

TRATAMIENTO DE SOMBRAS EN FOTOGRAFIAS DE DOCUMENTOS

```
# importamos librerias utilizadas
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage, misc
import matplotlib.pyplot as plt

# definimos una funcion para graficar las imagenes
def imprimir_pic(pic,etiqueta):
    plt.figure()
    plt.imshow(pic, cmap = 'gray')
    plt.title(etiqueta)
    plt.axis('off')

#imagen_entrada = "cejem1"
#input_pic = cv2.imread(imagen_entrada+".png")
#cv2.imwrite(imagen_entrada+".jpeg", input_pic)
```

Seleccionamos la imagen con la cual se va a trabajar

```
imagen_entrada = "input1"
input_pic = cv2.imread(imagen_entrada+".jpeg")
```

METODO BASE

Generamos la mascara o "perfil de sombra"

```
# Paquetes necesarios para la morfología matemática
from skimage.morphology import erosion, dilation, opening, closing
# Elementos estructurales
from skimage.morphology import disk, diamond, ball, rectangle, star
from scipy import ndimage as ndi
#reducimos la pic a grises
input_pic = cv2.cvtColor(input_pic, cv2.COLOR_BGR2GRAY)

#realizamos una copia de la pic recibida para poder modificarla en un
espacio de memoria diferente
mask = np.copy(input_pic)
#las siguientes 3 líneas se encargan de normalizar la pic de 0 a 255
mask = mask - mask.min()
mask = mask/mask.max()
mask = mask*255
```

#Creamos la mascara con un filtro de mediana, consideramos las letras como ruido

```
for i in range(3):  
    mask = ndimage.median_filter(mask, size=18, mode='reflect')
```

```
mask_gris = np.copy(mask)
```

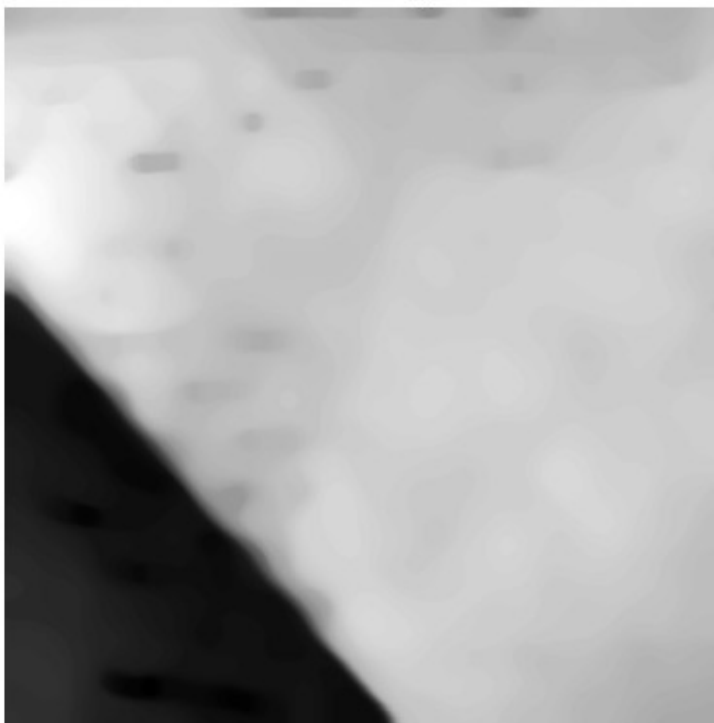
```
imprimir_pic(mask, 'Máscara en grises')
```

#la maskara debe estar en blanco y negro y no en escala de grises, asi que se realiza un if:

```
promedio = np.mean(mask)  
for (cor_y, cor_x), value in np.ndenumerate(mask):  
    if mask[cor_y, cor_x] < promedio: #Good Pixel  
        mask[cor_y, cor_x] = 0.1  
    elif mask[cor_y, cor_x] > promedio: #Bad Pixel  
        mask[cor_y, cor_x] = 254.9
```

```
imprimir_pic(mask, 'Máscara')  
imprimir_pic(input_pic, 'original')
```

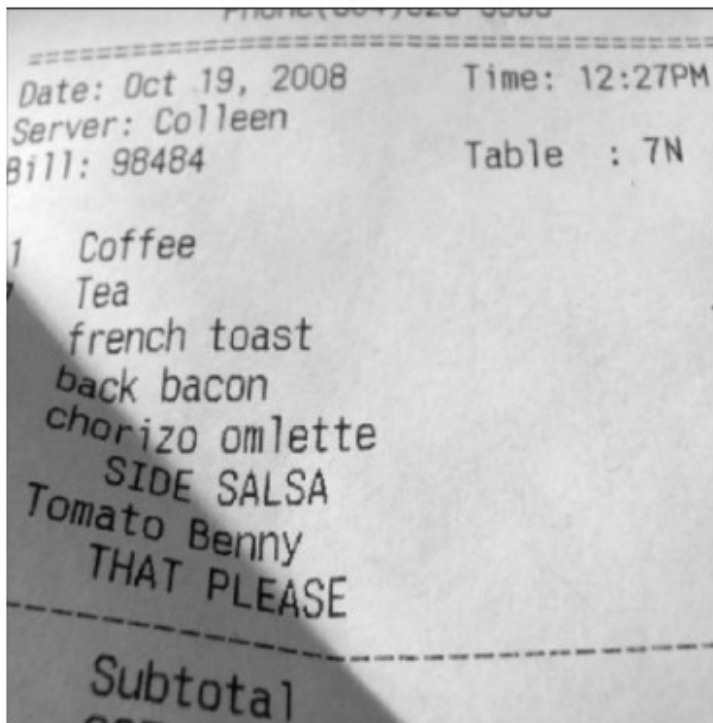
Máscara en grises



Máscara



original



Definimos la funcion de contraste que apartir de un percentil permite

```
def funcion_de_contraste(pic,percentil):  
    limite = np.percentile(pic,percentil)  
    blancos = pic >= limite  
    pic = pic + 255*blancos  
    negros_saturados = pic<limite  
    blancos_saturados = pic>=limite  
    pic[negros_saturados] = 0  
    pic[blancos_saturados] = 255  
    return pic
```

Definimos el array "Zona Oscura"

```
zona_oscura = np.full(input_pic.shape,np.nan)  
limite = np.nanmean(input_pic)  
for (coor_y,coor_x), value in np.ndenumerate(input_pic):  
    aux = mask[coor_y:coor_y+1,coor_x:coor_x+1]  
    if aux<limite: #Good Pixel  
        zona_oscura[coor_y,coor_x] = input_pic[coor_y,coor_x]  
    elif aux>limite: #Bad Pixel  
        None  
imprimir_pic(zona_oscura,"zona_oscura")
```

zona_oscura



Definimos el array "Zona Clara"

```
zona_clara = np.full(input_pic.shape,np.nan)
```

```
limite = np.nanmean(input_pic)
```

```
for (coor_y,coor_x), value in np.ndenumerate(input_pic):
```

```
    aux = mask[coor_y:coor_y+1,coor_x:coor_x+1]
```

```
    if aux>limite: #Good Pixel
```

```
        zona_clara[coor_y,coor_x] = input_pic[coor_y,coor_x]
```

```
    elif aux<=limite: #Bad Pixel
```

```
        None
```

```
imprimir_pic(zona_clara,"zona_clara")
```



Definimos la funcion que nos permite conocer el punto optimo para contrastar la imagen

def contraste_optimo_por_MSE(segmento,pic):

output = 255 - np.nan_to_num(segmento,nan=0)

base = 255 -pic

errorlist = []

for i **in** range(30):

 f = funcion_de_contraste(output,i)

 error = np.sum((base-f)**2)

 errorlist.append(error)

m = np.arange(30)

#plt.scatter(m,errorlist)

#plt.title("Error vs intensidad (por percentile)")

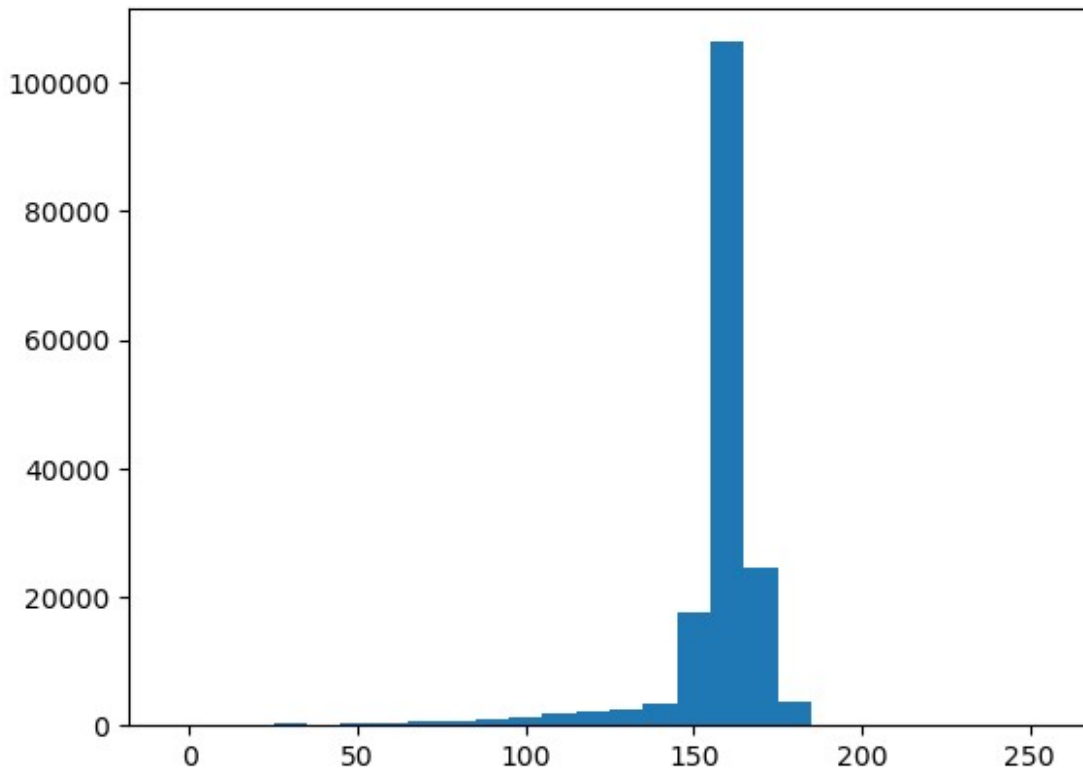
#plt.show()

return (m[np.argmin(errorlist)]+1)

METODO BASE

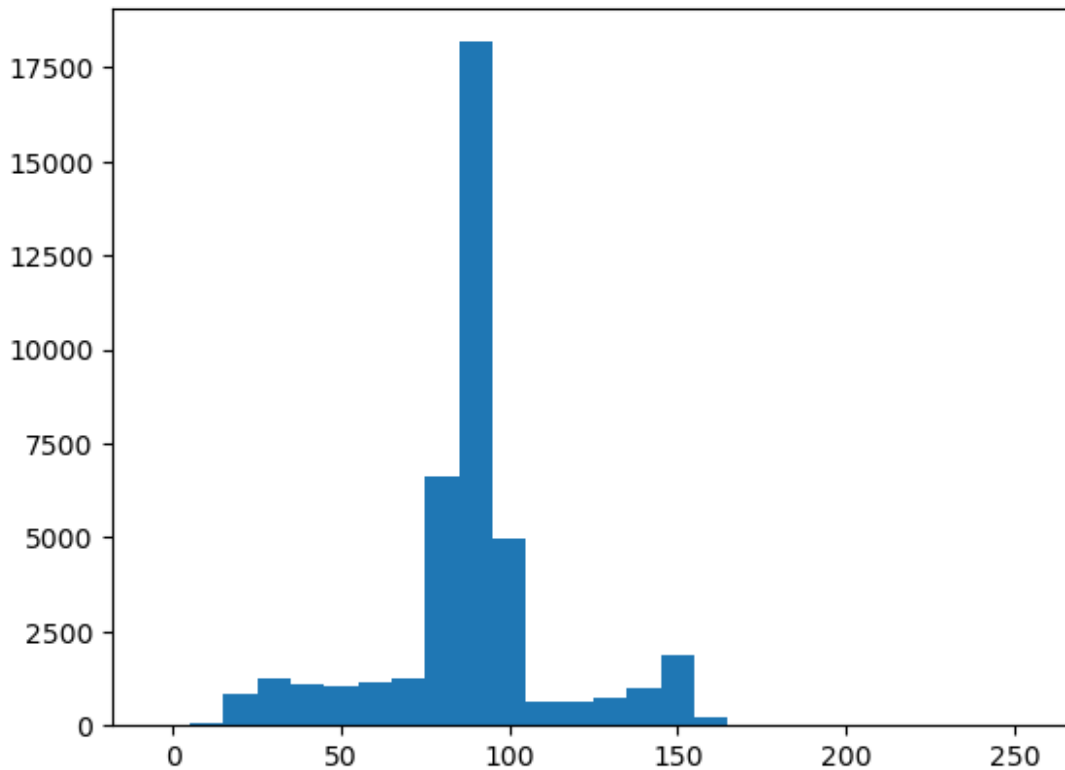
Obtenemos el histograma de la zona clara con el fin de obtener su moda

```
a,b = np.histogram(zona_clara,bins=np.arange(0,270,10))  
plt.bar( b[0:26],a[0:26],width=10)  
moda_zona_clara = b[np.argmax(a)+1]
```



Obtenemos el histograma de la zona oscura con el fin de obtener su moda

```
a,b = np.histogram(zona_oscura,bins=np.arange(0,270,10))  
plt.bar( b[0:26],a[0:26],width=10)  
moda_zona_oscura = b[np.argmax(a)+1]
```



Modificamos el histograma de la zona oscura con el fin de igualar la moda de ambas zonas

```
osc = np.copy(zona_oscura) + ( -moda_zona_oscura + moda_zona_clara)
```

```
#osc = zona_oscura
```

```
lessThen0 = osc<0
```

```
moreThen255 = osc>255
```

```
osc[lessThen0] = 0
```

```
osc[moreThen255] = 255
```

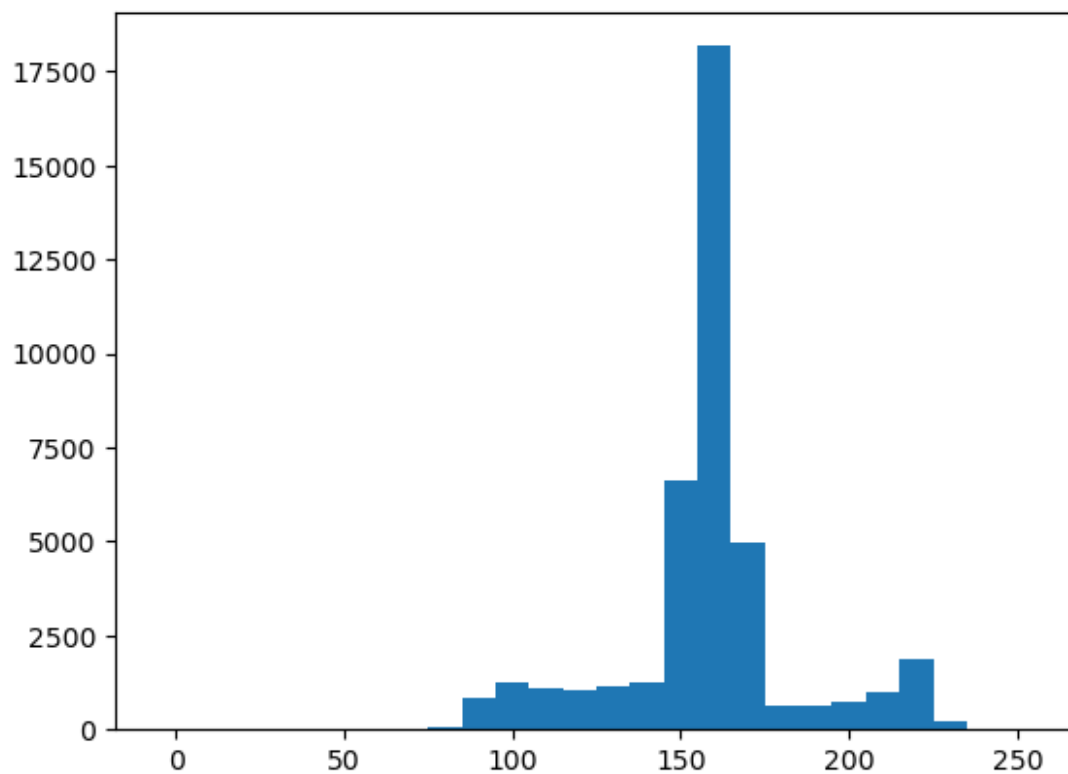
```
#imprimir_pic(osc,"osc")
```

```
a,b = np.histogram(osc,bins=np.arange(0,270,10))
```

```
plt.bar( b[0:26],a[0:26],width=10)
```

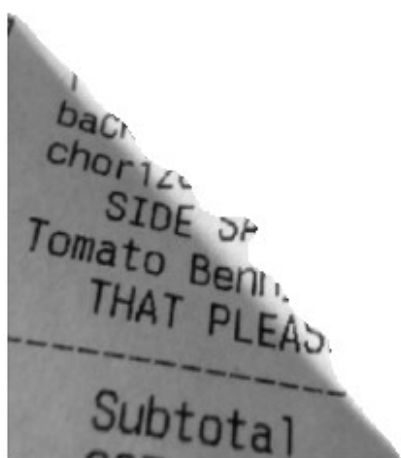
```
#moda_zona_oscura = b[np.argmax(a)+1]
```

```
<BarContainer object of 26 artists>
```

```
imprimir_pic(osc,"Zona oscura con histograma modificado")
```

Zona oscura con histograma modificado



```
imprimir_pic(zona_clara,"zona_clara")
```



Combinamos ambas zonas: la zona clara y la zona oscura modificada

```
z1 = np.nan_to_num(zona_clara,0)
```

```
z2 = np.nan_to_num(osc,0)
```

```
suma_parcial = (z1 + z2)
```

```
imprimir_pic(suma_parcial,"combianadas")
```

combianadas



Restamos las mascara (en tonos de gris) de la imagen original

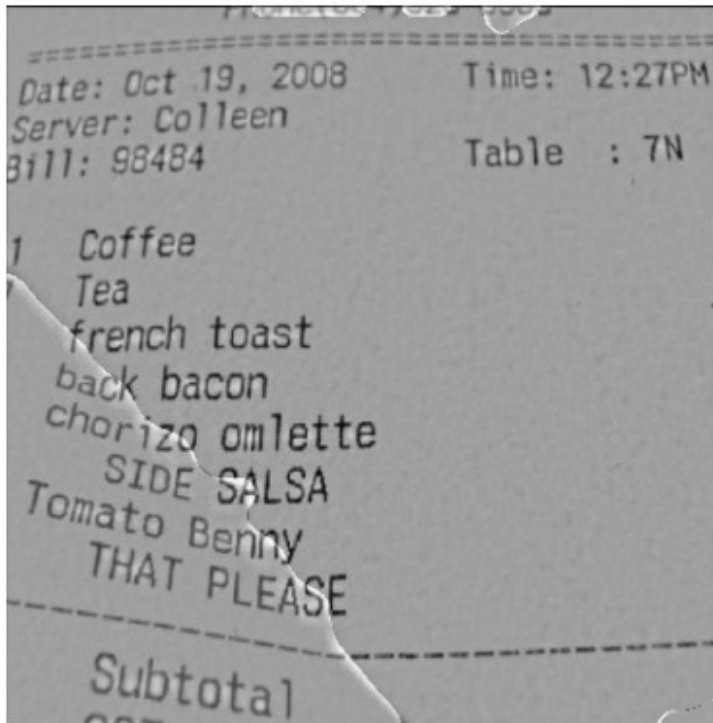
```
resta_parcial = np.copy(suma_parcial)

mask_gris = ndimage.median_filter(suma_parcial,
size=15,mode='reflect')

resta_parcial = ((suma_parcial - mask_gris)+255)/(2)
lessThen0 = resta_parcial<0
moreThen255 = resta_parcial>255
resta_parcial[lessThen0] = 0
resta_parcial[moreThen255] = 255

imprimir_pic(resta_parcial,"resta")
```

resta



Aplicamos un contraste optimizado por MSE a la imagen de "resta parcial"

```
p_indicado = contraste_optimo_por_MSE(255 - resta_parcial, 255 -  
input_pic)  
final_metodo_BASE = funcion_de_contraste(resta_parcial, p_indicado)  
imprimir_pic(final_metodo_BASE, "final_metodo_BASE")
```

final_metodo_BASE

```

=====
Date: Oct 19, 2008      Time: 12:27PM
Server: Colleen
Bill: 98484             Table : 7N

1 Coffee
1 Tea
french toast
back bacon
chorizo omlette
SIDE SALSA
Tomato Benny
THAT PLEASE
-----
Subtotal
  
```

METODO B

Importar las librerias necesarias:

```

import matplotlib.pyplot as plt
from PIL import Image
import scipy.ndimage as snd
import numpy as np
import cv2
from skimage.filters import threshold_otsu
from cv2 import threshold, adaptiveThreshold
  
```

Declarar metodos para aplicar Clausura y Threshold:

```

def closing(image_array):

    # Estructurar elemento de clausura:
    huella = np.ones((40, 40))
    # Aplicar clausura de grises:
    fondo = snd.grey_closing(image_array, footprint=huella)
    # Se resta el fondo de la imagen:
    fondo_libre = (image_array.astype(np.float64) -
fondo.astype(np.float64))
    # Se reescala la imagen con el fondo libre de 0 a 255:
    denominador = (fondo_libre.max() - fondo_libre.min())
  
```

```

    fondo_libre_normalizado = (fondo_libre - fondo_libre.min())*
255/denominador
    # Convertir fondo_libre_normalizado a uint8:
    fondo_libre_normalizado = fondo_libre_normalizado.astype(np.uint8)
    # Convertir fondo_libre_normalizado a imagen:
    fondo_libre_normalizado = Image.fromarray(fondo_libre_normalizado)
    return fondo_libre_normalizado

```

```

def threshold(image_array):
    th = cv2.adaptiveThreshold(np.asarray(image_array),
        255, # Maximo valor asignado al valor de un pixel que excede el
        'threshold'.
        cv2.ADAPTIVE_THRESH_MEAN_C, # suma ponderada "gaussiana" de los
        vecinos.
        cv2.THRESH_BINARY, # Tipo de threshold.
        15, # Tamaño del bloque (ventana 5x5)
        12)

    # Convertir th a imagen:
    th = Image.fromarray(th)

    return th

```

```

# Importar imagen:
input_image = input_pic
# Convertir la imagen en arreglo:
image_array = np.asarray(input_image)
# Aplicar Clausura:
closing_image = closing(image_array)
# Aplicar Threshold:
threshold_image = threshold(closing_image)

```

```

input_image

```

```

array([[ 35,  36,  36, ...,  35,  35,  36],
       [128, 136, 140, ..., 149, 148, 146],
       [137, 151, 152, ..., 163, 161, 159],
       ...,
       [ 96, 107, 108, ..., 151, 152, 152],
       [ 99, 107, 105, ..., 155, 156, 157],
       [ 98, 108, 107, ..., 153, 154, 154]], dtype=uint8)

```

```

closing_image

```

=====

Phone (604) 620-0000

Date: Oct 19, 2008
Server: Colleen
Bill: 98484

Time: 12:27PM

Table : 7N

1 Coffee
Tea
french toast
back bacon
chorizo omlette
SIDE SALSA
Tomato Benny
THAT PLEASE

Subtotal

threshold_image

=====

Date: Oct 19, 2008
Server: Colleen
Bill: 98484

Time: 12:27PM

Table : 7N

1 Coffee
Tea
french toast
back bacon
chorizo omlette
SIDE SALSA
Tomato Benny
THAT PLEASE

Subtotal

METODO A

```
mask = dilation(image=mask, selem=disk(40))
```

```
imprimir_pic(mask, 'Máscara')
```

```
imprimir_pic(input_pic, 'original')
```

```
/tmp/ipykernel_6417/2346952556.py:1: FutureWarning: `selem` is a  
deprecated argument name for `dilation`. It will be removed in version  
1.0. Please use `footprint` instead.
```

```
mask = dilation(image=mask, selem=disk(40))
```


Máscara



original



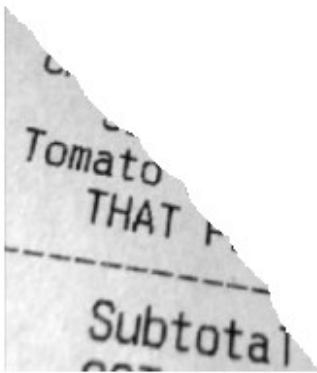
```
zona_oscura = np.full(input_pic.shape,np.nan)
limite = np.nanmean(input_pic)
```

```

for (coor_y,coor_x), value in np.ndenumerate(input_pic):
    aux = mask[coor_y:coor_y+1,coor_x:coor_x+1]
    if aux<limite: #Good Pixel
        zona_oscura[coor_y,coor_x] = input_pic[coor_y,coor_x]
    elif aux>limite: #Bad Pixel
        None
imprimir_pic(zona_oscura,"zona_oscura")

zona_oscura

```



```

zona_clara = np.full(input_pic.shape,np.nan)
limite = np.nanmean(input_pic)

for (coor_y,coor_x), value in np.ndenumerate(input_pic):
    aux = mask[coor_y:coor_y+1,coor_x:coor_x+1]
    if aux>limite: #Good Pixel
        zona_clara[coor_y,coor_x] = input_pic[coor_y,coor_x]
    elif aux<=limite: #Bad Pixel
        None
imprimir_pic(zona_clara,"zona_clara")

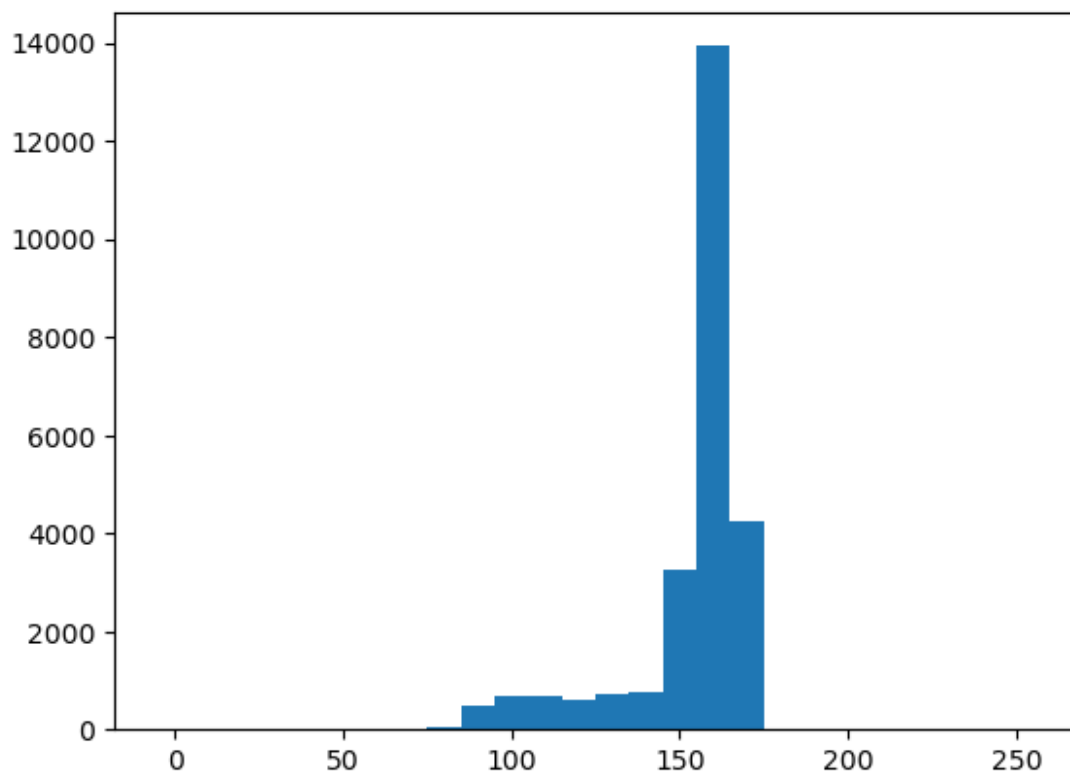
```

zona_clara



```
#a,b = np.histogram(zona_clara,bins=np.arange(0,270,10))
#plt.bar( b[0:26],a[0:26],width=10)
#moda_zona_clara = b[np.argmax(a)+1]
#
#a,b = np.histogram(zona_oscura,bins=np.arange(0,270,10))
#plt.bar( b[0:26],a[0:26],width=10)
#moda_zona_oscura = b[np.argmax(a)+1]

osc = np.copy(zona_oscura) + ( -moda_zona_oscura + moda_zona_clara)
#osc = zona_oscura
lessThen0 = osc<0
moreThen255 = osc>255
osc[lessThen0] = 0
osc[moreThen255] = 255
#imprimir_pic(osc,"osc")
a,b = np.histogram(osc,bins=np.arange(0,270,10))
plt.bar( b[0:26],a[0:26],width=10)
#moda_zona_oscura = b[np.argmax(a)+1]
imprimir_pic(osc,"Zona oscura con histograma modificado")
imprimir_pic(zona_clara,"zona_clara")
```



Zona oscura con histograma modificado

Tomato
THAT P

Subtotal
665

zona_clara



```
z1 = np.nan_to_num(zona_clara,0)
z2 = np.nan_to_num(osc,0)

suma_parcial = (z1 + z2)
imprimir_pic(suma_parcial,"combianadas")
```

combianadas



```
resta_parcial = np.copy(suma_parcial)
```

```
mask_gris = ndimage.median_filter(suma_parcial,  
size=15,mode='reflect')
```

```
resta_parcial = ((suma_parcial - mask_gris)+255)/(2)
```

```
lessThen0 = resta_parcial<0
```

```
moreThen255 = resta_parcial>255
```

```
resta_parcial[lessThen0] = 0
```

```
resta_parcial[moreThen255] = 255
```

```
imprimir_pic(resta_parcial,"resta")
```

```
p_indicado = contraste_optimo_por_MSE(255 - resta_parcial,255 -  
input_pic)
```

```
final_metodo_A = funcion_de_contraste(resta_parcial,p_indicado)
```

```
imprimir_pic(final_metodo_A,"final_metodo_A")
```

resta

=====

Date: Oct 19, 2008 Time: 12:27PM
Server: Colleen
Bill: 98484 Table : 7N

1 Coffee
 Tea
 french toast
 back bacon
 chorizo omlette
 SIDE SALSA
 Tomato Benny
 THAT PLEASE

Subtotal

final_metodo_A

=====

Date: Oct 19, 2008 Time: 12:27PM
Server: Colleen
Bill: 98484 Table : 7N

1 Coffee
 Tea
 french toast
 back bacon
 chorizo omlette
 SIDE SALSA
 Tomato Benny
 THAT PLEASE

Subtotal

```
plt.figure(figsize=(15,12))  
plt.subplot(1,4, 1)
```

```

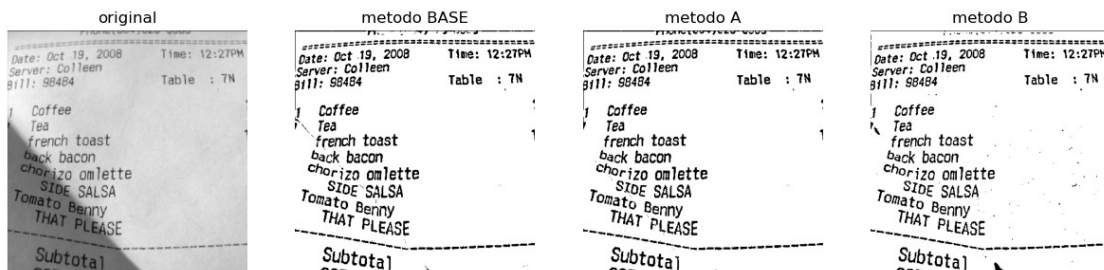
plt.imshow(input_pic, cmap = 'gray')
plt.title('original')
plt.axis('off')

plt.subplot(1,4, 2)
plt.imshow(final_metodo_BASE, cmap = 'gray')
plt.axis('off')
plt.title('metodo BASE')

plt.subplot(1,4, 3)
plt.imshow(final_metodo_A, cmap = 'gray')
plt.axis('off')
plt.title('metodo A')

plt.subplot(1,4, 4)
plt.imshow(threshold_image, cmap = 'gray')
plt.axis('off')
plt.title('metodo B')
plt.show()

```



Guardamos el mejor resultado como archivo jpeg

```
type(threshold_image)
```

```
PIL.Image.Image
```

```
nombre = imagen_entrada + "_PROCESADA.jpeg"
```

```
threshold_image.save(nombre)
```