", res_img_noise_kernel2)

# Plot results
plt.figure(figsize = (10, 10))
plt.subplot(2, 2, 1)
plt.imshow(res_img_kernel1, 'gray')
plt.title('Image: 5x5 kernel with var as 1')
plt.axis("off")
```

```
plt.subplot(2, 2, 2)
plt.imshow(res_img_kernel2, 'gray')
plt.title('Image: 11x11 kernel with var as 3')
plt.axis("off")

plt.subplot(2, 2, 3)
plt.imshow(res_img_noise_kernel1, 'gray')
plt.title('Noisy image: 5x5 kernel with var as 1')
plt.axis("off")

plt.subplot(2, 2, 4)
plt.imshow(res_img_noise_kernel2, 'gray')
plt.title('Noisy image: 11x11 kernel with var as 3')
plt.axis("off")

plt.show()
```

```
Sum of the kernel elements: 1.0
```




```
Sum of the kernel elements: 1.0
```







- **Problem 1.b Median filter {15 pts}:** \ (a)Write a function to generate an image with salt and pepper noise. The function takes two arguments, the input image and the probability that a pixel location has salt-pepper noise. A simple implementation can be to select pixel locations with probability 'p' where noise occurs and then with equal probability set the pixel value at those location to be 0 or 255.(**Hint: Use np.random.uniform()**) \ (b)Write a function to implement a median filter. The function takes two arguments, an image and a window size(if window size is 'k', then a kxk window is used to determine the median pixel value at a location) and returns the output image. **Do not** use any inbuilt library (like scipy.ndimage_filter) to directly generate the result. \ For this question display the outputs for "probabilty of salt and pepper noise" argument in the **noisy_image_generator** function equal to 0.1 and 0.2, and median filter window size in **median_filter** function equal to 5x5. \ (c) What is the Gaussian filter size (and sigma) that achieves a similar level of noise removal.

In [7]:
```
# Function to generate image with salt and pepper noise
def noisy_image_generator(img_in, probability = 0.1):
  # define your function here
  # Fill in your code here

  height, width = img_in.shape[:2]

  img_out = np.copy(img_in)

  for h in range(height):
    for w in range(width):
      p = np.random.uniform()
      if(p < probability):  # if p < 0.1, noise occurs
        img_out[h,w] = 0 if np.random.uniform() < 0.5 else 255  # if probabili
ty less than 50% then 0 else 255 (equal probability)

  return img_out

# Function to apply median filter(window size kxk) on the input image
def median_filter(img_in, window_size = 5):
  # define your function here
  # Fill in your code here

  height, width = img_in.shape[:2]

  result = np.copy(img_in)

  diff = int((window_size - 1)/2)

#   if I had to zero pad....
#   padded_img_in = np.zeros((height + window_size-1, width + window_size-1))
#   padded_img_in[diff:-diff, diff:-diff] = img_in

  # symmetric padding
  padded_img_in = np.pad(img_in, (window_size-1, window_size-1), 'symmetric')

  for h in range(0,height):
    for w in range(0,width):

      # listing values in array according to window size to find median from i
t
      arr = [padded_img_in[i,j] for i in range(h-diff, h+diff) for j in range(
w-diff, w+diff)]

      median_in_window = int(np.median(arr))
      result[h,w] = median_in_window

  return result

image_s_p1 = noisy_image_generator(img, probability = 0.1)
result1 = median_filter(image_s_p1, window_size = 5)

image_s_p2 = noisy_image_generator(img, probability = 0.2)
result2 = median_filter(image_s_p2, window_size = 5)

cv2.imwrite("Results/P1_05.jpg", result1)
```

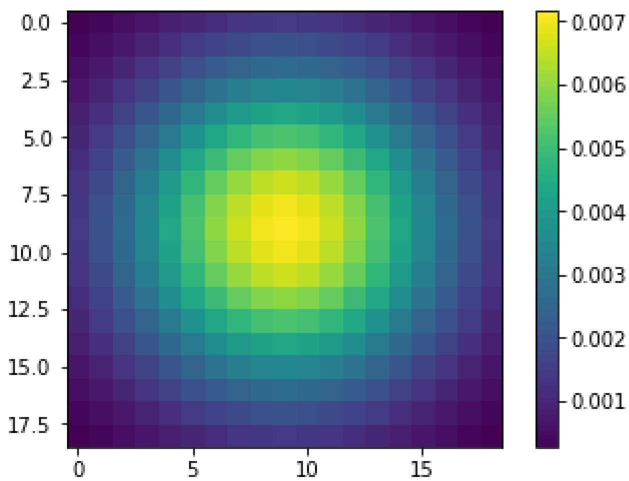
```
cv2.imwrite("Results/P1_06.jpg", result2)

# Plot results
plt.figure(figsize = (28, 20))
plt.subplot(1, 5, 1)
plt.imshow(img, 'gray')
plt.title('Original image')
plt.axis("off")

plt.subplot(1, 5, 2)
plt.imshow(image_s_p1, 'gray')
plt.title('Image with salt and pepper noise (noise_prob = 0.1)')
plt.axis("off")

plt.subplot(1, 5, 3)
plt.imshow(result1, 'gray')
plt.title('Image recovered after applying median filter')
plt.axis("off")

plt.subplot(1, 5, 4)
plt.imshow(image_s_p2, 'gray')
plt.title('Image with salt and pepper noise (noise_prob = 0.2)')
plt.axis("off")

plt.subplot(1, 5, 5)
plt.imshow(result2, 'gray')
plt.title('Image recovered after applying median filter')
plt.axis("off")

plt.show()
```


**Answer of Problem 1.b (c) What is the Gaussian filter size (and sigma) that achieves a similar level of noise removal.**

Gaussian filter size = 19x19, sigma = 5 can be used for similar level of noise removal as shown below. However, gaussian filter will not give similar denoising performance as median filter is doing in case of salt and pepper noise removal. Because to reduce noise intensity (lets say blur in case of gaussian), sigma should be increased and with higher sigma (standard deviation) high frequency details i.e. edges will get blurred.

In [8]:
```
kernel_test = genGaussianKernel(19,5) # 19 by 19 kernel with sigma of 5

result_test = cv2.filter2D(image_s_p1, -1, kernel_test)

plt.figure(figsize = (28, 20))
plt.subplot(1, 5, 1)
plt.imshow(image_s_p1, 'gray')
plt.title('Image with salt and pepper noise (noise_prob = 0.1)')
plt.axis("off")

plt.subplot(1, 5, 2)
plt.imshow(result1, 'gray')
plt.title('Image recovered after applying Median filter')
plt.axis("off")

plt.subplot(1, 5, 3)
plt.imshow(result_test, 'gray')
plt.title('Image recovered after applying Gaussian filter')
plt.axis("off")
plt.show()
```

```
Sum of the kernel elements: 1.0
```

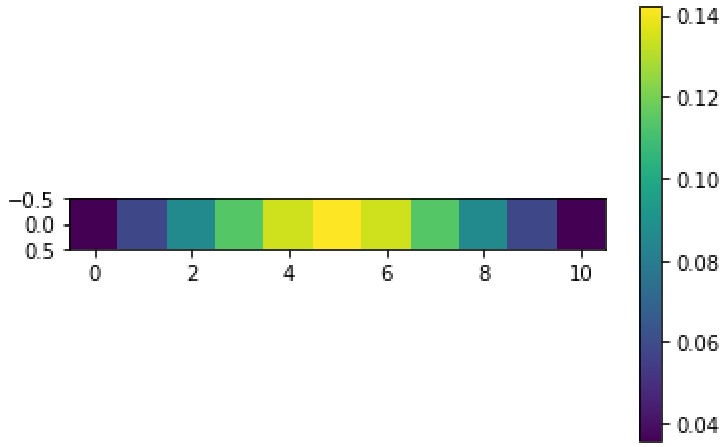



- **Problem 2 Separable convolutions {15 pts}:** The Gaussian kernel is separable, which means that convolution with a 2D Gaussian can be accomplished by convolving the image with two 1D Gaussians, one in the x direction and the other one in the y direction. Perform an 11x11 convolution with sigma = 3 from question 1 using this scheme. You can still use `filter2D` to convolve the images with each of the 1D kernels. Verify that you get the same results with what you did with 2D kernels by computing the difference image between the results from the two methods. This difference image should be close to black. Include your code and results in your colab Notebook file. There is no output image from this part. Be sure to display the result images in the notebook.

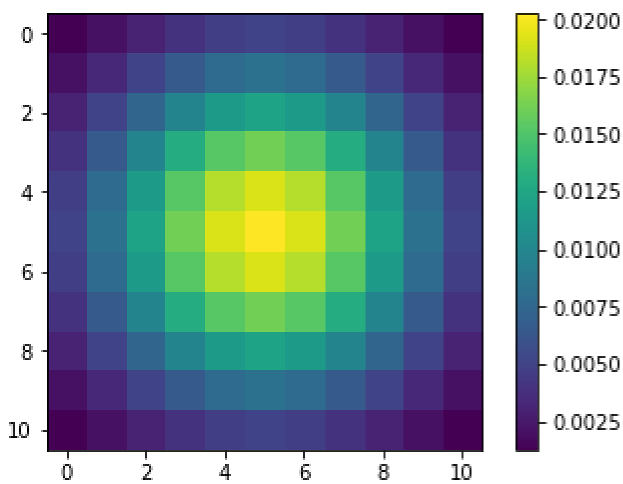
In [9]:
```
def genGausKernel1D(length, sigma):

    # define you 1d kernel here
    # Fill in your code here

    kernel_1d = np.zeros((1,length))
    diff = int((length - 1)/2)
    constant = 1 / np.sqrt(2 * np.pi * sigma**2)

    for i in range(0,length):
        kernel_1d[0][i] = np.exp(-((i-diff)**2/(2* (sigma**2))))

    kernel_1d *= constant

    kernel_1d /= np.sum(kernel_1d) # normalization
    print("Sum of the kernel elements:", np.sum(kernel_1d))

    plt.imshow(kernel_1d)
    plt.colorbar()
    plt.show()

    return kernel_1d

# Generate two 1d kernels here
width = 11
sigma = 3
kernel_x = genGausKernel1D(width, sigma) #Fill in your code here
kernel_y = kernel_x.T #Fill in your code here

# Generate a 2d 11x11 kernel with sigma of 3 here as before
kernel_2d = genGaussianKernel(width, sigma)

# Convolve with img_noise
res_img_noise_kernel1d_x  = cv2.filter2D(img, -1, kernel_x) # Fill in your cod
e here
res_img_noise_kernel1d_xy = cv2.filter2D(res_img_noise_kernel1d_x, -1, kernel_
y) # Fill in your code here
res_img_noise_kernel2d    = cv2.filter2D(img, -1, kernel_2d) # Fill in your co
de here

# Plot results
plt.figure(figsize=(22, 5))
plt.subplot(1, 4, 1)
plt.imshow(img_noise, 'gray')
plt.title('Q.2.1 Noisy image')
plt.axis("off")

plt.subplot(1, 4, 2)
plt.imshow(res_img_noise_kernel1d_x, 'gray')
plt.title('Q.2.2 Noisy img convolved with 11x11 GF in X')
plt.axis("off")

plt.subplot(1, 4, 3)
plt.imshow(res_img_noise_kernel1d_xy, 'gray')
plt.title('Q.2.3 Noisy img convolved with 11x11 GF in X and Y')
plt.axis("off")
```

```
plt.subplot(1, 4, 4)
plt.imshow(res_img_noise_kernel2d, 'gray')
plt.title('Q.2.4 Noisy img convolved with 11x11 GF in 2D')
plt.axis("off")

plt.show()

# Compute the difference array here
img_diff =  np.subtract(res_img_noise_kernel2d, res_img_noise_kernel1d_xy) # s
ubtraction # Fill in your code here

plt.gray()
plt.imshow(img_diff)
```

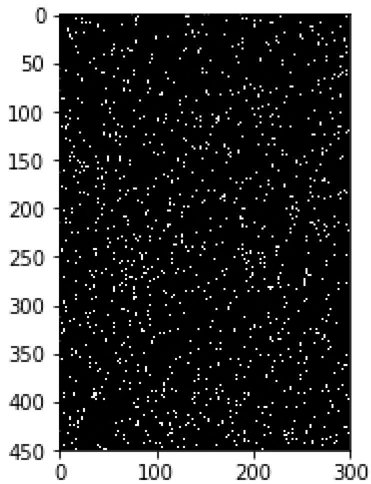
```
Sum of the kernel elements: 0.9999999999999999
```



```
Sum of the kernel elements: 1.0
```




Out[9]: <matplotlib.image.AxesImage at 0x7f279a372048>

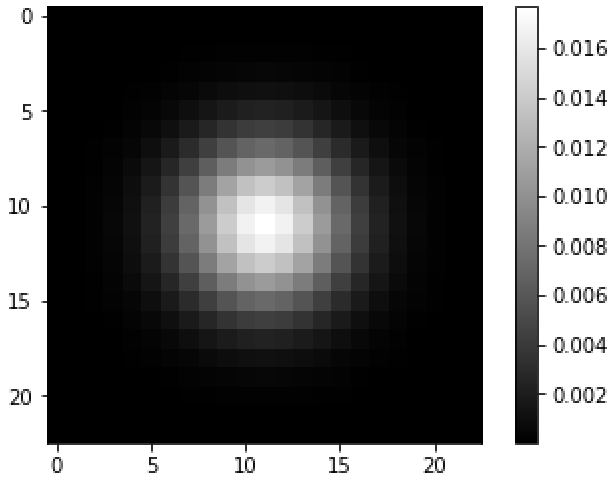


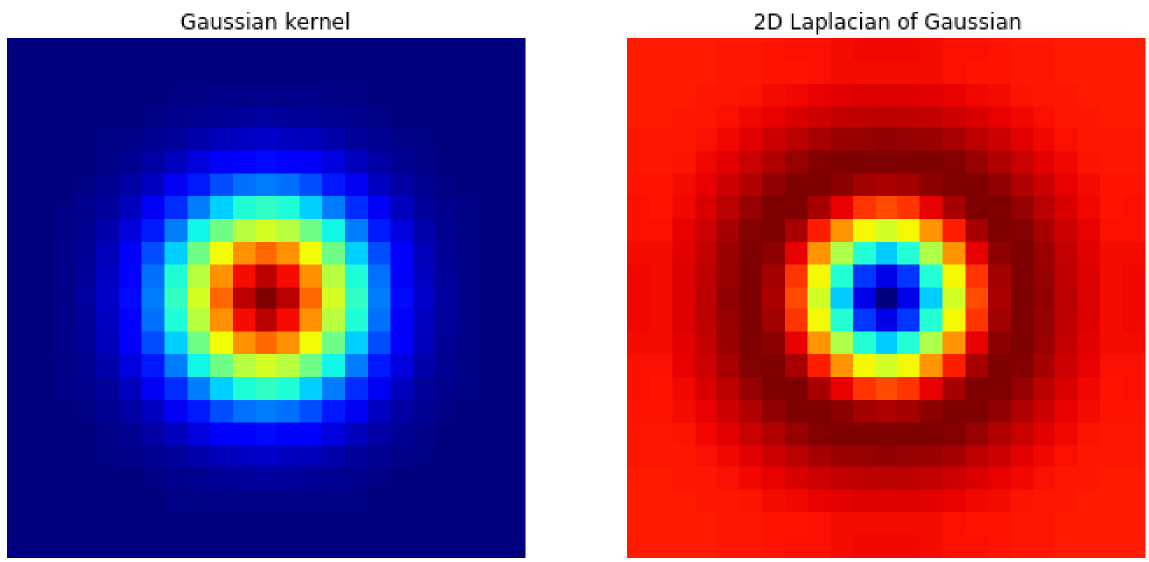

- **Problem 3 Laplacian of Gaussian {20 pts}:** Convolve a 23 by 23 Gaussian of sigma = 3 with the discrete approximation to the Laplacian kernel [1 1 1; 1 -8 1; 1 1 1]. Plot the Gaussian kernel and 2D Laplacian of Gaussian using the `Matplotlib` function `plot` . Use the `Matplotlib` function `plot_surface` to generate a 3D plot of LoG. Do you see why this is referred to as the Mexican hat filter? Include your code and results in your Colab Notebook file. Apply the filter to the **four output images generated in the previous question**. Discuss the results in terms of edge accuracy and sensitivity to noise.

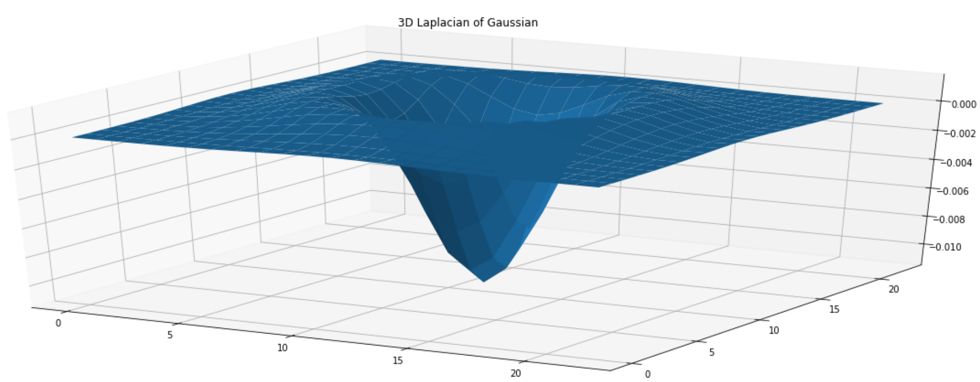
In [10]:
```
width = 23
sigma = 3

# Create your Laplacian kernel
Laplacian_kernel = np.ones((3,3))# Fill in your code here
Laplacian_kernel[1,1] = -8

# Create your Gaussian kernel
Gaussian_kernel = genGaussianKernel(width, sigma)# Fill in your code here

# Create your Laplacian of Gaussian
LoG = cv2.filter2D(Gaussian_kernel, -1, Laplacian_kernel)# Fill in your code h
ere

# Plot Gaussian and Laplacian
fig = plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.imshow(Gaussian_kernel, interpolation='none', cmap=cm.jet)
plt.title('Gaussian kernel')
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(LoG, interpolation='none', cmap=cm.jet)
plt.title('2D Laplacian of Gaussian')
plt.axis("off")


# Plot the 3D figure of LoG
# Fill in your code here
figure = plt.figure(figsize=(18, 6))
ax = Axes3D(figure)
x_axis, y_axis = np.meshgrid(range(width), range(width))
ax.plot_surface(x_axis, y_axis, LoG)

plt.title('3D Laplacian of Gaussian')
plt.show()

img_noise_LOG =                cv2.filter2D(img_noise, -1, LoG) # Fill in you
r code here
res_img_noise_kernel1d_x_LOG =  cv2.filter2D(res_img_noise_kernel1d_x, -1, LoG
) # Fill in your code here
res_img_noise_kernel1d_xy_LOG = cv2.filter2D(res_img_noise_kernel1d_xy, -1, Lo
G) # Fill in your code here
res_img_noise_kernel2d_LOG =    cv2.filter2D(res_img_noise_kernel2d, -1, LoG)
# Fill in your code here


# Plot results

plt.figure(figsize=(18, 6))
plt.subplot(1, 4, 1)
plt.imshow(img_noise_LOG, 'gray')
plt.title('Image from Q2.1 convolved with LOG')
plt.axis("off")

plt.subplot(1, 4, 2)
```

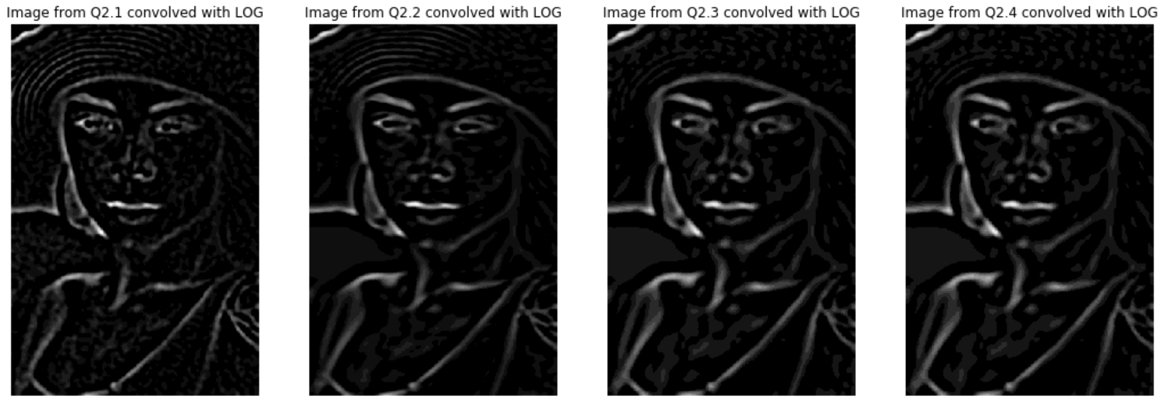
```
plt.imshow(res_img_noise_kernel1d_x_LOG, 'gray')
plt.title('Image from Q2.2 convolved with LOG')
plt.axis("off")

plt.subplot(1, 4, 3)
plt.imshow(res_img_noise_kernel1d_xy_LOG, 'gray')
plt.title('Image from Q2.3 convolved with LOG')
plt.axis("off")

plt.subplot(1, 4, 4)
plt.imshow(res_img_noise_kernel2d_LOG, 'gray')
plt.title('Image from Q2.4 convolved with LOG')
plt.axis("off")

plt.show()
```

```
Sum of the kernel elements: 1.0
```






**Discuss the results in terms of edge accuracy and sensitivity to noise.**

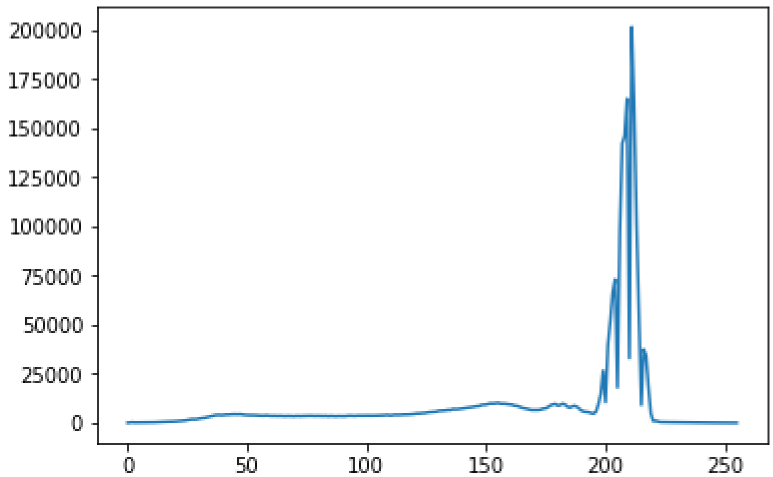
-> As laplacian filter (derivative filter) is very sensitive to noise, it is always advisable to smooth the image or lets say apply Gaussian filter before convolving with laplacian filter. Here it is visible that in first image which had noise initially is having small edges which are not meaningful or smaller compared to other important edges like face outlines. These unimportant edges are somewhere reducing the effect of the important edges because in the second image which had Gaussian filter applied in X direction, it is more clear to see the main edges of the human. Coming to the last two images, both were blurred (noise was reduced to make it less sensitive to noise) before applying laplacian, so they are able to show larger edges accurately but the smalled edges (such as lines in hat) are not accurate because of smoothing. Finally I can say that to keep it less sensitive we are losing smaller edges (like hair in first image) and if we care about smaller edges (not applying gaussian filter) it becomes very sensitive to noise.

- **Problem 4 Histogram equalization {20 pts}:** Refer to Szeliski's book on section 3.4.1, and within that section to eqn 3.9 for more information on histogram equalization. Getting the histogram of a grayscale image is incredibly easy with python. A histogram is a vector of numbers. Once you have the histogram, you can get the cumulative distribution function (CDF) from it. Then all you have left is to find the mapping from each value [0,255] to its adjusted value (just using the CDF basically). **DO NOT** use **cv2.equalizeHist()** directly to solve the exercise! We will expect to see in your code that you get the PDF and CDF, and that you manipulate the pixels directly (avoid a for loop, though). There will be one output image from this part which is the histogram equalized image. It will be compared against the ground truth.

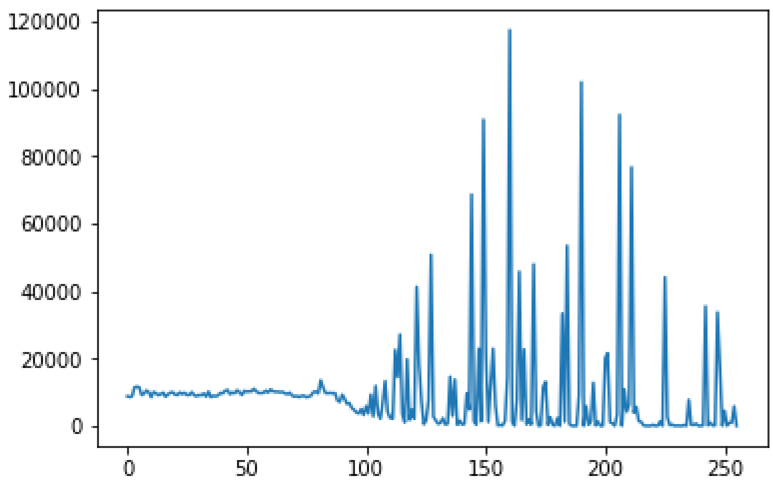
In [11]:
```
def histogram_equalization(img_in):

    # Write histogram equalization here
    # Fill in your code here

    grayscale_img = cv2.cvtColor(img_in, cv2.COLOR_BGR2GRAY)  # rgb to graysca
le

    histogram = np.zeros(256)
    height, width, channel = img_in.shape
    for h in range(height):
      for w in range(width):
        histogram[grayscale_img[h,w]] += 1

    print("histogram before equalization using grayscale")
    plt.plot(histogram)   # histogram before equalization
    plt.show()

    cdf = np.array([np.sum(histogram[:i]) for i in range(0,len(histogram))]) #
cumulative distribution function

    # values were crossing 255, and to map it with rgb image normalization is
 required
    cdf = ((cdf - np.min(cdf)) * 255) / (np.max(cdf) - np.min(cdf))

    img_out = np.zeros(img_in.shape)

    for h in range(height):
      for w in range(width):
        for c in range(channel):
          img_out[h,w,c] = cdf[img_in[h,w,c]] # mapping cdf to rgb image i.e.
 equalising histogram
    img_out = img_out.astype(np.uint8)   # output is here

    # finding again histogram to compare
    histogram = np.zeros(256)
    grayscale_img = cv2.cvtColor(img_out, cv2.COLOR_BGR2GRAY)
    height, width, channel = img_in.shape
    for h in range(height):
      for w in range(width):
        histogram[grayscale_img[h,w]] += 1

    print("histogram after equalization using grayscale")
    plt.plot(histogram)   # histogram after equalization
    plt.show()

    return True, img_out

# Read in input images
img_equal = cv2.imread('SourceImages/hist_equal.jpg', cv2.IMREAD_COLOR)

# Histogram equalization
succeed, output_image = histogram_equalization(img_equal)


# Plot results
```

```
fig = plt.figure(figsize=(20, 15))
plt.subplot(1, 2, 1)
plt.imshow(img_equal[..., ::-1])
plt.title('original image')
plt.axis("off")


# Plot results
plt.subplot(1, 2, 2)
plt.imshow(output_image[..., ::-1])
plt.title('histogram equal result')
plt.axis("off")


# Write out results
cv2.imwrite("Results/P4_01.jpg", output_image)
```

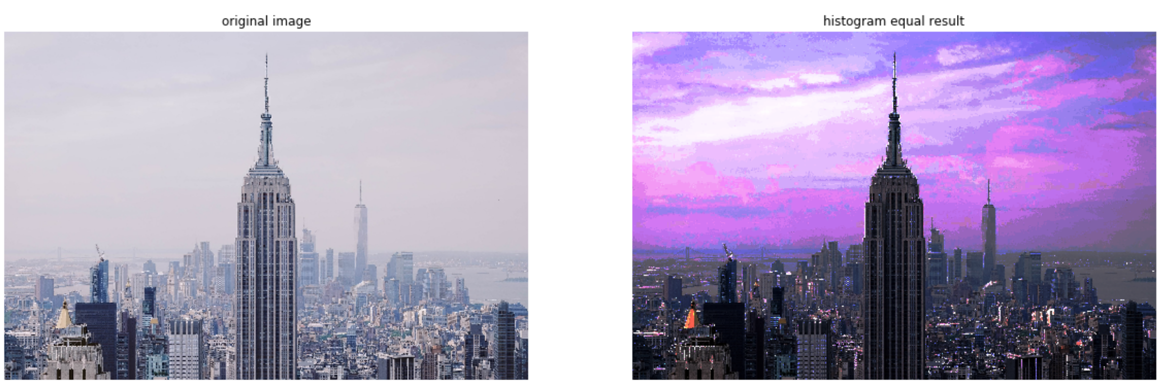
histogram before equalization using grayscale




histogram after equalization using grayscale




Out[11]: True



- **Problem 5 Low and high pass filters {20 pts}:** Start with the following tutorials: \
http://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html
(http://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html)
http://docs.opencv.org/2.4/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html
(http://docs.opencv.org/2.4/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html)
For your LPF (low pass filter), mask a 60x60 window of the center of the FT (Fourier Transform) image (the low frequencies). For the HPF, mask a 20x20 window excluding the center. The filtered low and high pass images will be the two outputs from this part and automatically saved to the Results folder.

In [12]:
```
def low_pass_filter(img_in):

    # Write low pass filter here
    # Fill in your code here

    dft = np.fft.fft2(img_in)  # discreat fourier transform
    dft_shift = np.fft.fftshift(dft)

    # mask
    # Fill in your code here
    height, width = img_in.shape
    half_height, half_column = height//2 , width//2
    mask = np.zeros((height, width),np.uint8)
    mask[half_height-30:half_height+30, half_column-30:half_column+30] = 1 # v
alue 1 at the centre

    # apply mask and inverse DFT
    # Fill in your code here
    fshift = dft_shift*mask
    f_ishift = np.fft.ifftshift(fshift) # reverse shift
    img_out = np.fft.ifft2(f_ishift) # inverse DFT
    img_out = np.abs(img_out) # absolute value

    return True, img_out.astype(np.uint8)


def high_pass_filter(img_in):

    # Write high pass filter here
    # Fill in your code here

    dft = np.fft.fft2(img_in)
    dft_shift = np.fft.fftshift(dft)

    # mask
    # Fill in your code here
    height, width = img_in.shape
    half_height, half_column = height//2 , width//2
    mask = np.ones((height, width)).astype(np.uint8)
    mask[half_height-10:half_height+10, half_column-10:half_column+10] = 0 # v
alue 0 at the centre

    # apply mask and inverse DFT
    # Fill in your code here
    fshift = dft_shift*mask
    f_ishift = np.fft.ifftshift(fshift) # reverse shift
    img_out = np.fft.ifft2(f_ishift) # inverse DFT
    img_out = np.abs(img_out) # absolute value

    return True, img_out.astype(np.uint8)

# Read in input images
img_filter = cv2.imread('SourceImages/Einstein.jpg', 0)

# Low and high pass filter
succeed1, output_low_pass_image1  = low_pass_filter(img_filter)
```

```python
succeed2, output_high_pass_image2 = high_pass_filter(img_filter)

# Plot results
fig = plt.figure(figsize=(18, 6))
plt.subplot(1, 3, 1)
plt.imshow(img_filter, 'gray')
plt.title('original image')
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(output_low_pass_image1, 'gray')
plt.title('low pass result')
plt.axis("off")

plt.subplot(1, 3, 3)
plt.imshow(output_high_pass_image2, 'gray')
plt.title('high pass result')
plt.axis("off")

# Write out results
cv2.imwrite("Results/P5_01.jpg", output_low_pass_image1)
cv2.imwrite("Results/P5_02.jpg", output_high_pass_image2)
```

Out[12]: True


In [0]: