

SOBRE EL PROBLEMA DE LA FORMACIÓN DE TRENES DE AUTOS EN EL TRÁFICO DE UNA CARRETERA ANGOSTA

Ignacio Lembo Ferrari ^{*1} y Manuel Avellaneda Molina ^{**1}

¹Instituto Balseiro

1. Transformaciones de contraste

Se tiene una imagen ImagenA.pgm de 181×217 pixels de la cual se obtuvo el histograma de intensidades, como se muestra en la Fig. ??a, 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.

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 concientran 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. ??b se observa la transformación semilineal 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 pixeles 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 ma-

^{*}Correo electrónico: ignacio.lembo@ib.edu.ar

^{**}Correo electrónico: manuel.avellaneda@ib.edu.ar

teria 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. ??.

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

$$T(r_k) = (r_{max} - r_{min}) \sum_{i=0}^k P_r(r_i) \quad (1)$$

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

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. ?? 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. ?? 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.

Por último, en las Figs. ??, ??, ?? y ?? 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. ?? . Se observa que la interpolación bilineal es más suave en la transición de la escala de grises que la interpolación por vecinos más cercanos.

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. ?? 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.

4. Filtros en el dominio de frecuencias

Dada una figura con componentes periódicas en su textura como la que se muestra en la Fig. ??a 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. ?? se muestra la imagen original (a) y la imagen procesada (b), mientras que en la Fig. ?? 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).

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. ???. 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. ?? 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.

6. Calibración de fantomas

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

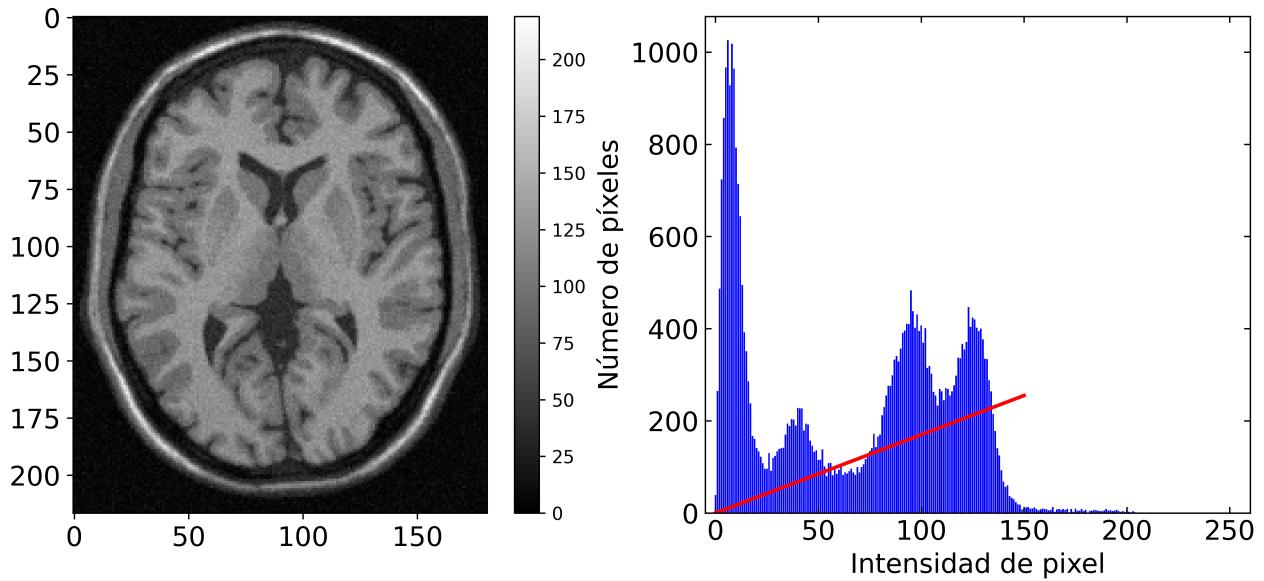
Corte	Corriente (mA)	μ	σ	SNR
2	180.0	57.074	4.422	12.907
3	100.0	56.722	5.769	9.832
4	60.0	56.728	7.286	7.786

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

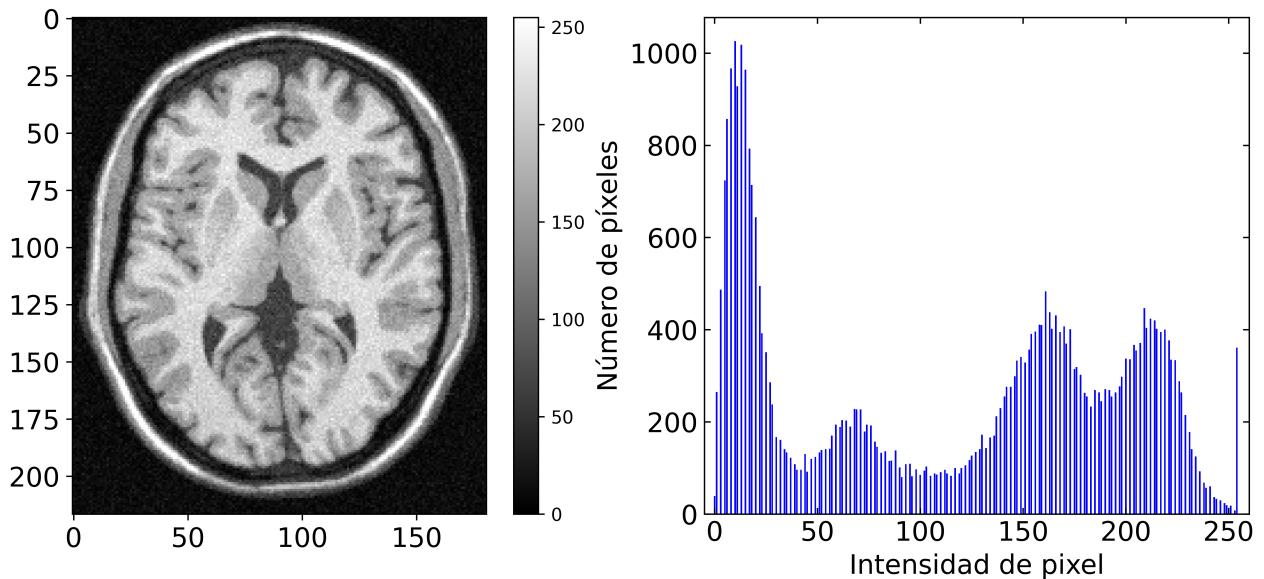
Por otro lado, se midieron el ancho total a media altura (FWHM por sus siglas en inglés) del perfil de intensidad del punto del fantoma para los cortes 11, 12, 13 y 14. En la Fig. ??a se muestra el corte 11 y la ROI sobre el cual se obtuvo el perfil de intensidad y en la Fig. ??b se muestra el FWHM para dicho perfil de intensidad. En la tabla ?? se muestran los valores de FWHM para cada corte.

Corte	FWHM
11	5.720
12	2.967
13	6.267
14	4.425

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



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



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

Figura 1: ImageA original (a) y su transformación semilineal 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.

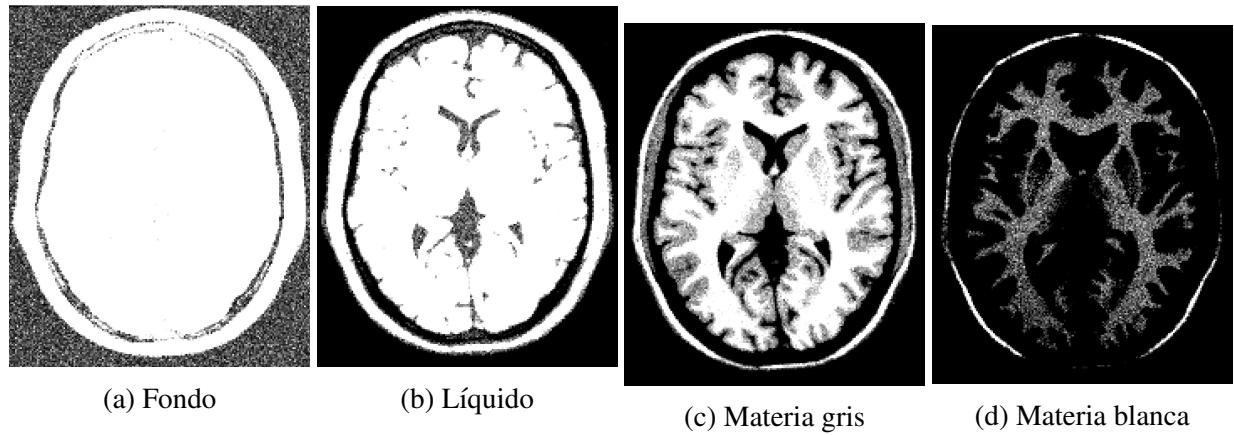


Figura 2: Transformación semilinear aplicada a distintas regiones de la imagen original.

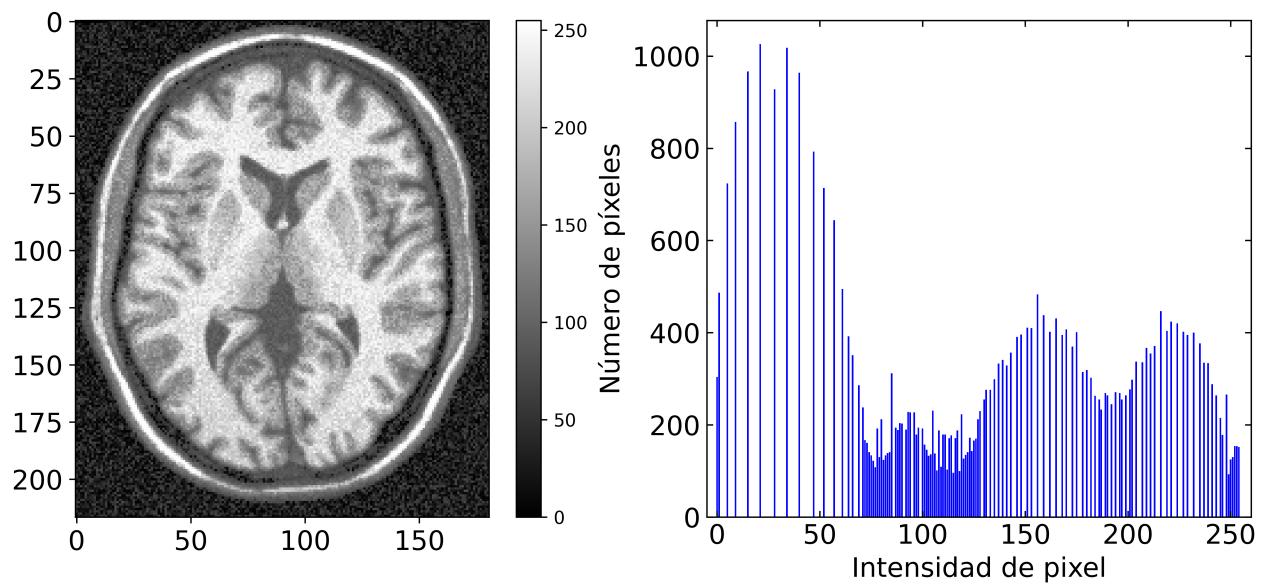


Figura 3: ImagenA ecualizada junto con su histograma de intensidades.

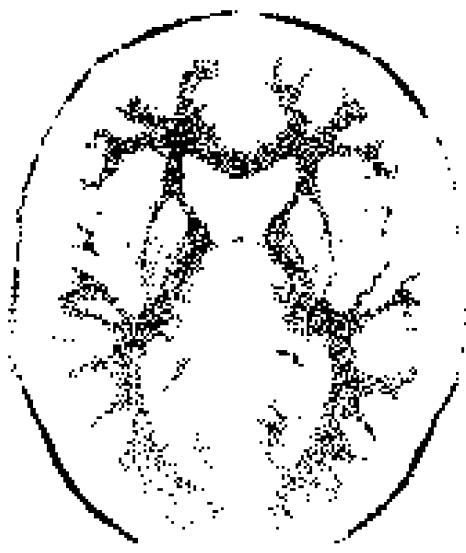


Figura 4: Transformación binaria aplicada a la imagenA original.

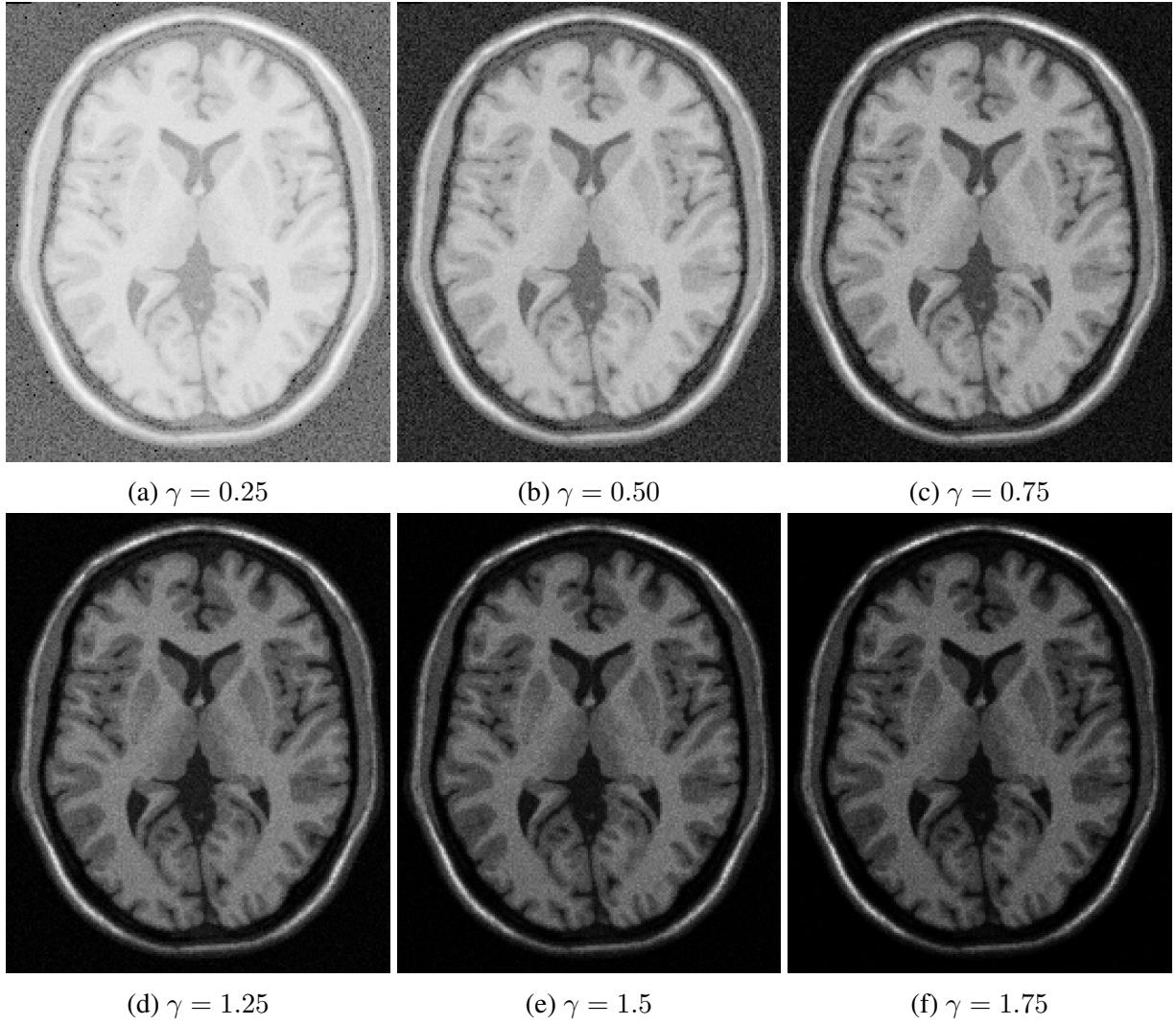


Figura 5: Transformación γ aplicada a la imagenA original.

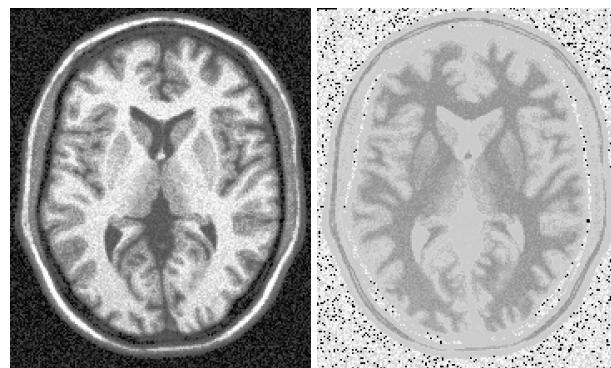


Figura 6: Diferencia entre la imagen original y la imagen ecualizada.

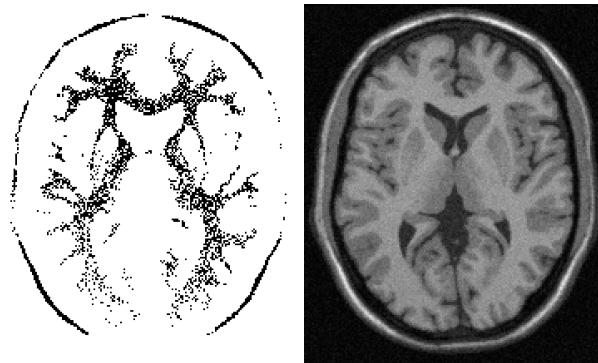


Figura 7: Diferencia entre la imagen original y la imagen binarizada.

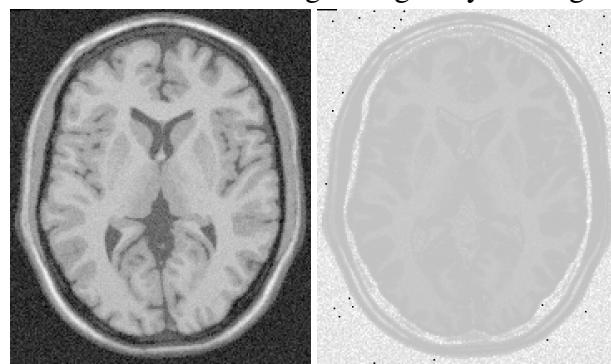


Figura 8: Diferencia entre la imagen original y la imagen transformada con $\gamma = 0.5$.

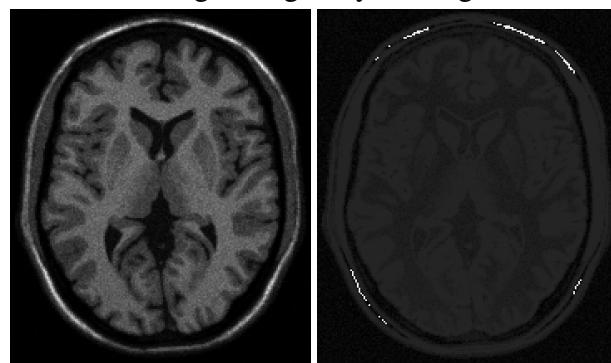


Figura 9: Diferencia entre la imagen original y la imagen transformada con $\gamma = 1.75$.

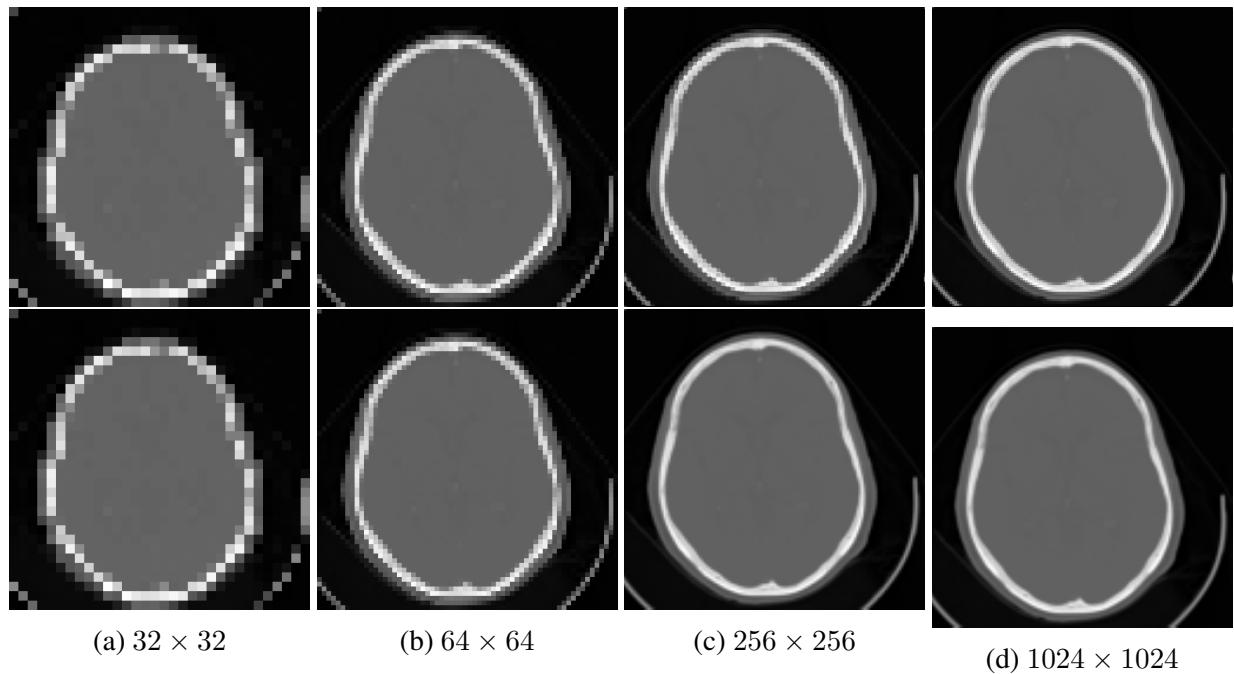


Figura 10: Distintos reescalados de la imagenC con interpolación por los métodos de vecinos más cercanos (arriba) y bilineal (abajo).

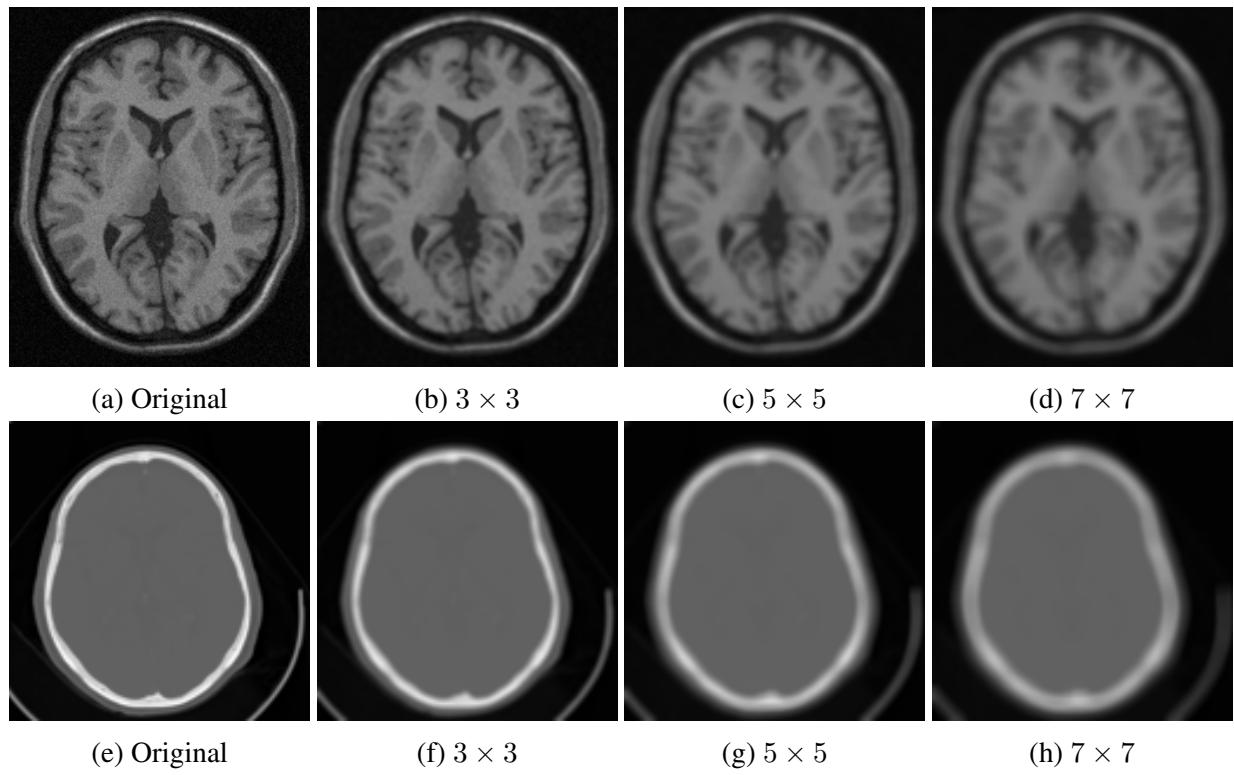


Figura 11: Filtros promedios pasabajos aplicados a la imagenA (arriba) y a la imagenC (abajo) para distintas dimensiones de kernels.



(a) Imagen original.



(b) Imagen procesada.

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

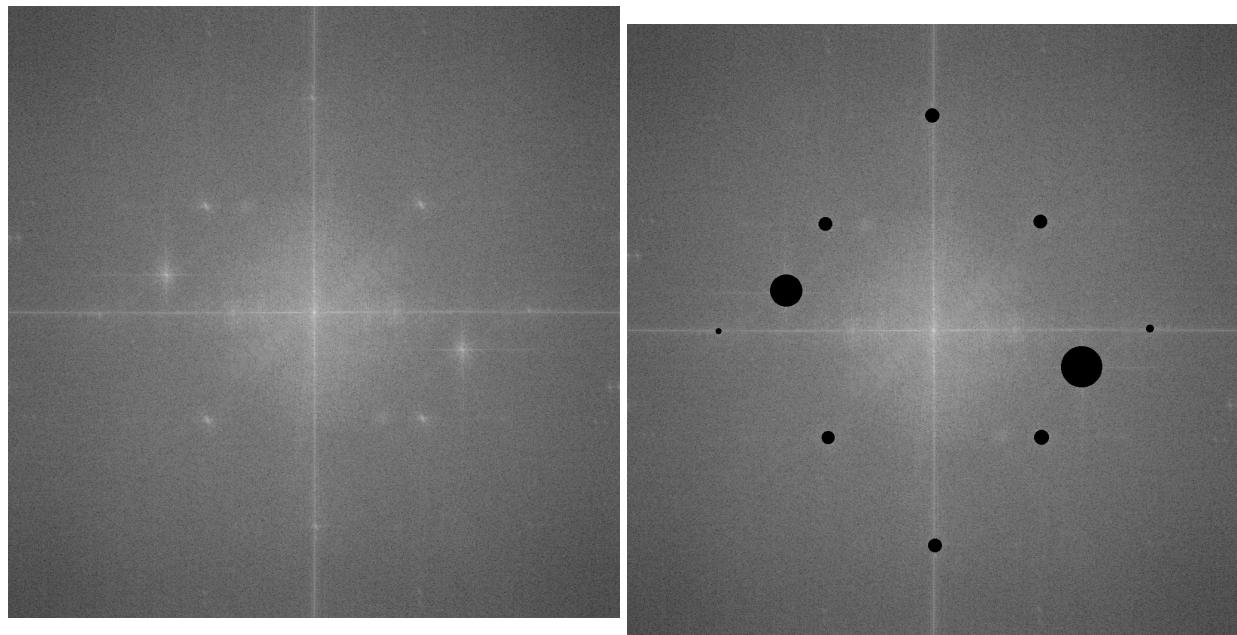
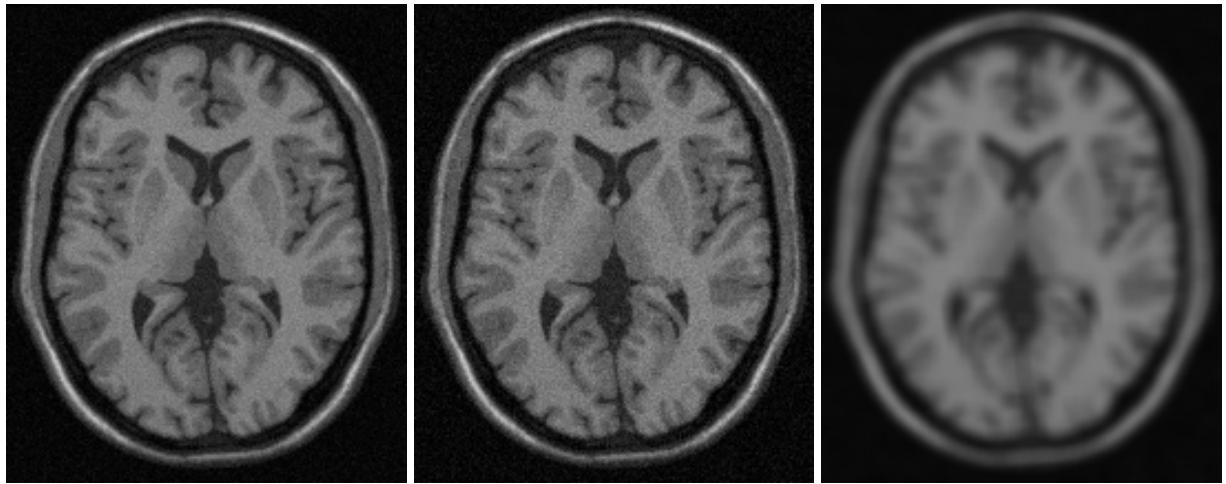


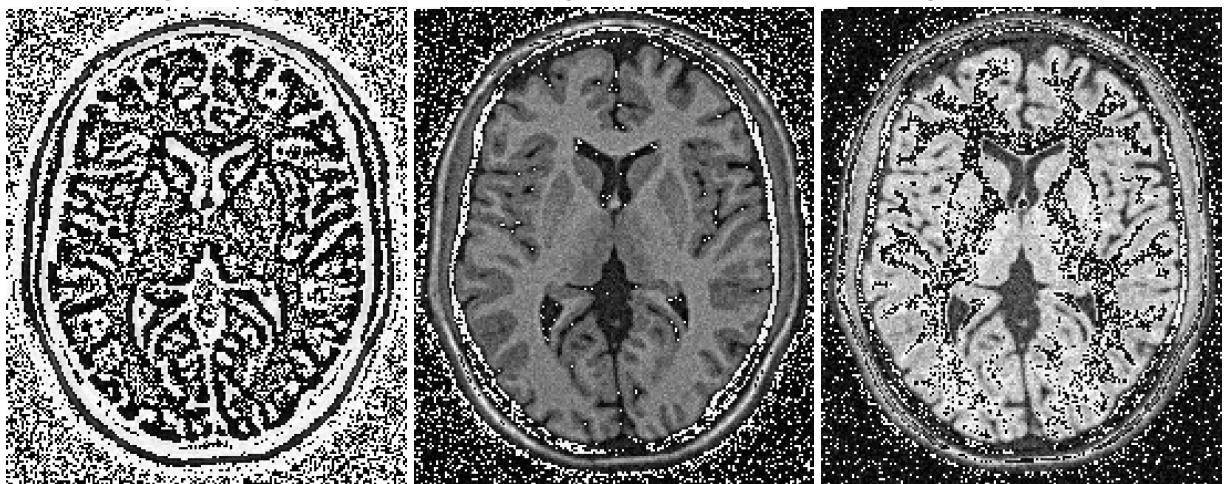
Figura 13: Transformada de Fourier rápida de Superman (a). Transformada de Fourier rápida sin las componentes periódicas. En el espacio de frecuencias las componentes periodicas se manifiestan como puntos de alta intensidad.



(a) ImagenA Original.

(b) ImagenA con ruido.

(c) ImagenA con ruido filtrada.

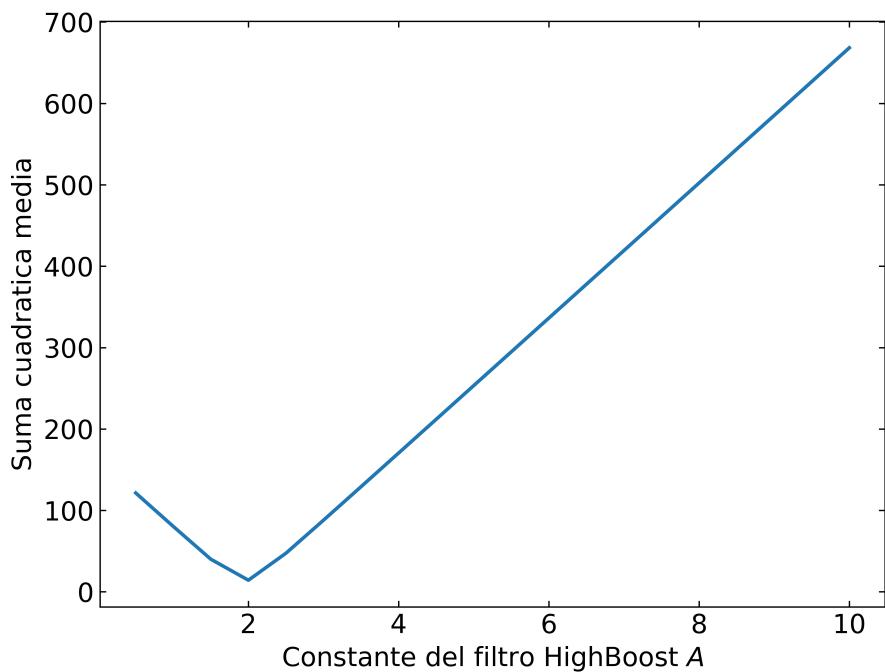


(d) Filtro Unsharp.

(e) Filtro Highboost con $A = 2$.

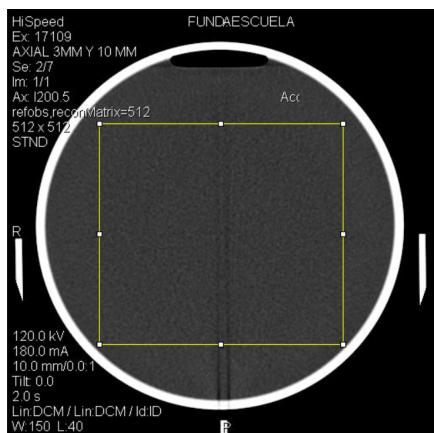
(f) Filtro Highboost con $A = 3$.

Figura 14: Filtraos Unsharp y Highboost aplicados a la imagenA con ruido gaussiano con desvio estandar $\sigma = 5$.

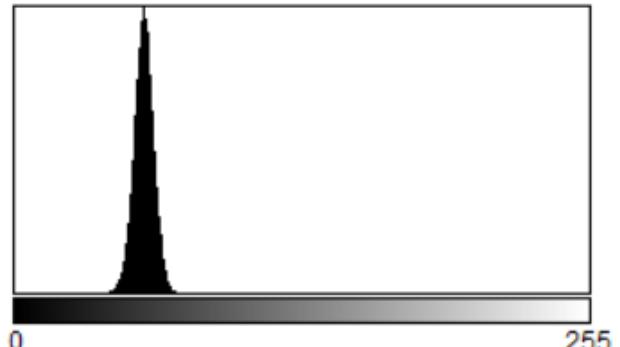


(a) ImagenA original junto con su histograma de intensidades.

Figura 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 .

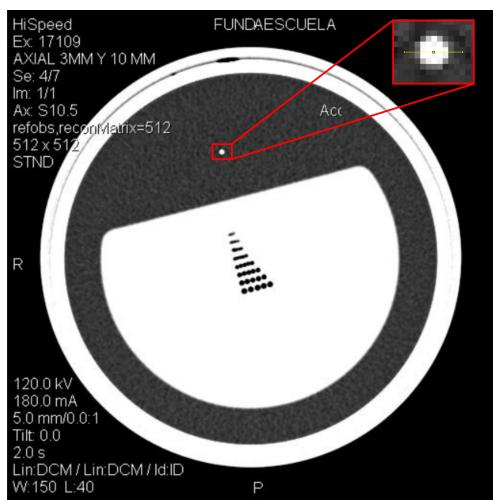


(a) Imagen con la ROI seleccionada.

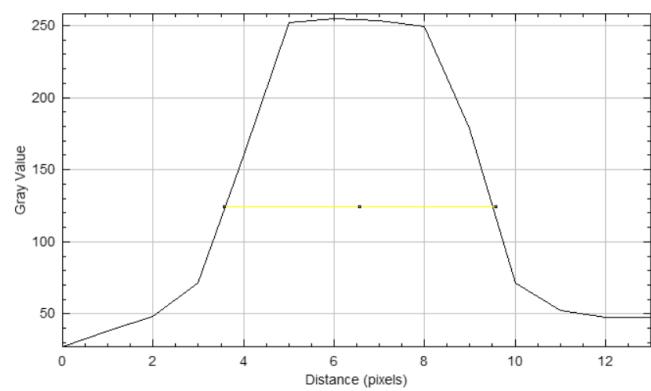


(b) Histograma de intensidades.

Figura 16: Estimación de la SNR para la imagen AAA0002.



(a) Imagen con la ROI seleccionada.



(b) Perfil de intensidad.

Figura 17: Estimación del FWHM para la imagen AAA0011.

A. Apéndice

```
import cv2
import sys
import numpy as np
import matplotlib.pyplot as plt
import os.path

def read_pgm_file(file_name):
    data_dir = os.path.dirname(os.path.abspath(__file__))
    # Test if file exists
    file_path = os.path.join(data_dir, file_name)
    assert os.path.isfile(file_path), 'file \'{0}\' does not exist'.format(file_path)

    img = cv2.imread(file_name, flags=cv2.IMREAD_GRAYSCALE)

    if img is not None:
        print('img.size: ', img.size)
    else:
        print('imread({0}) -> None'.format(file_path))

    return img

def save_img_hist(im, filename):
    vmin = np.amin(im)
    vmax = np.max(im)
    print("Intensity Min: {} Max:{}".format(vmin, vmax))

    L = vmax - vmin
    print("Number of Levels: {}".format(L))
    fig = plt.figure(figsize=(12,5))
    ax1 = plt.subplot(1, 2, 1)
    ax2 = plt.subplot(1, 2, 2)

    # imgplot = plt.imshow(im/np.amax(im))
    imgplot = ax1.imshow(im, cmap='gray', vmin=vmin, vmax=vmax)
    fig.colorbar(imgplot, ax=ax1)
    #ax1.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
    ax1.tick_params(axis='x', rotation=0, labelsize=15, color='black')
    ax1.tick_params(axis='y', labelsize=15, color='black')
    # cv2.imshow(infile, img)
    # cv2.waitKey(0)

    hist, bin_edges = np.histogram(im.ravel(), bins=L)
    ax2.bar(bin_edges[:-1], hist, alpha=1, color='b')
    ax2.set_xlabel(r"Intensidad de pixel", fontsize=15)
    ax2.set_ylabel(r"N mero de pxeles", fontsize=15)
    ax2.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
    ax2.tick_params(axis='x', rotation=0, labelsize=15, color='black')
    ax2.tick_params(axis='y', labelsize=15, color='black')
    ax2.set_xlim(-5, 260)
    #ax2.set_ylim(0, 1000)

    #x1 = rmin # Punto inicial en el eje x
    #x2 = rmax # Punto final en el eje x
    #y1 = 0 # Punto inicial en el eje y (puedes ajustar este valor seg n tu necesidad)
    #y2 = 255 # Punto final en el eje y (puedes ajustar este valor seg n tu necesidad)

    #Agrega la recta al segundo subplot (ax2)
    #ax2.plot([x1, x2], [y1, y2], color='r', linestyle='-', linewidth=2) # Trama la linea recta
```

```

fig.savefig(f"{filename}_hist.pdf")
fig.savefig(f"{filename}_hist.png", dpi=600)
#plt.show()

def save_img(im, filename):
    vmin = np.amin(im)
    vmax = np.max(im)
    print("Intensity Min: {}    Max:{}".format(vmin, vmax))

    L = vmax - vmin
    print("Number of Levels: {}".format(L))

    fig, ax = plt.subplots()
    imgplot = ax.imshow(im, cmap='gray', vmin=vmin, vmax=vmax)
    ax.axis('off')

    fig.savefig(f"{filename}.pdf", pad_inches = 0, bbox_inches='tight')
    fig.savefig(f"{filename}.png", dpi=600, pad_inches = 0, bbox_inches='tight')
    print("Size of image: {}".format(im.shape))
    #plt.show()

def SemilinearTrans_pgm_file(im, rmin, rmax):
    Imin = 0
    Imax = 255
    a = (Imax - Imin)/(rmax-rmin)
    b = Imin - a*rmin
    imout = a*im + b

    imout = np.where(imout < Imin, Imin, imout)
    imout = np.where(imout > Imax, Imax, imout)
    imout = imout.astype(int)
    return imout

def Equalize_pgm_file(im):
    imout = im
    vmin = np.amin(im)
    vmax = np.max(im)
    L = vmax - vmin
    hist, bin_edges = np.histogram(im.ravel(), bins=L)
    hist_norm = hist / np.sum(hist) #Normaliza el histograma

    suma = 0
    for i in range(0,im.shape[0]):
        for j in range(0,im.shape[1]):
            suma = 0
            for e in range(0,im[i,j]):
                suma += hist_norm[e]
            imout[i,j] = 255*suma

    imout = imout.astype(int)
    return imout

def Trans_binary(im):
    imout=np.where(im<128,255,0)
    imout = imout.astype(int)
    return imout

def Trans_exponential(im, gamma):
    imout=im
    imout=255*(imout/im.max())**gamma
    imout=imout.astype(int)
    return imout

def sustraction(im1, im2):

```

```

imout = im1 - im2
return imout

def interpolate_nn(im,f):
    Nx, Ny = im.shape[0],im.shape[1]
    Mx=int(f*Nx)
    My=int(f*Ny)
    imout=np.ndarray((Mx,My), dtype=int)

    for x in range(Mx):
        for y in range(My):
            imout[x][y]=im[round(x*Nx/Mx)%Nx][round(y*Ny/My)%Ny]

    return imout

def interpolate_bilinear(im, output_width, output_height):

    # Calculate scaling ratios
    old_height, old_width = im.shape[:2]
    x_ratio = old_width / output_width
    y_ratio = old_height / output_height

    # Create new image array
    new_img = np.zeros((output_height, output_width), dtype=np.uint8)

    # Perform bilinear interpolation
    for y in range(output_height):
        for x in range(output_width):
            x_original = x * x_ratio
            y_original = y * y_ratio
            x0 = int(x_original)
            y0 = int(y_original)
            x1 = min(x0 + 1, old_width - 1)
            y1 = min(y0 + 1, old_height - 1)

            Q11 = im[y0, x0]
            Q21 = im[y1, x0]
            Q12 = im[y0, x1]
            Q22 = im[y1, x1]

            x_weight = x_original - x0
            y_weight = y_original - y0

            R1 = Q11 * (1 - x_weight) + Q12 * x_weight
            R2 = Q21 * (1 - x_weight) + Q22 * x_weight

            P = R1 * (1 - y_weight) + R2 * y_weight

            new_img[y, x] = P

    return new_img

def high_boost(im,im_filtered,A):
    imout = A*im - im_filtered
    return imout

def unsharp(im,im_filtered):
    imout = im - im_filtered
    return imout

if __name__ == "__main__":
    if(len(sys.argv)<3):
        print("Usage: python read-write-pgm.py [infile.pgm] [outfile.pgm]")
        exit(1)

```

```

infile1 = sys.argv[1]
outfile1 = sys.argv[2]
#infile2 = sys.argv[3]
#outfile2 = sys.argv[4]

img1 = read_pgm_file(infile1)
#img2 = read_pgm_file(infile2)
im1 = np.array(img1)
#im2 = np.array(img2)

#Brute image
print("\n Brute image:")
print("Size of image 1: {}".format(im1.shape))
#print("Size of image 2: {}".format(im2.shape))
save_img_hist(im1,"ImageA_brute",0,150)

#Semilinear transformation for different rmin values
print("\n Semilinear transformation")
img = read_pgm_file(infile1)
im = np.array(img)
rmax = 150 # 25, 62 , 113 ,150
rmin = 113 # 0, 25 , 62 , 113
imout = SemilinearTrans_pgm_file(im,rmin,rmax)
print("Size of image: {}".format(imout.shape))
save_img_hist(imout,"ImageA_"+str(rmin)+"_"+str(rmax))
save_img(imout,"ImageA_"+str(rmin)+"_"+str(rmax))

#Equalize image
print("\n Equalize image")
imout = Equalize_pgm_file(im1)
save_img_hist(imout,"ImageA_EQ")
save_img(imout,"ImageA_EQ")
#cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])

#Transformacion binaria
print("\n Transformacion binaria")
imout = Trans_binary(im1)
save_img(imout,f"ImageA_binary")
#cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])

#Transformacion exponencial
print("\n Transformacion exponencial")
gamma = 0.5
imout = Trans_exponential(im1,gamma)
save_img(imout,f"ImageA_exp-gamma={gamma}")
#cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])

#Interpolate nn
f = 0.25
print("\n Interpolate NN")
imout = interpolate_nn(im1,f)
save_img(imout,f"Interpolate_nn_f={f}")

#Interpolate bilinear
#f = 0.25
print("\n Interpolate bilinear")
imout = interpolate_bilinear(im1, int(f* im1.shape[0]), int(f* im1.shape[1]))
save_img(imout,f"Interpolate_bilinear_f={f}")
cv2.imwrite(outfile1,imout,[cv2.IMWRITE_PXM_BINARY,0])

#Sustraction
print("\n Sustraction")
imout = sustraction(im1,im2)
save_img(imout,f"sustraction_exp_gamma=0.5")

```

```

cv2.imwrite(outfile1 ,imout ,[ cv2.IMWRITE_PXM_BINARY,0])

#Unsharp filter
print("\n Unsharp filter")
imout1 = unsharp(im1,im2)
imout1 = np.array(imout1)
print("Size of image 1: {}".format(imout1.shape))
save_img(imout1,"unsharp")

#HighBoost filter
print("\n HighBoost filter ")
A=3
imout2 = high_boost(im1,im2,A)
imout2 = np.array(imout2)
print("Size of image 2: {}".format(imout2.shape))
save_img(imout2,f"highboost_A={A}")

#High Boost filter plot
print("\n High Boost filterplot")
A = np.arange(0.5, 10.5, 0.5)
S = ([])

for a in A:
    imout2 = high_boost(im1,im2,a)
    im_sust = sustraction(imout2,im1)
    matriz_cuadrada = np.square(im_sust)
    suma_total = np.sum(matriz_cuadrada)
    media = np.sqrt(suma_total / im_sust.size)
    S = np.append(S, media)
    #imout2 = np.array(imout2)
    #print("Size of image 1: {}".format(imout2.shape))
    #imout2 = Equalize_pgm_file(imout2)
    #save_img(imout2,f"highboost_A={A}")
    #cv2.imwrite(outfile2 ,imout2 ,[ cv2.IMWRITE_PXM_BINARY,0])

fig , ax = plt.subplots(figsize=(8, 6))
ax.plot(A, S, linewidth=2)
ax.set_xlabel(f"Intensidad del filtro HighBoost $A$", fontsize=15)
ax.set_ylabel(f"Suma cuadratica media", fontsize=15)
ax.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
ax.tick_params(axis='x', rotation=0, labelsize=15, color='black')
ax.tick_params(axis='y', labelsize=15, color='black')
#ax.set_xlim(-5, 260)
#ax2.set_ylim(0, 1000)
fig.savefig(f"Highboost_plot.pdf", pad_inches = 0,bbox_inches='tight ')
fig.savefig(f"Highboost_plot.png", dpi=600, pad_inches = 0, bbox_inches='tight ')

```