

Trabajo práctico 1: Imágenes digitales

IGNACIO LEMBO FERRARI¹

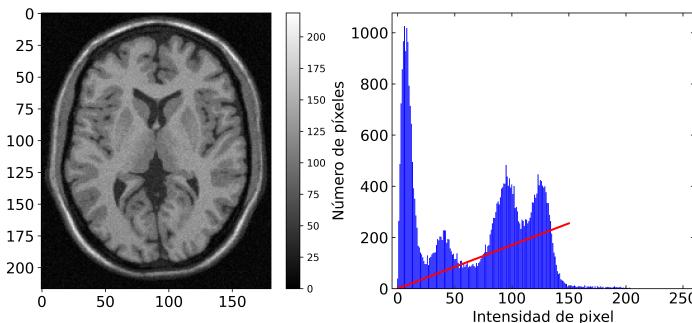
¹ignaciolembo@ib.edu.ar

1 de marzo del 2024.

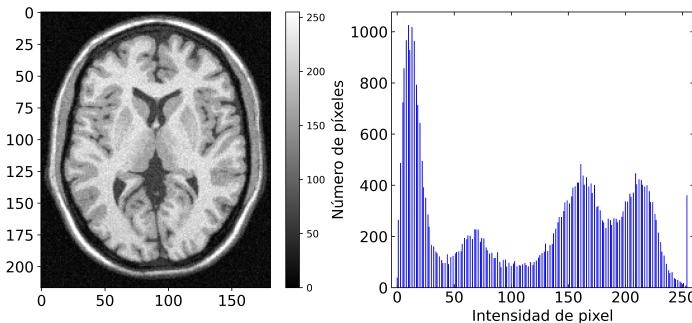
1. TRANSFORMACIONES DE CONTRASTE

Se tiene una imagen ImagenA.pgm de 181×217 pixeles de la cual se obtuvo el histograma de intensidades, como se muestra en la Fig. 1a, y sobre la cual se implementaron diversas transformaciones de contraste. Todas las implementaciones de esta sección se realizaron en python.

En primer lugar, se aplicó una transformación semilinear en los niveles de gris para optimizar el rango dinámico.



(a) ImagenA original junto con su histograma de intensidades.



(b) ImagenA transformada junto con su histograma de intensidades.

Fig. 1. ImageA original (a) y su transformación semilinear con $r_{min} = 0$ y $r_{max} = 150$ (b), junto con sus correspondientes histogramas de intensidades. Se observa en la imagen transformada que el histograma se corre hacia la derecha y se expande el rango dinámico.

Este método consiste en aplicar una transformación de la forma $T(r) = ar + b$, con r la intensidad del pixel, $a = (I_{max} - I_{min}) / (r_{max} - r_{min})$ y $b = I_{min} - ar_{min}$, además

- Si $ar + b < I_{min} \Rightarrow T(r) = I_{min}$,
- Si $ar + b > I_{max} \Rightarrow T(r) = I_{max}$.

Se tomó $I_{min} = 0$ e $I_{max} = 255$. Para determinar los valores r_{min} y r_{max} de la imagen, se observa, en el histograma de la imagen original, que la mayor cantidad de valores de intensidad se concentran en el rango [0, 150]. Luego se toma $r_{min} = 0$ y $r_{max} = 150$. De esta manera, se obtiene una transformación semilineal que expande el rango dinámico de la imagen. Cabe destacar que los valores por debajo de I_{min} y por encima de I_{max} saturan en dichos valores, lo cual resulta en pérdida de información de la imagen original. En la Fig. 1b se observa la transformación semilinear junto con su correspondiente histograma de intensidades. Se observa que el histograma se corre hacia la derecha y se expande el rango dinámico. Podemos ver, en el histograma de la imagen transformada que se acumulan píxeles con intensidad máxima I_{max} .

En el histograma de la imagen original se distinguen 4 picos de intensidad, que corresponden 4 regiones de la imagen; De menor a mayor valor de intensidad: Fondo, líquido, materia gris y materia blanca. Se eligieron convenientemente los valores de r_{min} y r_{max} para que la transformación semilinear resalte cada una de estas regiones. El resultado se muestra en la Fig. 2.

Por otro lado, se ecualizó la imagen A como se muestra en la Fig. 3. La ecualización corresponde a una transformación de la forma

$$T(r)) (I_{max} - I_{min}) \sum_{i=0}^r P(i) \quad (1)$$

donde $P(i)$ es la distribución de probabilidad de que un pixel tenga intensidad i . En particular, se toma esta distribución normalizando el histograma de intensidades de la imagen original. Se observa que se expande el

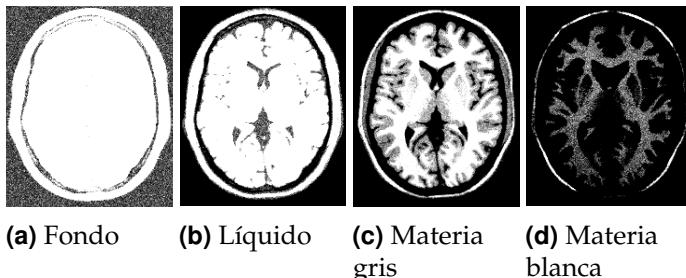


Fig. 2. Transformación semilinear aplicada a distintas regiones de la imagen original.

rango dinámico, mejorando el contraste de la imagen sin acumulación de intensidades como sucedía en la transformación semilinear.

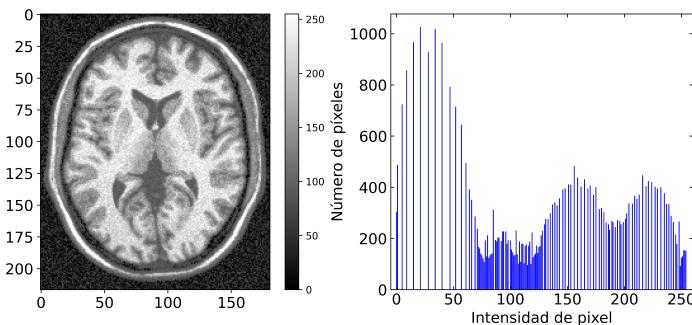


Fig. 3. ImagenA ecualizada junto con su histograma de intensidades.

Se pueden realizar otras transformaciones de contraste, como la binarización de la imagen, que consiste en transformar la imagen según

$$T(r) = \begin{cases} 0 & \text{si } r \geq 128 \\ 255 & \text{si } r < 128 \end{cases} \quad (2)$$

o una transformación según una ley de potencias de la forma

$$T(r) = I_{max} \left(\frac{r}{r_{max}} \right)^\gamma \quad (3)$$

donde se tomó $I_{max} = 255$ y distintos valores de γ .

En la Fig. 4 se muestra la imagen binarizada de la imagen A original. Esta transformación muestra en negro los valores de la imagen por encima de un cierto umbral, en este caso 128, y en blanco el resto. Esto sirve para resaltar estructuras por encima de dicho umbral eliminando el resto de información de la imagen.

En la Fig. 5 se muestra la transformación γ aplicada a la imagenA original. En este caso se ve que valores de $\gamma < 1$ generan una imagen más brillante y valores de $\gamma > 1$ más oscura.

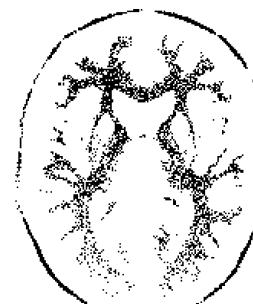


Fig. 4. Transformación binaria aplicada a la imagenA original.

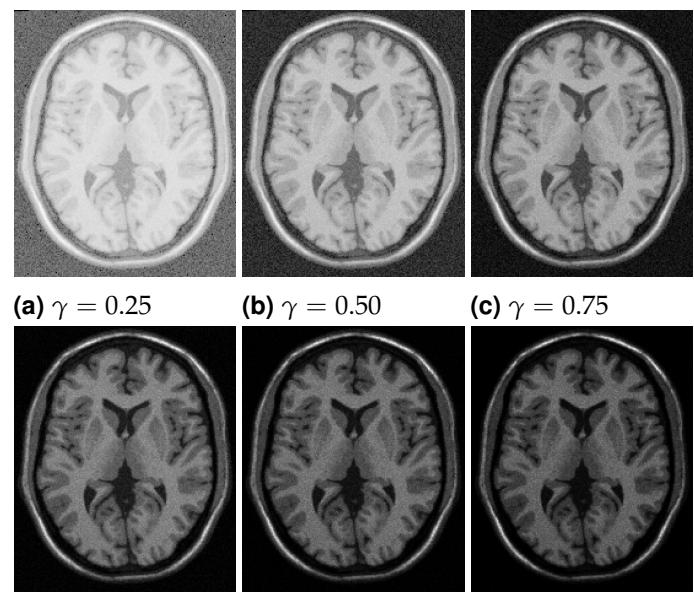


Fig. 5. Transformación γ aplicada a la imagenA original.

Por último, en las Figs. 6, 7, 8 y 9 se muestran la resta de la imagenA original con la imagenA luego de aplicarle transformaciones de ecualización, binarización y γ con $\gamma = 0.5, 1.75$ respectivamente.

2. INTERPOLACIÓN

Se tomó la imagen C de tamaño 128×128 pixels² y se interpoló utilizando el método de vecinos más cercanos y bilineal implementados en python para obtener imágenes de 32×32 , 64×64 , 256×256 y 1024×1024 . La implementación de ambas técnicas se realizó en python. Los resultados se muestran en la Fig. 10. Se observa que la interpolación bilineal es más suave en la transición de la escala de grises que la interpolación por vecinos más cercanos.

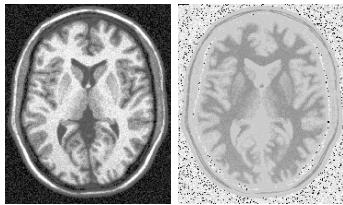


Fig. 6. Diferencia entre la imagen original y la imagen ecualizada.

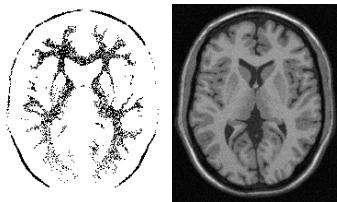


Fig. 7. Diferencia entre la imagen original y la imagen binarizada.

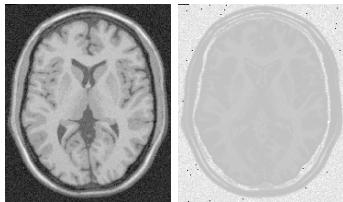


Fig. 8. Diferencia entre la imagen original y la imagen transformada con $\gamma = 0.5$.

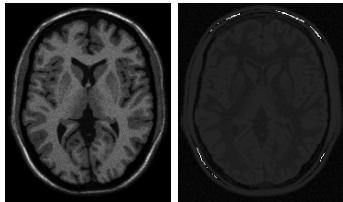


Fig. 9. Diferencia entre la imagen original y la imagen transformada con $\gamma = 1.75$.

3. FILTROS EN EL DOMINIO ESPACIAL

Se aplicaron filtros promedios pasabajos a las imágenes A y C con kernels de 3×3 , 5×5 y 7×7 como se muestra en la Fig. 11 utilizando ImageJ. Estos filtros se definen a partir de kernels de la forma

$$h = \frac{1}{n^2} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \quad (4)$$

donde n es la dimensión de la matriz.

Se observa que a medida que se aumenta la dimensión del kernel, la imagen se vuelve cada vez más borrosa, es decir, el suavizado es mayor.

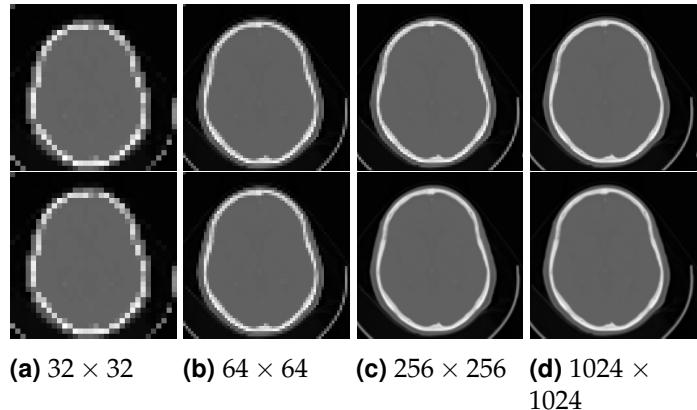


Fig. 10. Distintos reescalamientos de la imagenC con interpolación por los métodos de vecinos más cercanos (arriba) y bilineal (abajo).

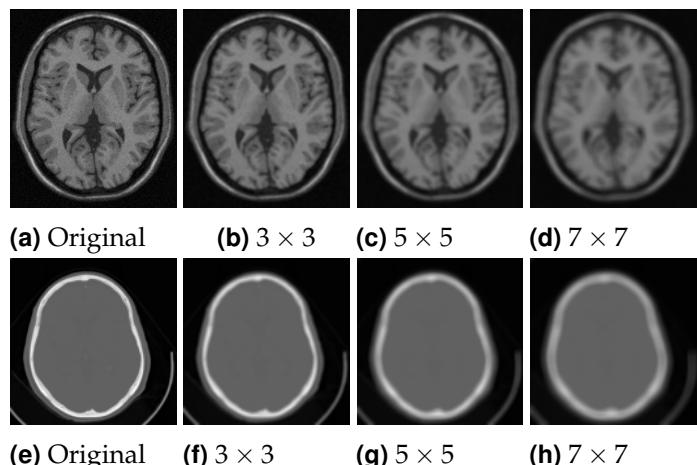


Fig. 11. Filtros promedios pasabajos aplicados a la imagenA (arriba) y a la imagenC (abajo) para distintas dimensiones de kernels.

4. FILTROS EN EL DOMINIO DE FRECUENCIAS

Dada una figura con componentes periódicas en su textura como la que se muestra en la Fig. 12a es posible eliminar dichas componentes a partir de un procesamiento en el espacio de frecuencias de dicha imagen. Para esto se calcula la transformada de Fourier rápida (FFT por sus siglas en inglés) de la imagen con ImageJ y luego de reconocer las componentes periódicas, que en el espacio de frecuencias se manifiestan como puntos de alta intensidad, se procede a eliminarlas. Una vez eliminadas estas componentes se antitransforma la imagen (inverse FFT) y se recupera la imagen original con las componentes periódicas eliminadas. En la Fig. 12 se muestra la imagen original (a) y la imagen procesada (b), mientras que en la Fig. 13 se muestra el

espacio de frecuencias con la transformada de Fourier de la imagen original (a) y la transformada de Fourier con las componentes periódicas eliminadas (b).

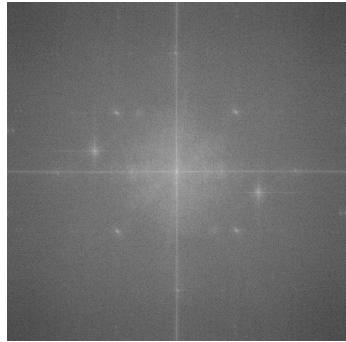


(a) Imagen original.

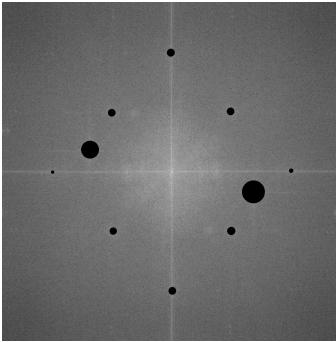


(b) Imagen procesada.

Fig. 12. Comparación imagen original de Superman (a) con la imagen procesada (b) en el espacio de frecuencias sin las componentes periódicas en la textura.



(a) FFT de Superman.



(b) FFT de superman procesada.

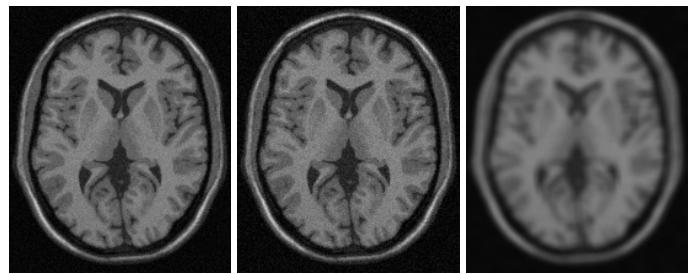
Fig. 13. Transformada de Fourier rápida de Superman (a). Transformada de Fourier rápida sin las componentes periódicas. En el espacio de frecuencias las componentes periódicas se manifiestan como puntos de alta intensidad.

5. FILTROS UNSHARP Y HIGHBOOST

Los filtros Unsharp y Highboost son filtros que se utilizan para resaltar los bordes de una imagen, se definen de la siguiente manera: Sea $f(x, y)$ la imagen original con ruido, $\tilde{f}(x, y)$ la imagen original luego de aplicarle un filtro pasabajos y $g(x, y)$ la imagen resultante, entonces los filtros se definen como

- Unsharp: $g(x, y) = f(x, y) - \tilde{f}(x, y)$
- Highboost: $g(x, y) = A f(x, y) - \tilde{f}(x, y)$

donde A es la intensidad del filtro High-boost y vemos que para $A = 1$ el filtro High-boost es equivalente al filtro Unsharp. Se tomó la imagen A y se le agregó ruido gaussiano con desvío estándar $\sigma = 5$. Luego, se aplicaron los filtros Unsharp y Highboost utilizando como filtro pasa bajo promedio con un kernel de 7×7 . Los resultados se muestran en la Fig. 14. Se observa que ambos filtros resaltan los bordes de la imagen y amplifican el ruido, lo cual es esperable para este tipo de filtro que esencialmente se comportan como pasa altos. Se puede ver que el filtro Highboost con $A = 2$ resalta los bordes, conservando mejor la imagen original. En la Fig. 15 se grafica la media cuadrática de la resta entre la imagen con el filtro High-boost y la imagen original sin ruido en función de la intensidad del parámetro A . Se observa que $A = 2$ es el valor óptimo para resaltar los bordes de la imagen alejándose lo menos posible (en media) de la imagen original.



(a) ImagenA Original. (b) ImagenA con ruido.

(c) ImagenA con ruido filtrada.

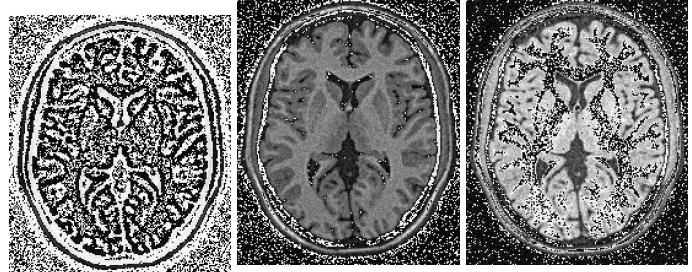
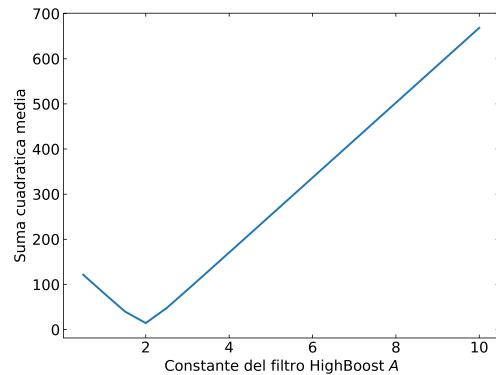
(d) Filtro Unsharp. (e) Filtro Highboost con $A = 2$.(f) Filtro Highboost con $A = 3$.

Fig. 14. Filtraos Unsharp y Highboost aplicados a la imagenA con ruido gaussiano con desvío estandar $\sigma = 5$.

6. CALIBRACIÓN DE FANTOMAS

Se calcularon las relaciones señal-ruido (SNR) para los cortes 2, 3 y 4 de un fantoma adquiridos con CT



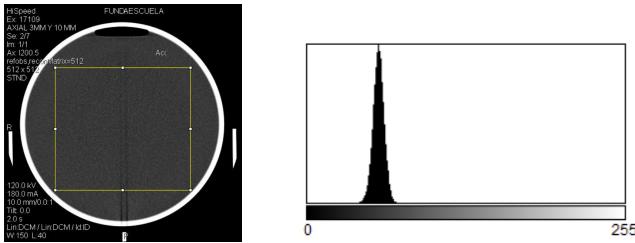
(a) ImagenA original junto con su histograma de intensidades.

Fig. 15. Gráfica de la media cuadrática de la diferencia entre la imagen original sin ruido y la imagen filtrada con el filtro High-boost en función del parámetro A .

HiSpeed. En este caso, se utilizó ImageJ para obtener los histogramas de intensidad de pixel de una región de interés (ROI por sus siglas en inglés) de las imágenes de los fantomas tomadas como se muestra en la Fig. 16 para el caso de la imagen AAA0002. La SNR se calcula como

$$SNR = \frac{\mu}{\sigma} \quad (5)$$

donde μ es la media de la intensidad de la imagen y σ es la desviación estándar en la ROI elegida. En la tabla 1 se muestran los valores de μ , σ y SNR para cada corte. Se observa que el SNR disminuye a medida que disminuye la corriente del tubo de rayos X. Esto es consistente con lo observado en las imágenes de los cortes, donde se observa que a menor corriente, las imágenes se vuelven más ruidosas.



(a) Imagen con la ROI seleccionada. (b) Histograma de intensidades.

Fig. 16. Estimación de la SNR para la imagen AAA0002.

Por otro lado, se midieron el ancho total a media altura (FWHM por sus siglas en inglés) del perfil de

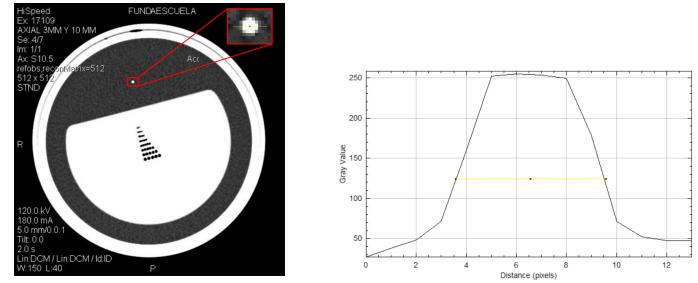
Corte	Corriente (mA)	μ	σ	SNR
2	180.0	57.074	4.422	12.907
3	100.0	56.722	5.769	9.832
4	60.0	56.728	7.286	7.786

Tabla 1. Tabla de corriente, media, desviación estándar y SNR para las imágenes AAA000x siendo x el corte.

intensidad del punto del fantoma para los cortes 11, 12, 13 y 14. En la Fig. 17a se muestra el corte 11 y la ROI sobre el cual se obtuvo el perfil de intensidad y en la Fig. 17b se muestra el FWHM para dicho perfil de intensidad. En la tabla 2 se muestran los valores de FWHM para cada corte.

Corte	FWHM
11	5.720
12	2.967
13	6.267
14	4.425

Tabla 2. Tabla FWHM para las imágenes AAA00xx siendo x el corte.



(a) Imagen con la ROI seleccionada. (b) Perfil de intensidad.

Fig. 17. Estimación del FWHM para la imagen AAA0011.

A. APÉNDICE

```
1 import cv2
2 import sys
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import os.path
6
7 def read_pgm_file(file_name):
8
9     data_dir = os.path.dirname(os.path.abspath(__file__))
10
11     # Test if file exists
12     file_path = os.path.join(data_dir, file_name)
13     assert os.path.isfile(file_path), 'file \'{0}\' does not exist'.format(file_path)
14
15     img = cv2.imread(file_name, flags=cv2.IMREAD_GRAYSCALE)
16
17     if img is not None:
18         print('img.size: ', img.size)
19     else:
20         print('imread({0}) -> None'.format(file_path))
21
22     return img
23
24 def save_img_hist(im,filename):
25
26     vmin = np.amin(im)
27     vmax = np.max(im)
28     print("Intensity Min: {} Max:{}".format(vmin,vmax))
29
30     L = vmax - vmin
31     print("Number of Levels: {}".format(L))
32     fig = plt.figure(figsize=(12,5))
33     ax1 = plt.subplot(1, 2, 1)
34     ax2 = plt.subplot(1, 2, 2)
35
36     # imgplot = plt.imshow(im/np.amax(im))
37     imgplot = ax1.imshow(im,cmap='gray', vmin=vmin, vmax=vmax)
38     fig.colorbar(imgplot, ax=ax1)
39     #ax1.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
40     ax1.tick_params(axis='x', rotation=0, labelsize=15, color='black')
41     ax1.tick_params(axis='y', labelsize=15, color='black')
42     # cv2.imshow(infile,img)
43     # cv2.waitKey(0)
44
45     hist, bin_edges = np.histogram(im.ravel(),bins=L)
46     ax2.bar(bin_edges[:-1], hist, alpha=1, color='b')
47     ax2.set_xlabel(r"Intensidad de pixel", fontsize=15)
48     ax2.set_ylabel(r"N mero de p xeles", fontsize=15)
49     ax2.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
50     ax2.tick_params(axis='x', rotation=0, labelsize=15, color='black')
51     ax2.tick_params(axis='y', labelsize=15, color='black')
52     ax2.set_xlim(-5, 260)
53     #ax2.set_ylim(0, 1000)
54
55     #x1 = rmin # Punto inicial en el eje x
56     #x2 = rmax # Punto final en el eje x
57     #y1 = 0 # Punto inicial en el eje y (puedes ajustar este valor seg n tu necesidad)
58     #y2 = 255 # Punto final en el eje y (puedes ajustar este valor seg n tu necesidad)
59
60     #Agrega la recta al segundo subplot (ax2)
61     #ax2.plot([x1, x2], [y1, y2], color='r', linestyle='-', linewidth=2) # Trama la l nea recta
62
63     fig.savefig(f"{filename}_hist.pdf")
64     fig.savefig(f"{filename}_hist.png", dpi=600)
65     #plt.show()
66
67 def save_img(im,filename):
68
69     vmin = np.amin(im)
70     vmax = np.max(im)
```

```
71     print("Intensity Min: {}    Max:{}".format(vmin,vmax))
72
73     L = vmax - vmin
74     print("Number of Levels: {}".format(L))
75
76     fig, ax = plt.subplots()
77     imgplot = ax.imshow(im,cmap='gray', vmin=vmin, vmax=vmax)
78     ax.axis('off')
79
80     fig.savefig(f"{filename}.pdf", pad_inches = 0,bbox_inches='tight')
81     fig.savefig(f"{filename}.png", dpi=600, pad_inches = 0, bbox_inches='tight')
82     print("Size of image: {}".format(im.shape))
83     #plt.show()
84
85 def SemilinearTrans_pgm_file(im,rmin,rmax):
86     Imin = 0
87     Imax = 255
88     a = (Imax - Imin)/(rmax-rmin)
89     b = Imin - a*rmin
90     imout = a*im + b
91
92     imout = np.where(imout < Imin, Imin, imout)
93     imout = np.where(imout > Imax, Imax, imout)
94     imout = imout.astype(int)
95     return imout
96
97 def Equalize_pgm_file(im):
98     imout = im
99     vmin = np.amin(im)
100    vmax = np.max(im)
101    L = vmax - vmin
102    hist, bin_edges = np.histogram(im.ravel(),bins=L)
103    hist_norm = hist / np.sum(hist)  #Normaliza el histograma
104
105    suma = 0
106    for i in range(0,im.shape[0]):
107        for j in range(0,im.shape[1]):
108            suma = 0
109            for e in range(0,im[i,j]):
110                suma += hist_norm[e]
111            imout[i,j] = 255*suma
112
113    imout = imout.astype(int)
114    return imout
115
116 def Trans_binary(im):
117     imout=np.where(im<128,255,0)
118     imout = imout.astype(int)
119     return imout
120
121 def Trans_exponential(im,gamma):
122     imout=im
123     imout=255*(imout/im.max())**gamma
124     imout=imout.astype(int)
125     return imout
126
127 def sustraction(im1,im2):
128     imout = im1 - im2
129     return imout
130
131 def interpolate_nn(im,f):
132     Nx, Ny = im.shape[0],im.shape[1]
133     Mx=int(f*Nx)
134     My=int(f*Ny)
135     imout=np.ndarray((Mx,My), dtype=int)
136
137     for x in range(Mx):
138         for y in range(My):
139             imout[x][y]=im[round(x*Nx/Mx)%Nx][round(y*Ny/My)%Ny]
140
141     return imout
142
```

```
143     def interpolate_bilinear(im, output_width, output_height):
144
145         # Calculate scaling ratios
146         old_height, old_width = im.shape[:2]
147         x_ratio = old_width / output_width
148         y_ratio = old_height / output_height
149
150         # Create new image array
151         new_img = np.zeros((output_height, output_width), dtype=np.uint8)
152
153         # Perform bilinear interpolation
154         for y in range(output_height):
155             for x in range(output_width):
156                 x_original = x * x_ratio
157                 y_original = y * y_ratio
158                 x0 = int(x_original)
159                 y0 = int(y_original)
160                 x1 = min(x0 + 1, old_width - 1)
161                 y1 = min(y0 + 1, old_height - 1)
162
163                 Q11 = im[y0, x0]
164                 Q21 = im[y1, x0]
165                 Q12 = im[y0, x1]
166                 Q22 = im[y1, x1]
167
168                 x_weight = x_original - x0
169                 y_weight = y_original - y0
170
171                 R1 = Q11 * (1 - x_weight) + Q12 * x_weight
172                 R2 = Q21 * (1 - x_weight) + Q22 * x_weight
173
174                 P = R1 * (1 - y_weight) + R2 * y_weight
175
176                 new_img[y, x] = P
177
178         return new_img
179
180     def high_boost(im, im_filtered, A):
181         imout = A*im - im_filtered
182         return imout
183
184     def unsharp(im, im_filtered):
185         imout = im - im_filtered
186         return imout
187
188     if __name__ == "__main__":
189
190         if len(sys.argv)<3:
191             print("Usage: python read-write-pgm.py [infile.pgm] [outfile.pgm]")
192             exit(1)
193
194         infile1 = sys.argv[1]
195         outfile1 = sys.argv[2]
196         #infile2 = sys.argv[3]
197         #outfile2 = sys.argv[4]
198
199         img1 = read_pgm_file(infile1)
200         #img2 = read_pgm_file(infile2)
201         im1 = np.array(img1)
202         #im2 = np.array(img2)
203
204         #Brute image
205         print("\n Brute image:")
206         print("Size of image 1: {}".format(im1.shape))
207         #print("Size of image 2: {}".format(im2.shape))
208         save_img_hist(im1,"ImageA_brute",0,150)
209
210         #Semilinear transformation for different rmin values
211         print("\n Semilinear transformation")
212         img = read_pgm_file(infile1)
213         im = np.array(img)
214         rmax = 150 # 25, 62 , 113 ,150
```

```
215     rmin = 113 # 0, 25 , 62 , 113
216     imout = SemilinearTrans_pgm_file(im,rmin,rmax)
217     print("Size of image: {}".format(imout.shape))
218     save_img_hist(imout,"ImageA_"+str(rmin)+"_"+str(rmax))
219     save_img(imout,"ImageA_"+str(rmin)+"_"+str(rmax))
220
221     #Equalize image
222     print("\n Equalize image")
223     imout = Equalize_pgm_file(im1)
224     save_img_hist(imout,"ImageA_EQ")
225     save_img(imout,"ImageA_EQ")
226     #cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])
227
228     #Transformaci n binaria
229     print("\n Transformaci n binaria")
230     imout = Trans_binary(im1)
231     save_img(imout,f"ImageA_binary")
232     #cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])
233
234     #Transformaci n exponencial
235     print("\n Transformaci n exponencial")
236     gamma = 0.5
237     imout = Trans_exponential(im1,gamma)
238     save_img(imout,f"ImageA_exp_gamma={gamma}")
239     #cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])
240
241     #Interpolate nn
242     f = 0.25
243     print("\n Interpolate NN")
244     imout = interpolate_nn(im1,f)
245     save_img(imout,f"Interpolate_nn_f={f}")
246
247     #Interpolate bilinear
248     #f = 0.25
249     print("\n Interpolate bilinear")
250     imout = interpolate_bilinear(im1, int(f* im1.shape[0]), int(f* im1.shape[1]))
251     save_img(imout,f"Interpolate_bilinear_f={f}")
252     cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])
253
254     #Sustraction
255     print("\n Sustraction")
256     imout = sustraction(im1,im2)
257     save_img(imout,f"sustraction_exp_gamma=0.5")
258     cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])
259
260     #Unsharp filter
261     print("\n Unsharp filter")
262     imout1 = unsharp(im1,im2)
263     imout1 = np.array(imout1)
264     print("Size of image 1: {}".format(imout1.shape))
265     save_img(imout1,"unsharp")
266
267     #HighBoost filter
268     print("\n HighBoost filter ")
269     A=3
270     imout2 = high_boost(im1,im2,A)
271     imout2 = np.array(imout2)
272     print("Size of image 2: {}".format(imout2.shape))
273     save_img(imout2,f"highboost_A={A}")
274
275     #High Boost filter plot
276     print("\n High Boost filterplot")
277     A = np.arange(0.5, 10.5, 0.5)
278     S = []
279
280     for a in A:
281         imout2 = high_boost(im1,im2,a)
282         im_sust = sustraction(imout2,im1)
283         matriz_cuadrada = np.square(im_sust)
284         suma_total = np.sum(matriz_cuadrada)
285         media = np.sqrt(suma_total / im_sust.size)
286         S.append(S, media)
```

```
287     #imout2 = np.array(imout2)
288     #print("Size of image 1: {}".format(imout2.shape))
289     #imout2 = Equalize_pgm_file(imout2)
290     #save_img(imout2,f"highboost_A={A}")
291     #cv2.imwrite(outfile2,imout2,[cv2.IMWRITE_PXM_BINARY ,0])
292
293 fig, ax = plt.subplots(figsize=(8, 6))
294 ax.plot(A, S, linewidth=2)
295 ax.set_xlabel(f"Intensidad del filtro HighBoost ${A}$", fontsize=15)
296 ax.set_ylabel(f"Suma cuadratica media", fontsize=15)
297 ax.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
298 ax.tick_params(axis='x', rotation=0, labelsize=15, color='black')
299 ax.tick_params(axis='y', labelsize=15, color='black')
300 #ax.set_xlim(-5, 260)
301 #ax2.set_ylim(0, 1000)
302 fig.savefig(f"Highboost_plot.pdf", pad_inches = 0,bbox_inches='tight')
303 fig.savefig(f"Highboost_plot.png", dpi=600, pad_inches = 0, bbox_inches='tight')
```