Part I Convolution

In the first part, you are required to implement a function that performs a 2D convolution on an image.

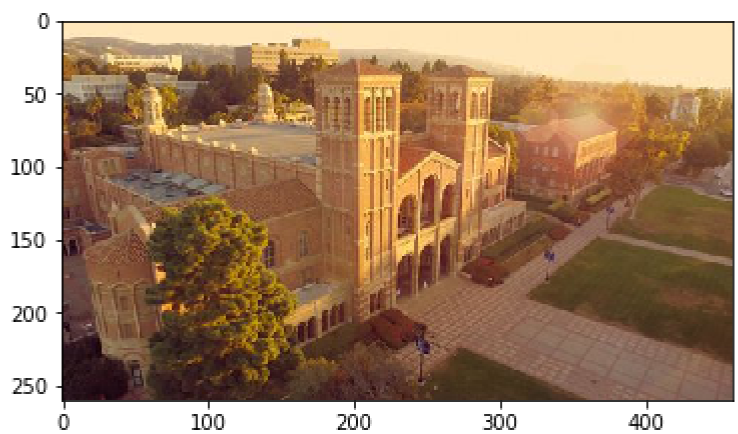
In [22]:
```
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import math
```

Please load example1.jpg.

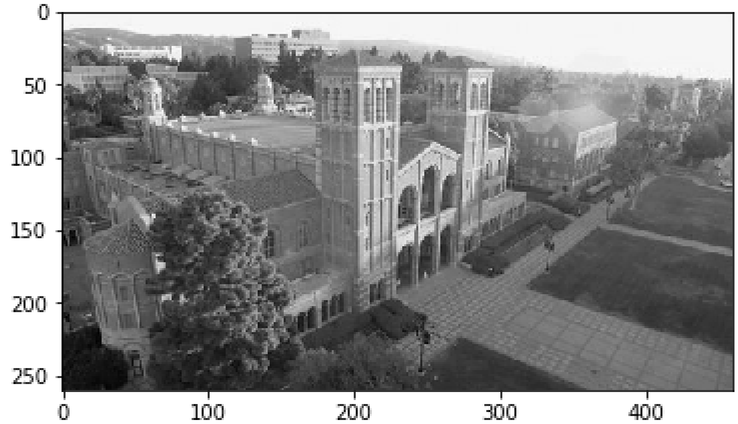
In [23]:
```
# if you are using local jupyter notebook, please use the below codes to load ima
img = Image.open('example1.jpg')
```

In [24]:
```
# Show the image
h,w,_ = np.shape(img)
print('height:',h,' width: ',w)
plt.figure()
plt.imshow(img)
plt.show()
```

```
height: 260  width:  460
```



In [25]:
```
# now convert the RGB image into the gray image for further process
gray_image = np.array(img.convert('L'))
plt.figure()
plt.imshow(gray_image, cmap='gray')
plt.show()
```



Question 1

Now you shoud implement your 2d convolution function.

The output image should have the same shape as the input.

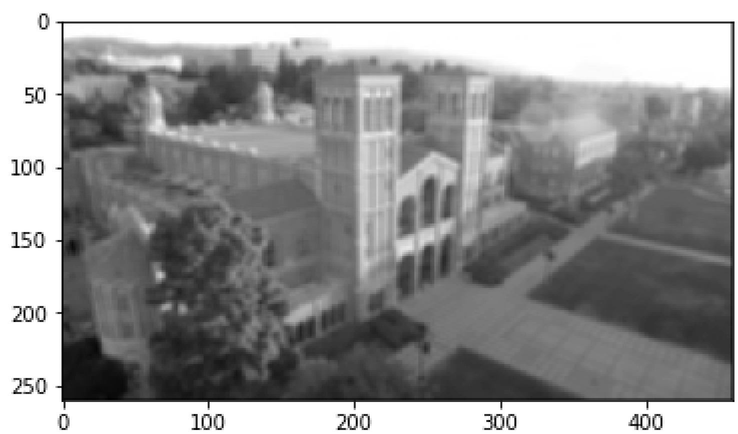
For border strategy, you will assume that value of the pixels falling outside the input image is 0.

In [26]:

```
def convolution_2d(image, filter):
  stride=1
  h, w = np.shape(image)
  f_h, f_w = np.shape(filter)
  pad=(f_h-1)//2

  if f_h==1:
    pad=(f_w-1)//2
    output = np.zeros([h, w])
    max1=np.zeros([h,w+2*pad])
    row,col=np.shape(max1)
    for i in range(h):
      for j in range(pad,w+pad):
        max1[i,j]=image[i,j-pad]
    for y in range(h):
      for x in range(w):
        window = max1[y:y+1,x*stride:x*stride+f_w]
        output[y,x] = np.sum(np.multiply(filter, window))
    return output

  if f_w==1:
    pad=(f_h-1)//2
    output = np.zeros([h, w])
    max1=np.zeros([h+2*pad,w])
    row,col=np.shape(max1)
    for i in range(pad,h+pad):
      for j in range(w):
        max1[i,j]=image[i-pad,j]
    for y in range(h):
      for x in range(w):
        window = max1[y*stride:y*stride+f_h,x:x+1]
        output[y,x] = np.sum(np.multiply(filter, window))
    return output

  output = np.zeros([h, w])
  max1=np.zeros([h+2*pad,w+2*pad])
  row,col=np.shape(max1)
#   print('height:',row,' width: ',col)
  for i in range(pad,h+pad):
    for j in range(pad,w+pad):
      max1[i,j]=image[i-pad,j-pad]
#   print('height:',pad,' width: ',h+pad)
  for y in range(row-f_h+1):
    for x in range(col-f_w+1):
      window = max1[y*stride:y*stride+f_h, x*stride:x*stride+f_w]
      output[y,x] = np.sum(np.multiply(filter, window))
  return output
```

Now we will check the result after a gaussian filter. If you write the 2d convolution correctly, you will find the image become vague

In [27]:
```
# Gaussian filter
def get_gaussian(filter_size, sigma):
  # return a gaussian filter with a size of filter_size and parameter sigma
    dim_n = len(filter_size)
    gaussian_weight = np.zeros(shape=filter_size)
    if dim_n == 2:
        dim_x, dim_y = filter_size
        center_x = dim_x/2
        center_y = dim_y/2
        for id_x in range(dim_x):
            for id_y in range(dim_y):
                weight = (id_x-center_x) ** 2 + (id_y-center_y)**2
                gaussian_weight[id_x, id_y] = math.exp(-weight/(2*sigma**2))/(ma
    return gaussian_weight

gaussian_filter = get_gaussian([5, 5], 4)
output_gaussian = convolution_2d(gray_image, gaussian_filter)
plt.figure()
plt.imshow(output_gaussian, cmap='gray')
plt.show()
```



Compare your result with the output from the convolution function provided by scipy. You will get full credit if your output shape is right and the mse error is smaller than 1e-5

In [28]:
```
from scipy import signal

# check the shape of input and output
shape_check = np.shape(output_gaussian) == np.shape(gray_image)
if shape_check:
  print('The shape of the convolution output is the same as input')
else:
  print('The shape of the convolution output and input do not match, please checl

# check the mse error
output_standard = signal.convolve2d(gray_image, gaussian_filter, mode='same', bo
mse = np.mean(((output_standard - output_gaussian)/255)**2)
print('The mse error is:', mse)
```

```
The shape of the convolution output is the same as input
The mse error is: 9.719184789213436e-05
```

Bonus

Write a function that can determine whether a square filter provided as input is separable. If the filter is separable, the function returns the two 1D vectors of the decomposed 2D filter, like [vector_h, vecor_v]. If not, the function returns None

Hint: You can use numpy.linalg.matrix_rank() to determine the rank of a matrix.

Question2: Separate a filter

In [29]:
```
# the filter is a 2D square matrix
# if the filter is separable, it will return two 1d vectors in a form [vector_h,
# if not, it will return None
def separate_filter(filter):
  '''
  Write your own code here
  '''
  if np.linalg.matrix_rank(filter) != 1:
    return None
  vector2 = [filter[0]]
  vector = [[1]]
  for element in range(1, len(filter)):
    vector.append([filter[element][0]/filter[0][0]])
  return [vector,vector2]
```

Now we will check the your code with a random image and a random filter. You will receive full credit if your mse error is less than 1e-5 .

In [30]:
```
import time

# generate random filter
v = np.random.randint(1, 10, size=[5, 1])
h = np.random.randint(1, 10, size=[1, 5])
# print(h)
# print(v)
random_filter = h*v
filter=random_filter
# generate a random image
random_image = np.random.randint(0, 255, size=[1000, 1000])

# original_convolution
start_time = time.time()
output_2d = convolution_2d(np.array(random_image), filter)
end_time = time.time()
time_1 = end_time - start_time
print('execution time for 2D convolution is: %.3f s'%(time_1))


# convolution with seperable filters
separate_result = separate_filter(filter)
if not separate_result:
  print('The filter is not separable')
else:
  start_time = time.time()
  filter_h, filter_v = separate_result
  output_1d = convolution_2d(np.array(random_image), filter_h)
  output_1d = convolution_2d(np.array(output_1d), filter_v)
  end_time = time.time()
  time_2 = end_time - start_time
  print('execution time for two 1D convolution is: %.3f s'%(time_2))

  # calculate the mse loss beblow, if your loss is less than 1e-3, meaning that t
  err = np.mean(np.power(output_2d - output_1d, 2))
  print('The mse loss for the results is: %.6f' % err)
```

```
execution time for 2D convolution is: 7.297 s
execution time for two 1D convolution is: 18.689 s
The mse loss for the results is: 0.000000
```

Question3: Which method runs faster? Give a brief explanation about your result.

when the size of filter is small, the 2d square filter is faster because it has less overhead. when the size of filter is large, the two 1d filter are faster, because it reduce the total complexity from (M^2)wh to 2Mwh.