|  |  |  |
| --- | --- | --- |
|  |  | Image binarization  Image Processing |
| دانشجو نام  محمد شکری  شماره دانشجویی  981531027  استاد  دکتر صادق فدایی ترم بهار 1401 |  | مقدمهتعریف Image binarization با زبان ساده همان سیاه ، سفید کردن عکس است. در واقع هر پیکس ماکزیمم یا مینیمم مقدار ممکن را میگیرد کاربرد  * جداسازی پس زیمنه از محتوا * بدست آوردن اطلاعات از بخش های مختلف عکس  ساز و کارنوع عکس معمولا بر روی عکس های خاکستری انجام می گیرد.(عکس های رنگی تبدیل به حاکستری می شوند) منطق تبدیل هر پیکسل با مقداری به نام threshold مقایسه شده   * Pixel > threshold => pixel = 255 * Else => pixel = 0  دسته بندیGlobal در این روش یک مقدار کلی threshold برای تمامی پیکسل های عکس اعمال می شود Local در این روش عکس به پنجره های مختلفی شکسته شده و برای هر پنجره یک مقدار threshold پیدا شده و برای همان پنجره اعمال می شود. |
|  |  | الگوریتم های مورد بررسی |

* Constant T
* Global iterative
* Adaptive or local iterative
* Niblack
* Global OTSU

Image threshold module

یک ماژول نوشته شده توسط شخص خود برای راختی کار و همچنین clean code

بودن است که شامل توابعی است :

* getWindow(img, x, y, w ,h) : این تابع عکسی را می گیرد و با توجه به

مختصات x,y و طول پنجره w,h پیکسل های آن پنجره را به ما بر می گرداند.

یکی از فواید آن این است اگر به آخر عکس رسیده باشیم و طول یا عرض یا

هر دو پنجره از عکس بیرون بزند ، خود تابع متوجه شده و مقدار را تنظیم

می کند.

* setWindow(img, x, y, window) : این تابع پنجره ای را میگیرد و با توجه

به مختصات داده شده آن را در عکس جایگزین می کند.

* applyT(img , t) : این تابع یک عکس را می گیرد و t را که threshold

است برای تمام پیکسل ها اعمال می کند.

* aapplyTInWindow(window, t) : این تابع یک پنجره را می گیرد و t را

که threshold است برای تمام پیکسل ها ی آن پنجره اعمال می کند.

* lightOccurances(window) : این تابع یک پنجره را میگیرد و لیستی از

فراوانی پیکسل های آن پنجره بر می گیرداند.(در gray-scale از 0 تا 255

است)

* histogram(data , count) : این تابع هیستوگرام را حساب می کند.
* findT(hist, T = 128) : این تابع هیستوگرام و یک threshold اولیه را

میگیرد و threshold را محاسبه می کند. ( برای روش های global iterative

و local iterative به کار می رود)

* meanOfWindow(window) : میانگین پیکسل های پنجره را می دهد.
* stdOfWindow(window, avg) : انحراف معیار پنجره را بر می گرداند.

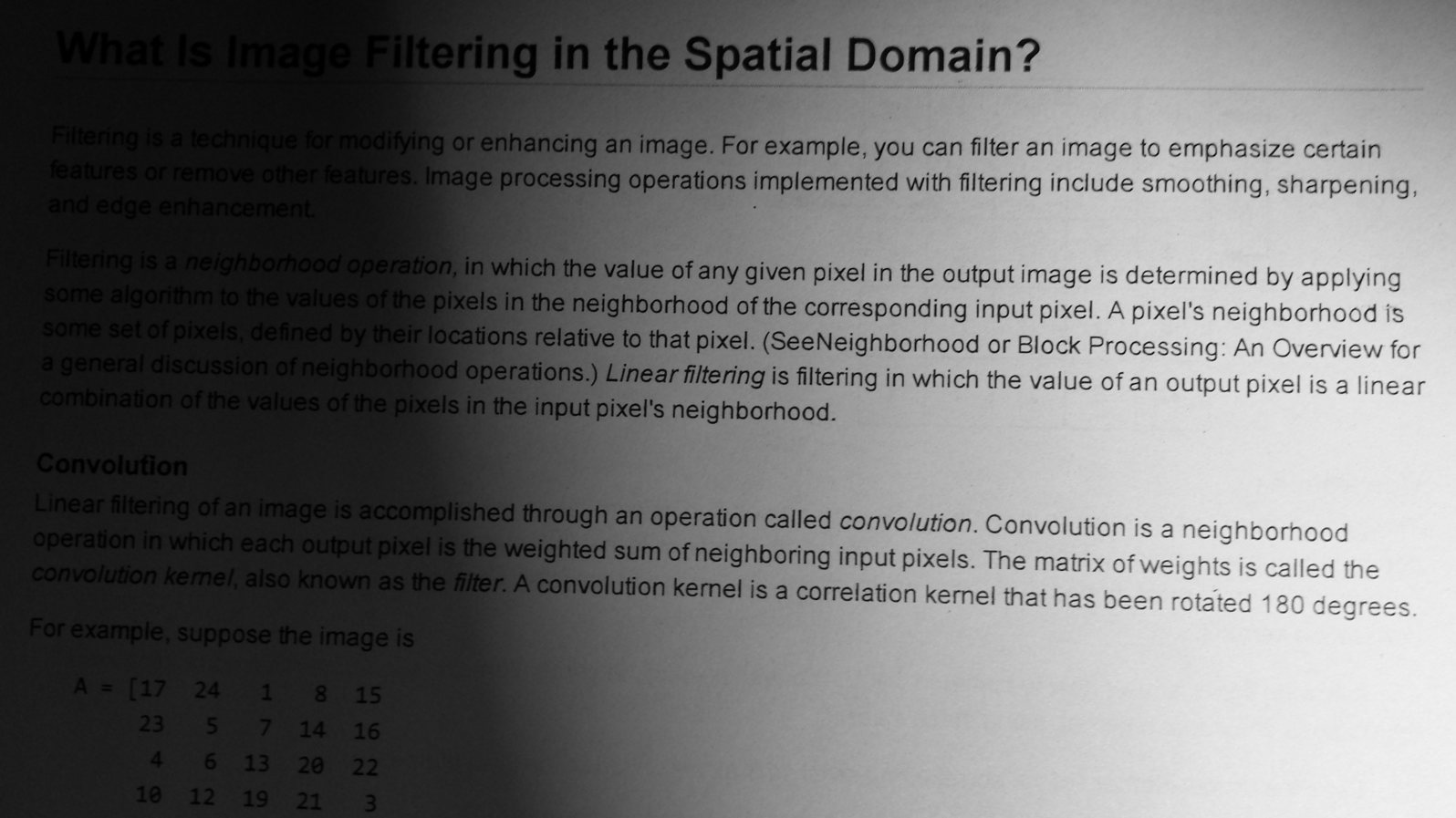
### Constant T

* در این روش به طور مثال threshold را مقدار 55 در نظر گرفته و

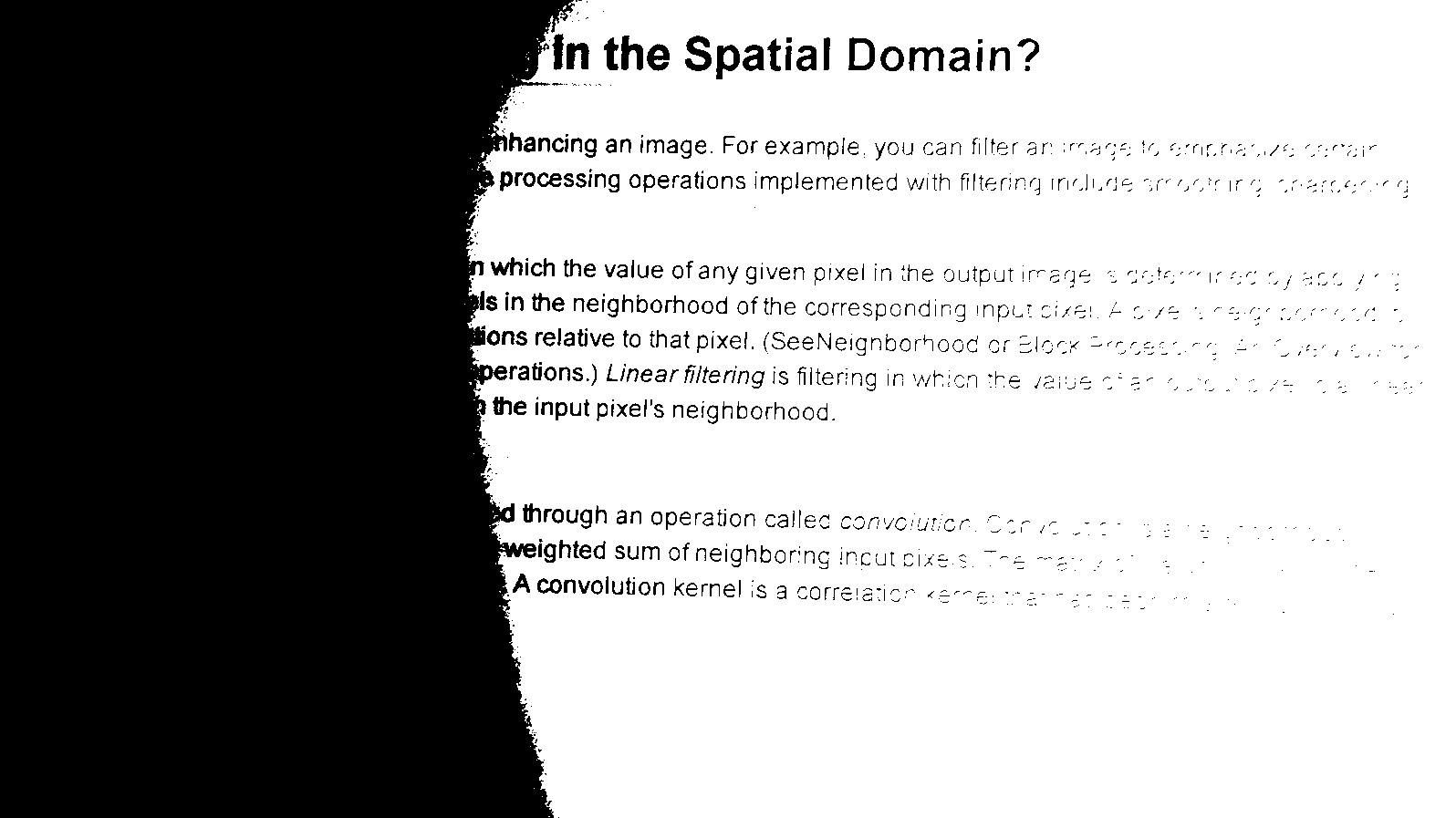
کل پیکسل ها اعمال می کنیم.

#### کد :

from ImageThreshold import \*  
  
def applyConstantT(src,t = 55):  
 img = Image.open(src).convert('L')  
 return applyT(img,t)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 applyConstantT('image1.png').save('image1\_constant\_t\_128.png')

**عکس اولیه :**

**عکس ثانویه :**



### Global iterative

* در این روش به هیستوگرام تمام پیکسل ها محاسبه می شود.
* سپس داده ها را بر اساس مقدار اولیه t ، جدا کرده و میانگین مجموع دو

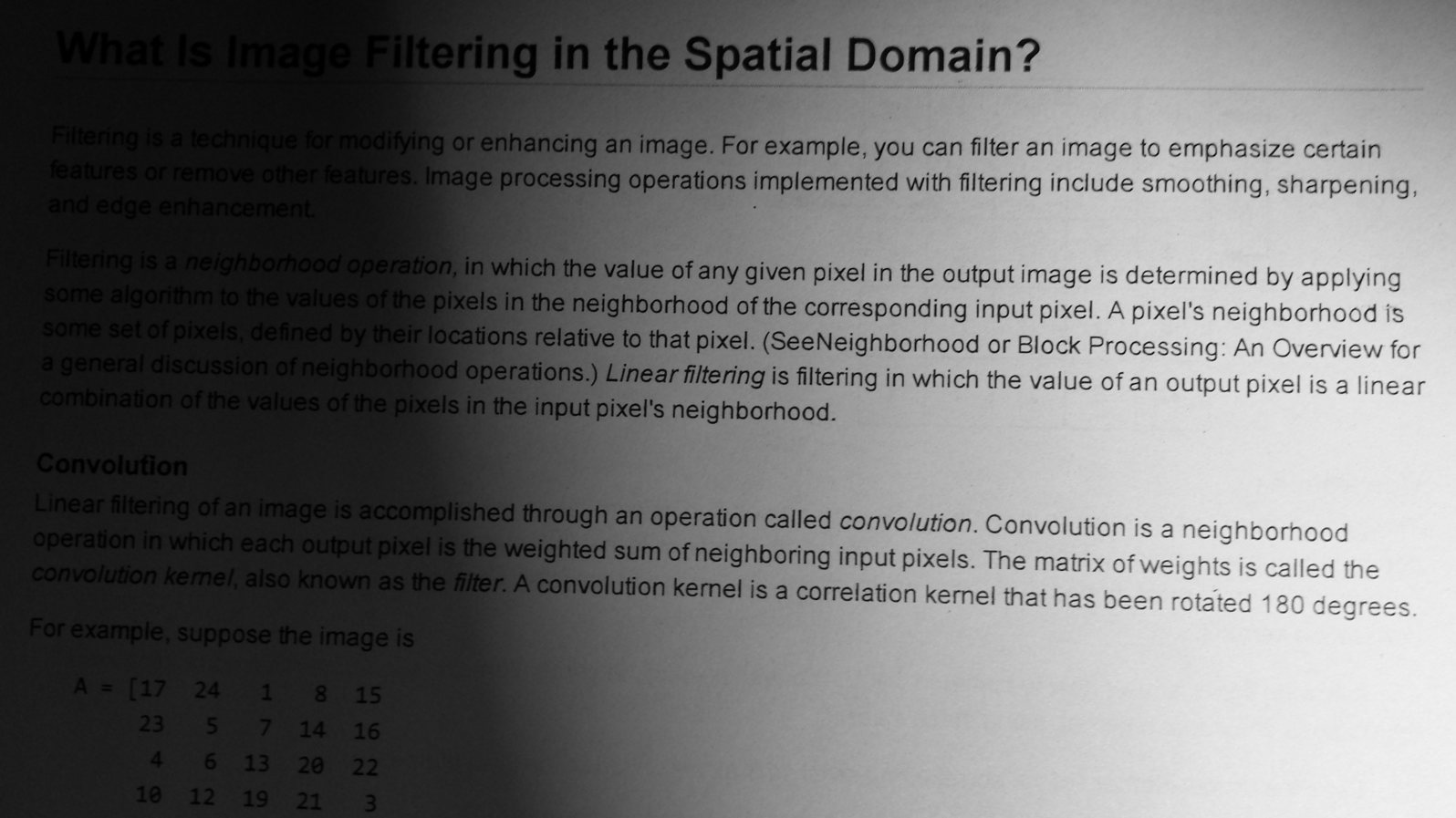
طرف را محاسبه کرده تا t2، بدست آید . حال اگر قدر مطلق تفاوت در t ها

آن قدر کم بود ( مانند 0.0000001( ، t بدست می آید در غیر این صورت

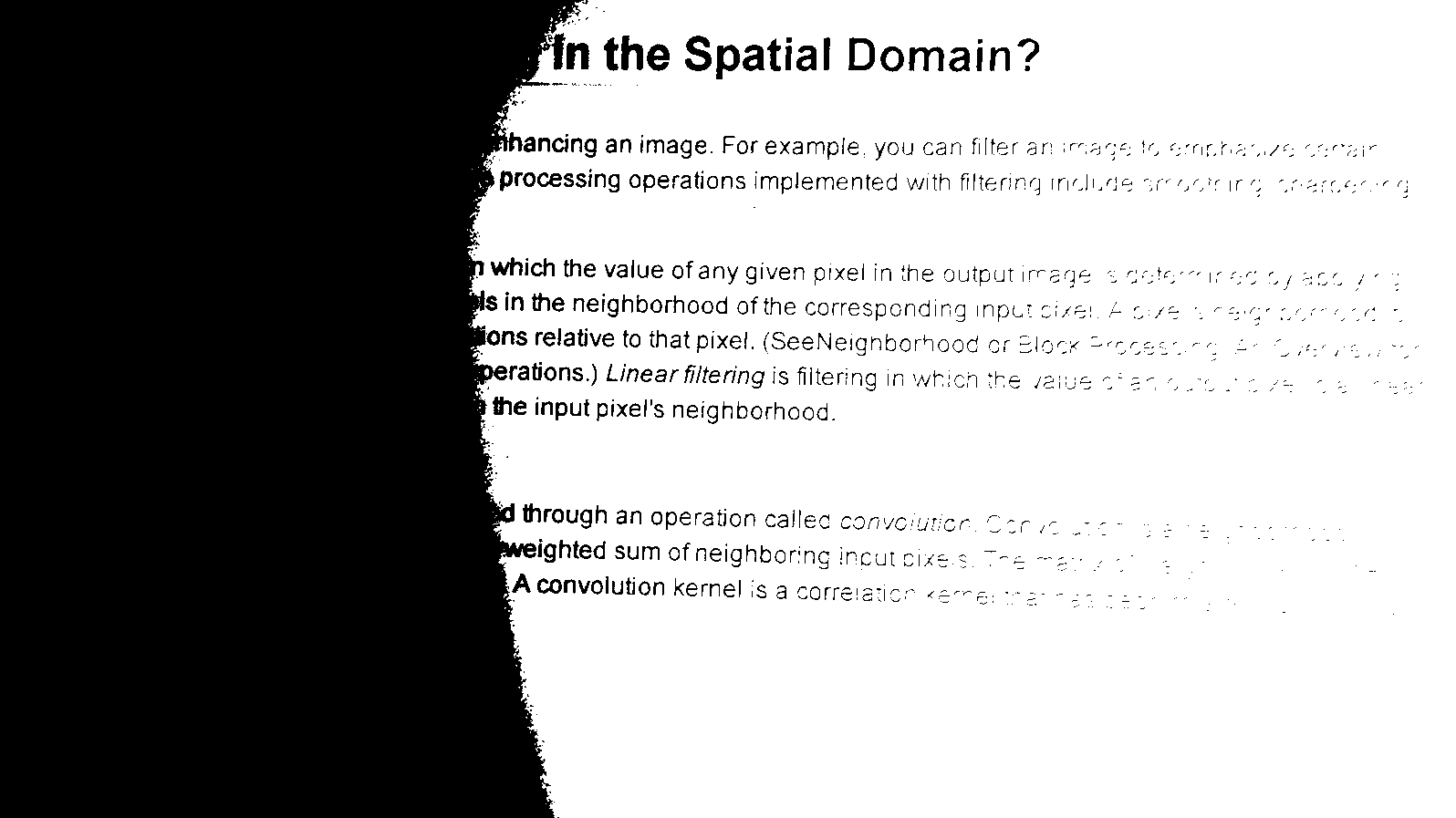
این روند را برای هر t جدید تکرار کرده.

**کد :**

from ImageThreshold import \*  
  
def applyGlobalIterative(src):  
 img = Image.open(src).convert('L')  
 width, height = img.size  
 window = getWindow(img,0,0,width,height)  
 lights = lightOccurances(window)  
 hist = histogram(lights,width \* height)  
 t = findT(hist)  
 print('t:',t)  
 return applyT(img,t)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 applyGlobalIterative('image1.png').save('image1\_global\_iterative.png')

**عکس اولیه :**

**عکس ثانویه :**



# T بدست آمده برابر با 56 است در این روش.

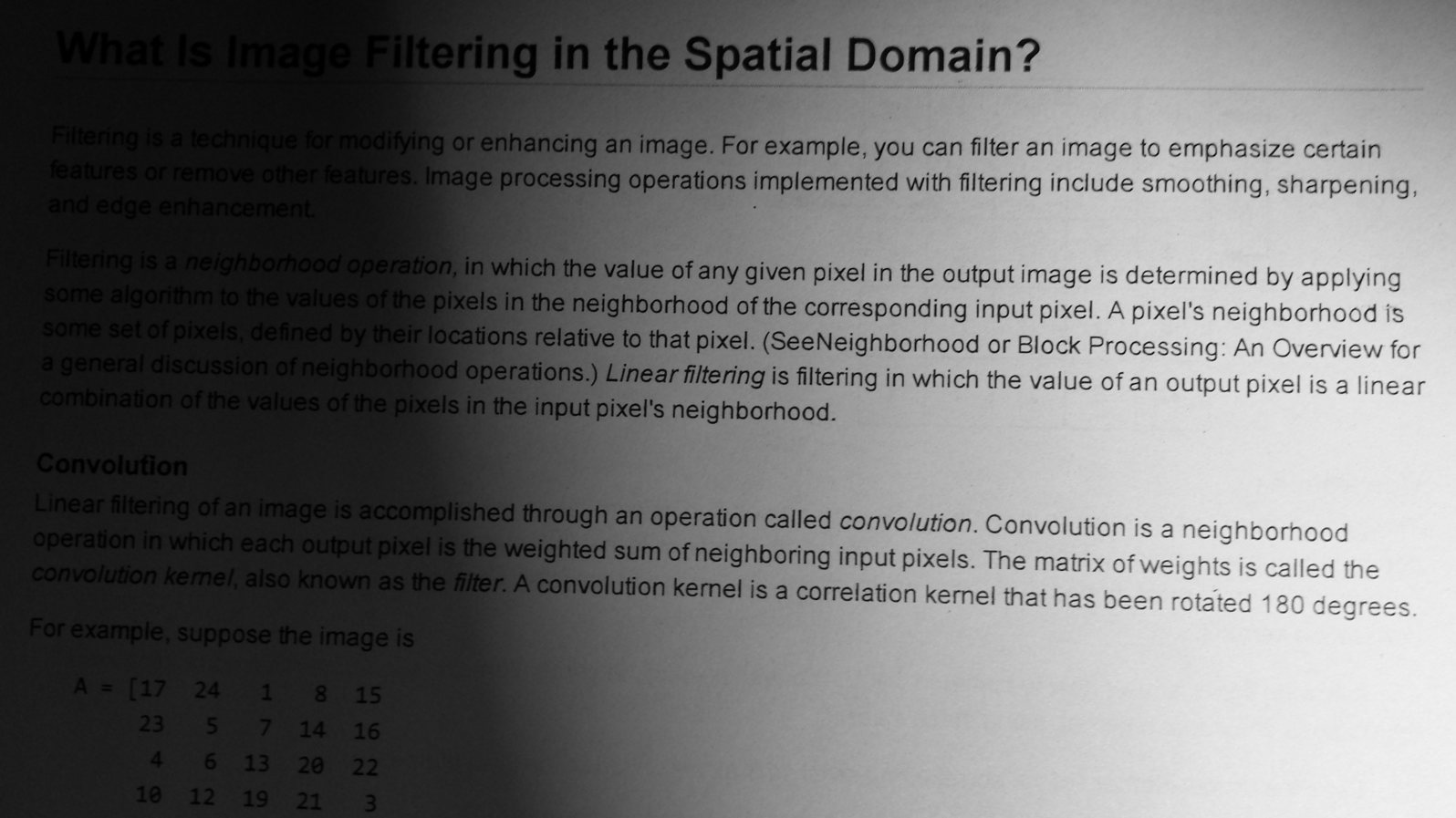
### Local iterative

* این روش مانند روشش قبل است تنها تمام عملیات ها فقط برای هر پنجره

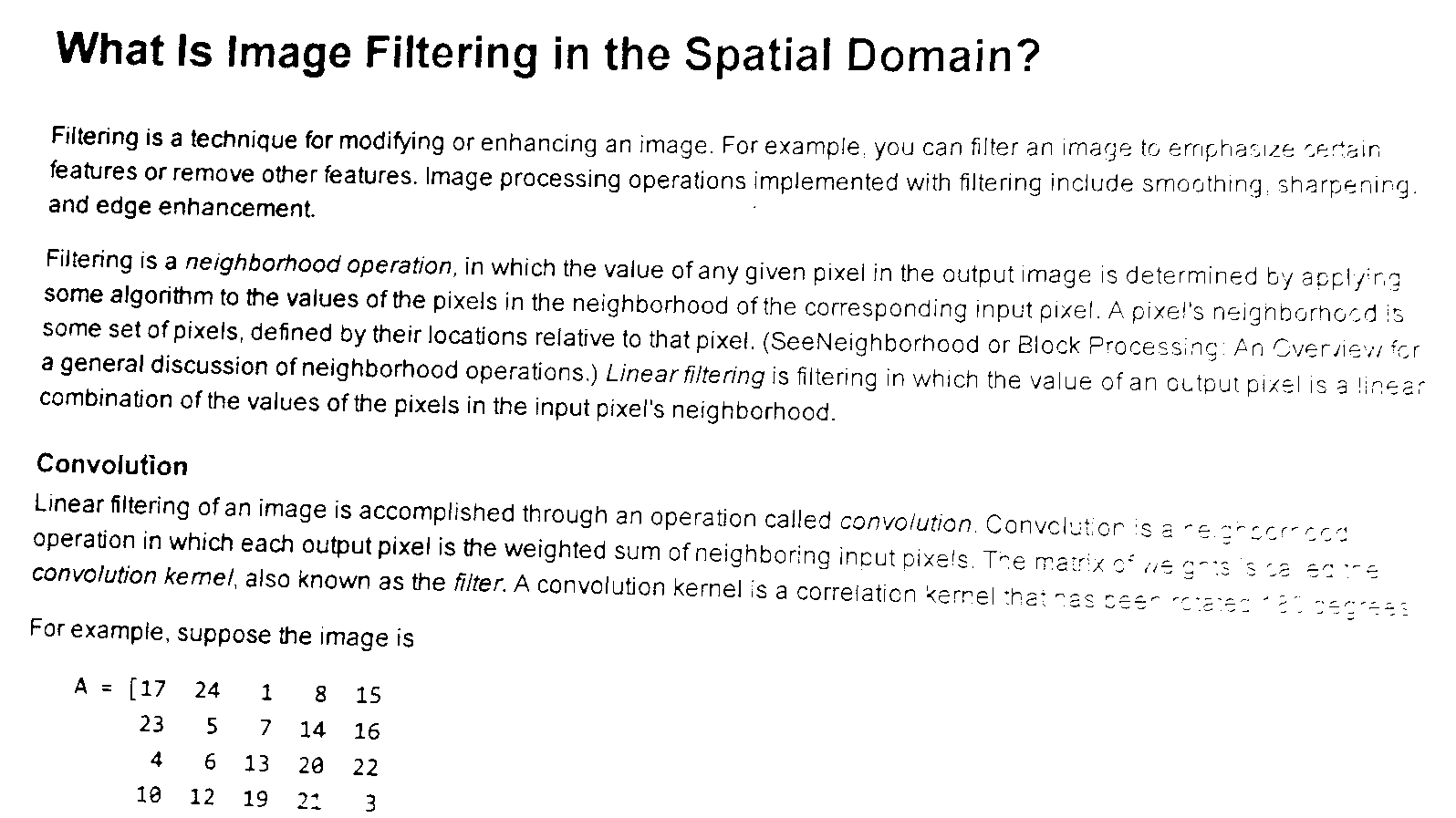
تکرار می شوند.

**کد :**

from ImageThreshold import \*  
  
def applyLocalIterative(src, windowX, windowY):  
 img = Image.open(src).convert('L').copy()  
 width, height = img.size  
  
 for i in range(0,width,windowX):  
 for j in range(0,height,windowY):  
 window = getWindow(img, i, j, windowX, windowY)  
 lights = lightOccurances(window)  
 hist = histogram(lights, windowX \* windowY)  
 t = findT(hist)  
 setWindow(img,i,j, applyTInWindow(window,t) )  
  
 return img  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 applyLocalIterative('image1.png', 25, 25).save('image1\_local\_iterative.png')

**عکس اولیه :**

**عکس ثانویه :**



### niblack

* در این روش انحراف معیار و میانگین هر پنجره محاسبه می شود سپس

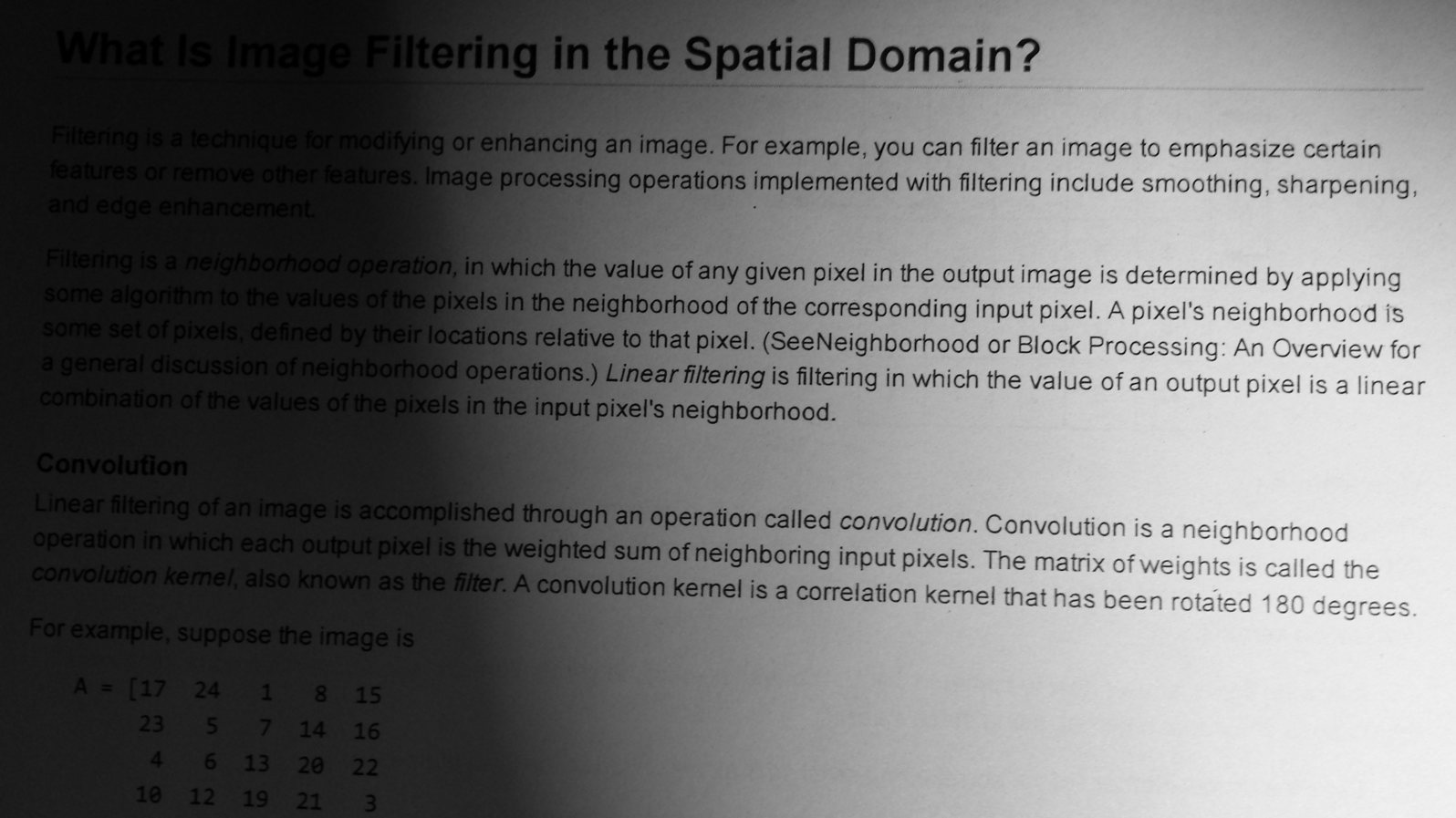
Threshold به صورت mean – k\*std محاسبه شده.در واقع std بالاتر به

معنای غیر یکنواختی بیشتر به معنای اطلاعات بیشتر است.

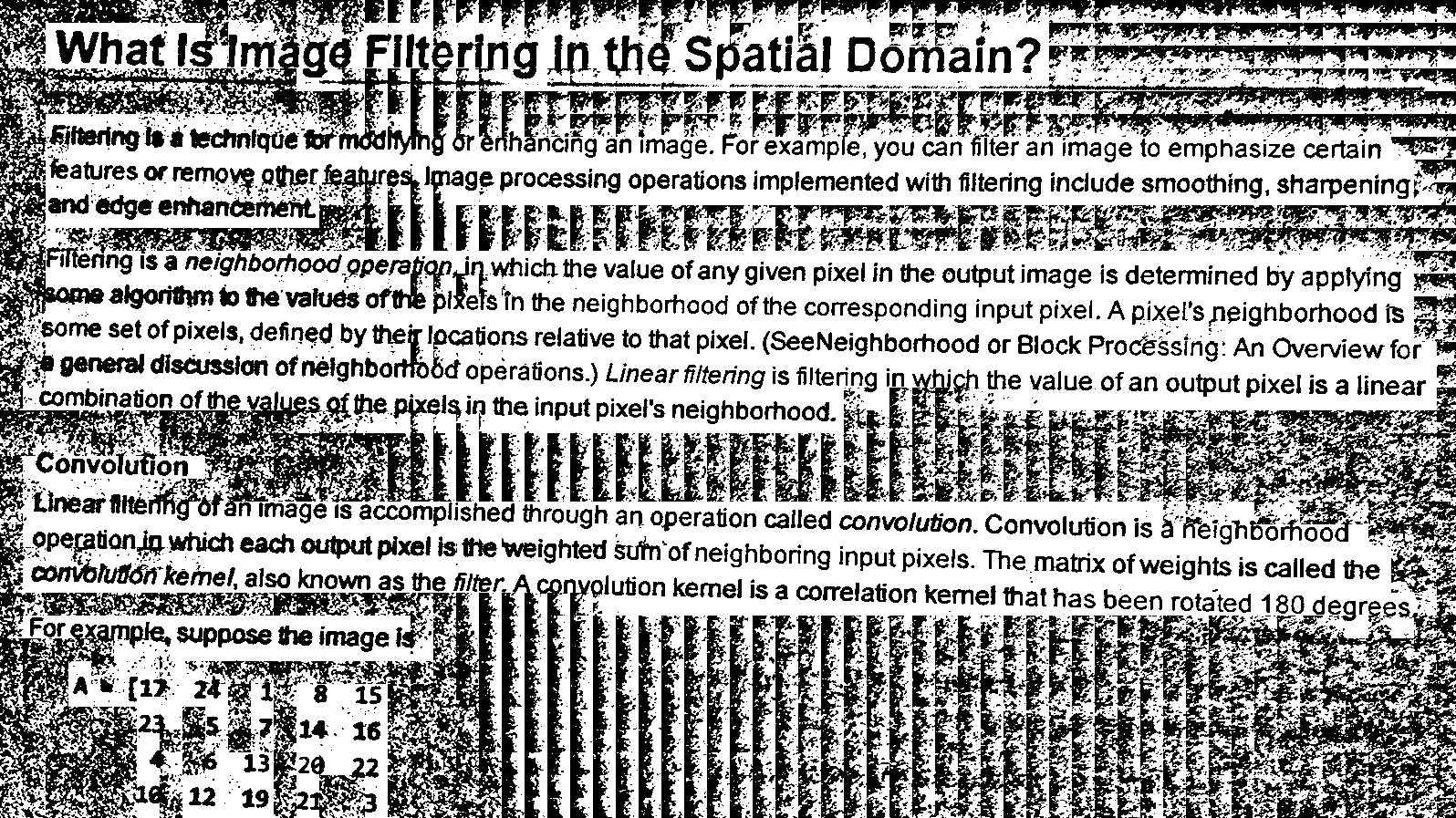
* طبق تجربه در این روش پنجره 15\*15 و k=-0.2 نتایج بهتری دارد.

**کد :**

from ImageThreshold import \*  
  
  
def applyNiblack(src,k, windowX, windowY):  
 img = Image.open(src).convert('L').copy()  
 width, height = img.size  
 for i in range(0,width,windowX):  
 print(i)  
 for j in range(0,height,windowY):  
 window = getWindow(img, i, j, windowX, windowY)  
 mean = meanOfWindow(window)  
 std = stdOfWindow(window,mean)  
 t = int(numpy.round(mean + (k \* std)))  
 window = applyTInWindow(window, t)  
 setWindow(img, i, j, window)  
 return img  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 applyNiblack('image1.png',-0.2, 25, 25).save('image1\_niblack.png')

**عکس اولیه :**

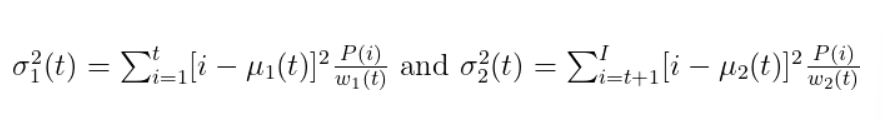
**عکس ثانویه :**



### OTSU

* این روش بر اساس واریانس است. کلید اصلی پیدا کردن threshold این است

که وزن واریانس بین پس زمینه و محتوا را مینیمم کرد.



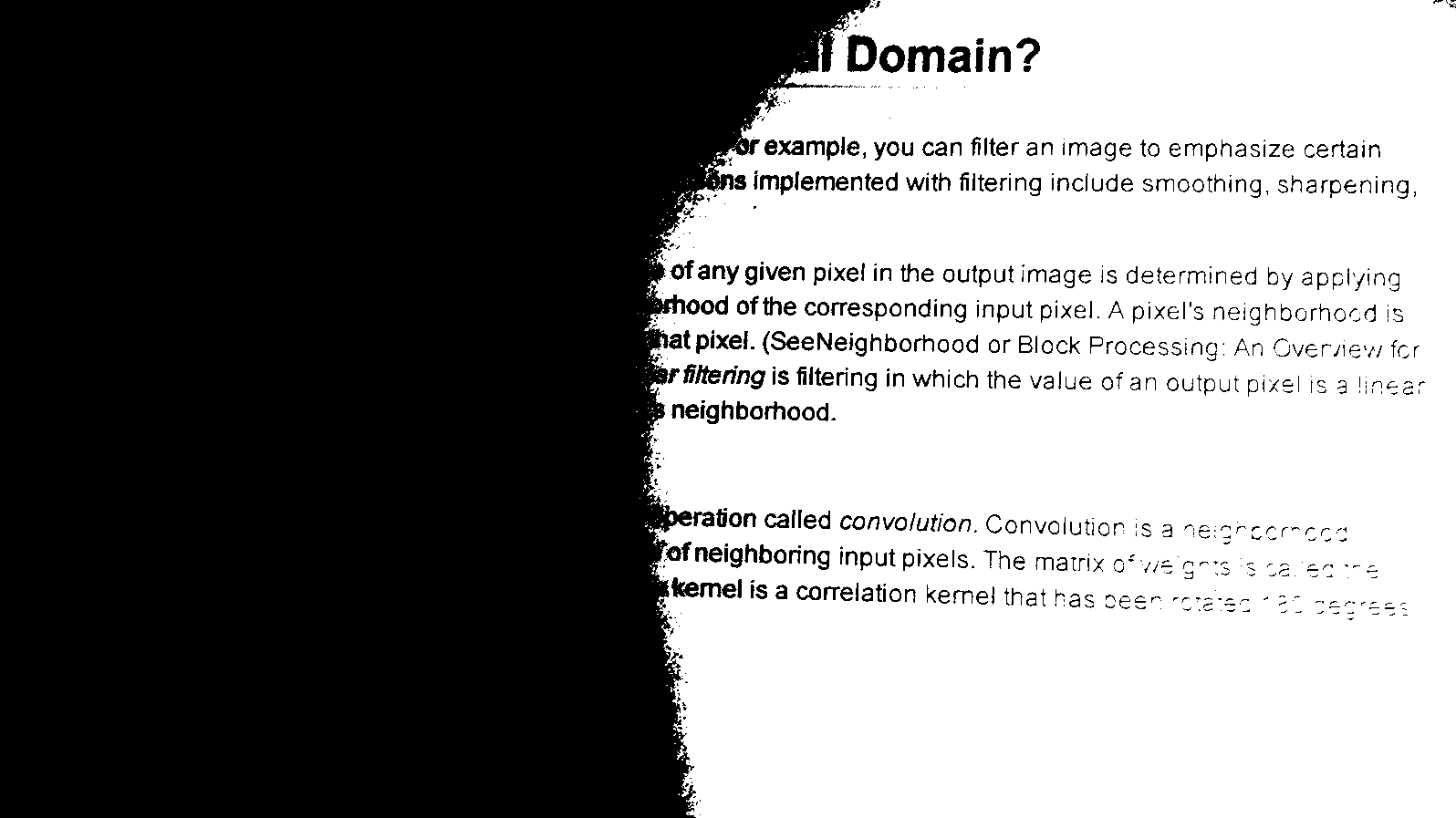
**کد :**

from ImageThreshold import \*  
  
def applyLocalIterative(src):  
 img = Image.open(src).convert('L').copy()  
 width, height = img.size  
 window = getWindow(img, 0, 0, width, height)  
 lightOccurances = [0 for i in range(256)]  
  
 for light in range(256):  
 beforeLight = []  
 afterLight = []  
 print(light)  
 for i in range(width):  
 for j in range(height):  
 pixel = window['window'][i][j]  
 if light <= pixel:  
 afterLight.append(pixel)  
 else:  
 beforeLight.append(pixel)  
 meanBefore = sum(beforeLight) / len(beforeLight) if len(beforeLight) else 0  
 meanAfter = sum(afterLight) / len(afterLight) if len(afterLight) else 0  
 lightOccurances[light] = len(beforeLight) \* len(afterLight) \* (meanBefore - meanAfter) \*\* 2  
  
 t = 0  
 for i in range(len(lightOccurances)):  
 if lightOccurances[t] < lightOccurances[i]:  
 t = i  
 print(t)  
  
 return applyT(img, t)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 applyLocalIterative('image1.png').save('image1\_otsu.png')

A picture containing text, document, letter, black and white

Description automatically generated**عکس اولیه :**

**عکس ثانویه :**



# نتیجه گیری

|  |  |  |  |
| --- | --- | --- | --- |
| # | مزایا | معایب | اساس |
| Constant | سرعت بالا | Threshold کلی عکس را به دو بخش اعمال می کند و چون global است به خوبی تمایز هارا نشان نمی دهد | انتخاب یک T و اعمال آن به کل تصویر |
| Global | نسبت به Constant T خوبی آن این است که T را خودش پیدا می کند . | Threshold کلی عکس را به دو بخش اعمال می کند و چون global است به خوبی تمایز هارا نشان نمی دهد | پیدا کردن هیستوگرام سپس پیدا کردن T optimal |
| Local | به خاطر محلی بودن و optimal بودن T بین پس زمینه و محتوا خوب تفاوت ایجاد می شود. | سرعت پایین به خاطر پنجره های زیاد و ممکن است بخشی از اطلاعات کم رنگ شود. | پیدا کردن هیستوگرام سپس پیدا کردن T optimal برای هر پنجره |
| Niblack | مناسب برای تشخیص متن | پس زمینه مانند local آن قدر یکدست مجزا نمی شود | غیر یکنواختی بر اساس میانگین و انحراف معیار محلی هر پنجره |
| OTSU | - | در نویز بالا و شی های کوچک خوب عمل نمی کند | واریانس |