**Practical lesson №5. Filtering image. Box-filtering**

**Task №1.**

Read the image parrot.png (in grayscale) and implement Box filtering. You do not need add rows and columns in image (as a result its shape will decrease).   
Take the 11\*11 filter kernel.

Display the result and execution time.

Hint:   
It is a time-consuming operation. So for tracking (it means that our process is been performing and not stopped) the calculation process we will use module tqdm. Example of using tqdm is below.

**import** **numpy** **as** **np**

**from** **tqdm** **import** tqdm

n\_trials = 50000 *# Number of iterations.*

pbar = tqdm(total=n\_trials) *# Initialize pbar*

arr\_1 = np.random.rand(500,500)

arr\_2 = np.random.rand(500,500)

**for** it **in** np.arange(50000):

arr\_1 \* arr\_2;

pbar.update() *# update pbar on each iteration*

pbar.close()

**import** **numpy** **as** **np**

**from** **time** **import** time

arr\_1 = np.random.rand(500,500)

arr\_2 = np.random.rand(500,500)

start\_time = time() *# initializing time*

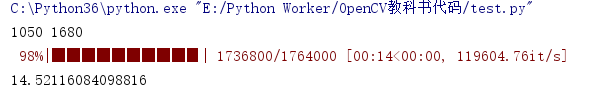
**for** it **in** np.arange(50000):

arr\_1 \* arr\_2;

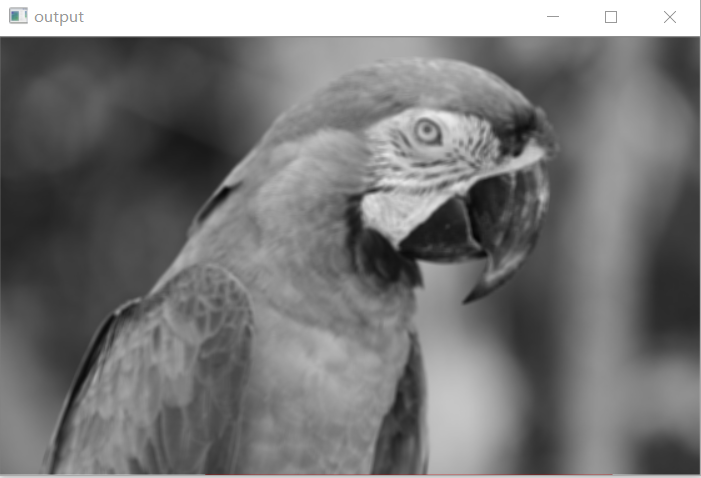
print(time() - start\_time) *# calculate execution time*

**import** cv2 **as** cv  
**import** numpy **as** np  
**from** scipy.signal **import** correlate2d, convolve2d  
**import** matplotlib.pyplot **as** plt  
**from** tqdm **import** tqdm  
**from** time **import** time  
src=cv.imread(**"D:/picture1/parrot.png"**,cv.IMREAD\_GRAYSCALE)  
w,h=src.shape[:2]  
print(w,h)  
n\_trials = h\*w *# Number of iterations.*pbar = tqdm(total=n\_trials) *# Initialize pbar*start\_time = time() *# initializing time*k = 11  
kernel = 1/k\*\*2 \* np.ones((k,k))  
new\_img=np.zeros((w-k+1,h-k+1),dtype=np.float32)*#新图片的尺寸***for** i **in** range(k//2, w-k//2):  
 **for** j **in** range(k//2, h-k//2):  
 new\_img[i-k//2,j-k//2]=np.sum(src[i-k//2:i+k//2 + 1,j-k//2:j+k//2 +1]\* kernel)  
 pbar.update() *# update pbar on each iteration*pbar.close()  
print(time() - start\_time) *# calculate execution time*new\_img=(new\_img.astype(np.uint8))

**result:**







**Task №2.**

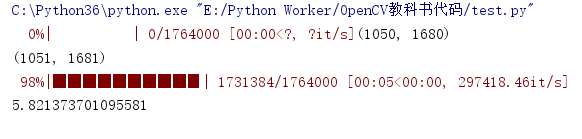
Read the image from the parrot file.png (in grayscale) and implement Box filtering (21\*21kernel) using integral image:

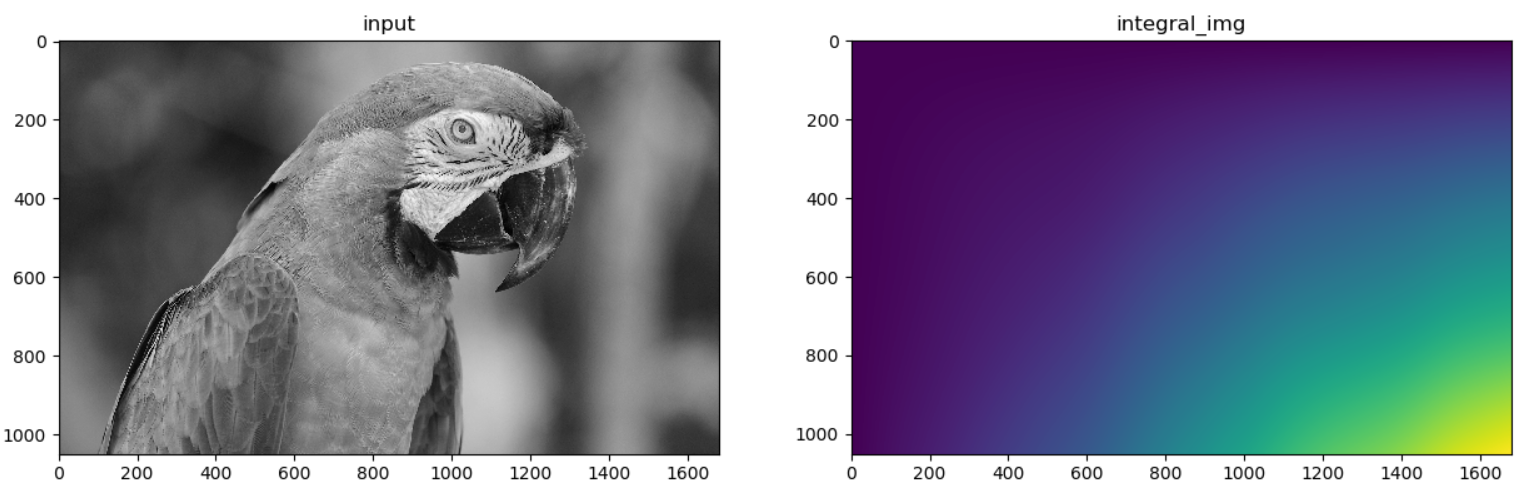
1. Get the integral image (skimage.transform.integral\_image).
2. Implement Box filtering using integral image.

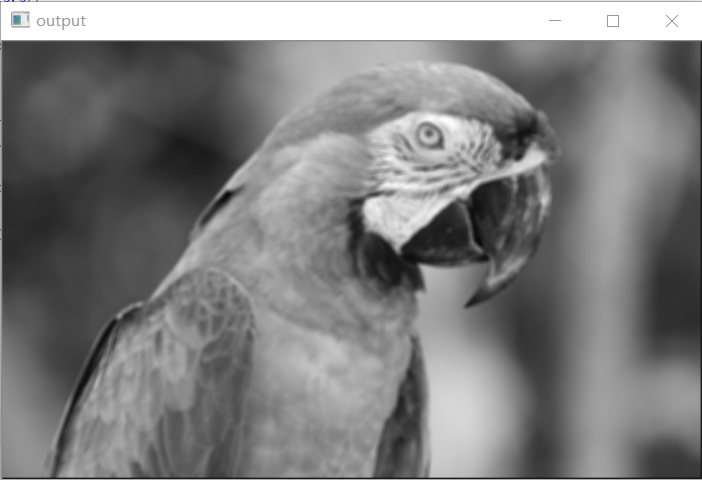
Display the result and execution time.

**import** cv2 **as** cv  
**import** numpy **as** np  
**from** skimage.transform **import** integral\_image, resize  
**import** matplotlib.pyplot **as** plt  
**from** tqdm **import** tqdm  
**from** time **import** time  
src=cv.imread(**"D:/picture1/parrot.png"**,cv.IMREAD\_GRAYSCALE)  
h,w=src.shape[:2]  
n\_trials = h\*w *# Number of iterations.*pbar = tqdm(total=n\_trials) *# Initialize pbar*start\_time = time() *# initializing time*integral\_img=integral\_image(src)  
*#####integral\_show=integral\_img.astype(np.uint8)*print(integral\_img.shape[:2])  
rows = integral\_img.shape[0]  
integral\_img = np.hstack((np.zeros((rows, 1), dtype=float), integral\_img))  
columns = integral\_img.shape[1]  
integral\_img = np.vstack((np.zeros((1, columns), dtype=float), integral\_img))  
print(integral\_img.shape[:2])  
k = 11  
new\_img=np.zeros((h-k+1,w-k+1),dtype=np.float32)*#create new image for this task.***for** i **in** range(k+1, h):  
 **for** j **in** range(k+1, w):  
 A=integral\_img[i,j]  
 B=integral\_img[i-k-1,j-k-1]  
 C=integral\_img[i,j-k-1]  
 D=integral\_img[i-k-1,j]  
 new\_img[i-k-1,j-k-1]=(A-C-D+B)/(k\*\*2)  
 pbar.update() *# update pbar on each iteration*pbar.close()  
print(time() - start\_time) *# calculate execution time*new\_img=new\_img/new\_img.max()  
plt.figure(figsize=(15,5))  
plt.subplot(121)  
plt.imshow(src,cmap=plt.cm.gray)  
plt.title(**'input'**)  
plt.subplot(122)  
plt.imshow(integral\_img)  
plt.title(**'integral\_img'**) *# 图像题目*plt.show()  
cv.imshow(**"output"**,cv.resize(new\_img,(w//3,h//3),interpolation=cv.INTER\_LINEAR))  
cv.waitKey(0)  
cv.destroyAllWindows()

result：







Evaluation: Two methods for processing images yield similar results, but method 2 of processing images faster.

**import** cv2 **as** cv  
**import** numpy **as** np  
**from** skimage.transform **import** integral\_image, resize  
**import** matplotlib.pyplot **as** plt  
**from** tqdm **import** tqdm  
**import** time  
src=cv.imread(**"D:/picture1/parrot.png"**,cv.IMREAD\_GRAYSCALE)  
h1,w1=src.shape[:2]  
src=cv.resize(src,(h1//2,w1//2),interpolation=cv.INTER\_LINEAR)  
h,w=src.shape[:2]  
print(h,w)  
integral\_img=integral\_image(src)  
print(integral\_img.shape[:2])  
rows = integral\_img.shape[0]  
integral\_img = np.hstack((np.zeros((rows, 1), dtype=float), integral\_img))  
columns = integral\_img.shape[1]  
integral\_img = np.vstack((np.zeros((1, columns), dtype=float), integral\_img))  
print(integral\_img.shape[:2])  
y1=[]  
y2=[]  
*#x = np.uint8(np.linspace(11,51,5))*x=[3,13,23,33,43]  
pbar = tqdm(total=len(x))  
**def** method1(k):  
 start=time.time()  
 kernel = 1 / k \*\* 2 \* np.ones((k, k))  
 new\_img1 = np.zeros((h - k + 1, w - k + 1), dtype=float) *# 新图片的尺寸* **for** i **in** range(k // 2, h - k // 2):  
 **for** j **in** range(k // 2, w - k // 2):  
 new\_img1[i - k // 2, j - k // 2] = np.sum(src[i - k // 2:i + k // 2 + 1, j - k // 2:j + k // 2 + 1] \* kernel)  
 end=time.time()  
 **return** end-start  
**def** method2(k):  
 start = time.clock()  
 new\_img2=np.zeros((h-k+1,w-k+1),dtype=float)*#create new image for this task.* **for** i **in** range(k-1, h):  
 **for** j **in** range(k-1, w):  
 A=integral\_img[i,j]  
 B=integral\_img[i-k,j-k]  
 C=integral\_img[i,j-k]  
 D=integral\_img[i-k,j]  
 new\_img2[i-k+1,j-k+1]=(A-C-D+B)/(k\*\*2)  
 end=time.clock()  
 **return** end-start  
**for** k **in** x:  
 y2.append(method2(k))  
 pbar.update()  
**for** xx **in** x:  
 y1.append(method1(xx))  
 pbar.update()  
pbar.close()  
plt.figure(figsize=(15,5))  
plt.plot(x,y2,**"r"**,label=**"method 2"**)  
plt.plot(x,y1,**"g"**,label=**"method 1"**)  
plt.legend()  
plt.show()

