

Redes Neuronales y Técnicas de Procesamiento Avanzado de Imágenes para el Diagnóstico Automatizado de Tumores Mamarios

Marlene Fuhr
Sofía Goldszer

20 de Diciembre 2024

1. Introducción

El diagnóstico temprano de tumores mamarios es crucial para aumentar las tasas de supervivencia y mejorar los resultados clínicos en pacientes con cáncer de mama. Sin embargo, este proceso suele depender de la experiencia subjetiva del especialista y de herramientas tradicionales de análisis, que pueden presentar limitaciones en términos de precisión y tiempo. En este contexto, la aplicación de modelos de redes neuronales y técnicas avanzadas de procesamiento de imágenes representa un avance significativo hacia la automatización y mejora del diagnóstico. Las redes neuronales, con su capacidad para identificar patrones complejos en grandes volúmenes de datos, se han convertido en herramientas clave para clasificar imágenes médicas con alta precisión. Por otro lado, las técnicas avanzadas de procesamiento de imágenes permiten extraer características relevantes, reducir ruido y optimizar la calidad de las imágenes para su análisis, aumentando la confiabilidad de los modelos predictivos. El desarrollo de un sistema automatizado que combine estas tecnologías no solo promete aumentar la eficiencia y precisión en el diagnóstico, sino también reducir la carga de trabajo de los especialistas y mejorar el acceso a diagnósticos de calidad en regiones con recursos limitados. Este informe explora la implementación de estas tecnologías, destacando su potencial para transformar el diagnóstico del cáncer de mama y establecer un nuevo estándar en la práctica médica.

Marco Teórico: Problemática

El cáncer de mama constituye una de las principales causas de mortalidad entre las mujeres a nivel global, representando un desafío significativo para los sistemas de salud y la sociedad en general. Este tipo de cáncer se desarrolla cuando las células del tejido mamario comienzan a proliferar de manera descontrolada, superando los mecanismos normales de regulación celular. Este crecimiento anormal puede dar lugar a la formación de tumores malignos, los cuales no solo afectan el tejido mamario, sino que también tienen el potencial de invadir estructuras vecinas y diseminarse a otras partes del cuerpo a través de un proceso conocido como metástasis.

El cáncer de mama es altamente heterogéneo, lo que implica que su comportamiento biológico, su agresividad y su respuesta a los tratamientos pueden variar considerablemente entre pacientes. Esta variabilidad está influenciada por múltiples factores, incluidos el tipo histológico del tumor, su estadio en el momento del diagnóstico, la presencia de receptores hormonales y otras características moleculares. Por ejemplo, mientras algunos subtipos de cáncer de mama son más agresivos y requieren tratamientos intensivos, otros pueden ser menos invasivos y responder bien a terapias más conservadoras.

La detección temprana del cáncer de mama es esencial para mejorar las tasas de supervivencia, ya que los tratamientos suelen ser más efectivos cuando la enfermedad se encuentra en etapas iniciales. Diversos estudios han demostrado que el diagnóstico precoz no solo incrementa las posibilidades de curación, sino que también reduce la necesidad de tratamientos invasivos y mejora la calidad de vida de los pacientes. Los métodos de detección actuales, como la mamografía, están diseñados para identificar signos tempranos de la enfermedad, tales como masas anormales, microcalcificaciones u otras irregularidades en el tejido mamario. Sin embargo, estos métodos no siempre son concluyentes, lo que subraya la necesidad de herramientas diagnósticas más precisas y eficientes.

Estado del Arte

En los últimos años, el procesamiento avanzado de imágenes y la inteligencia artificial (IA) han transformado significativamente la forma en que se analizan las mamografías. Estas técnicas permiten mejorar la precisión y eficiencia del diagnóstico del cáncer de mama, abordando algunas de las limitaciones de los métodos tradicionales.

Los algoritmos de procesamiento de imágenes pueden realzar características relevantes, como bordes de lesiones y microcalcificaciones, permitiendo una interpretación más clara por parte de los especialistas. Además, herramientas basadas en aprendizaje profundo, como las redes neuronales convolucionales (CNN), han demostrado un desempeño sobresaliente en la clasificación de tejidos como benignos o malignos. Estas redes se entrena con grandes conjuntos de datos de mamografías anotadas, logrando niveles de precisión comparables o superiores a los de los radiólogos expertos en algunos escenarios.

Entre las arquitecturas más prominentes, ResNet (Redes Residuales) ha demostrado ser particularmente efectiva en tareas de clasificación y segmentación de imágenes médicas, incluidas las mamografías. Introducida por He et al., ResNet se basa en bloques residuales que permiten entrenar redes profundas al mitigar el problema

del desvanecimiento del gradiente. Esto se logra mediante conexiones de atajo que saltan una o más capas, facilitando el flujo de información y gradientes a través de la red. En el contexto de mamografías, ResNet se ha utilizado para identificar lesiones malignas con alta precisión, logrando una detección automatizada confiable y consistente.

Investigaciones recientes han combinado ResNet con técnicas como el aprendizaje por transferencia, aprovechando modelos preentrenados en grandes conjuntos de datos generales (como ImageNet) y ajustándolos para tareas específicas en mamografías. Esto ha demostrado ser especialmente útil en escenarios donde los conjuntos de datos médicos son limitados, permitiendo entrenar modelos robustos con menos ejemplos etiquetados. Además, se han explorado variantes como ResNet-50 y ResNet-101 para mejorar el equilibrio entre la profundidad de la red y el rendimiento computacional, adaptándose a entornos clínicos con restricciones de hardware.

Dataset

Las imágenes médicas son fundamentales para la detección, diagnóstico y seguimiento del cáncer de mama. Entre las técnicas más utilizadas se encuentran la mamografía, el ultrasonido, la resonancia magnética (RM) y la tomosíntesis mamaria. El dataset que elegimos utiliza imágenes mamográficas. La mamografía es el estándar de oro para la detección del cáncer de mama. Utiliza rayos X de baja energía para capturar imágenes bidimensionales de las mamas. Es particularmente efectiva para identificar microcalcificaciones y masas sospechosas. Sin embargo, su sensibilidad puede verse reducida en mujeres con tejido mamario denso.

Trabajamos con 120 imágenes del dataset público llamado Mammographic Image Analysis(MIAS) al cual se puede acceder con el siguiente link: DRIVE <http://peipa.essex.ac.uk/pix/mias/>

Objetivos

Calcificaciones. Las mujeres mayores pueden ocasionalmente encontrar pequeñas manchas de calcio en sus senos. Estas manchas se denominan microcalcificaciones, que, debido a su pequeño tamaño, no son palpables. Sin embargo, en las imágenes de mamografía, aparecen como manchas pequeñas y brillantes. En la mayoría de los casos, son benignas; sin embargo, su observación en ciertos patrones puede causar preocupación. Por ejemplo, en algunos casos, crecen en forma de un grupo o una línea (con un crecimiento similar a un racimo o lineal), lo cual puede ser un signo de cáncer.

Masa o Tumor. Los senos están compuestos por una glándula (parénquima) y un tejido ductal; y las masas, que se describen como bultos que ocupan espacio, pueden ocultarse alrededor del tejido parenquimatoso de los senos. En consecuencia, puede ser de alguna manera difícil distinguir entre el área normal y la anormal.

Entonces, el objetivo del presente trabajo es desarrollar y desarrollar un algoritmo que pueda detectar y clasificar los tumores de mamas de la manera más eficiente posible utilizando técnicas de Machine Learning y redes neuronales convolucionales. Para mejorar la eficiencia de estas técnicas, se hace uso de técnicas de procesamiento avanzado de imágenes para hacer preprocesamiento, extracción de características, mejoramiento y segmentación antes de proceder con las técnicas de clasificación.

2. Materiales y Métodos

2.1. Importar librerías y lectura de imágenes

Código a mano código. Se puede acceder a los códigos de librerías utilizadas y lectura de las 120 imágenes en la sección 1 del anexo.

```
#Lectura de imágenes
carpetas=[]
for i in range(0,120):
    if i<9:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TPfinal_imagenes/mdb0{i+1}.pgm')
        carpetas.append(carpeta)
    elif i<99:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TPfinal_imagenes/mdb0{i+1}.pgm')
        carpetas.append(carpeta)
    else:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TPfinal_imagenes/mdb{i+1}.pgm')
```

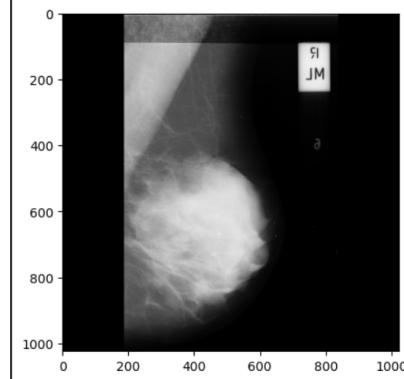
```

    carpetas.append(carpeta)

plt.imshow(carpetas[1], cmap="gray")

```

2.2. Preprocesamiento



Como podemos ver, algunas imágenes están orientadas a la derecha y otras a la izquierda. Primero, haremos que todas las imágenes estén orientadas a la derecha.

Voltear todas las imágenes hacia la derecha.

```

def right_orient_mammogram(image):
    left_nonzero = cv2.countNonZero(image[:, 0:int(image.shape[1]/2)])
    right_nonzero = cv2.countNonZero(image[:, int(image.shape[1]/2):])

    if(left_nonzero < right_nonzero):
        image = cv2.flip(image, 1)

    return image

flippedImg = np.zeros((120,1024,1024))

for i in range(0,120):
    flippedImg[i] = right_orient_mammogram(cv2.cvtColor(carpetas[i], cv2.COLOR_BGR2GRAY))

```

Volteo manualmente:

Pero algunas imágenes aún no se han volteado, por lo que lo estamos haciendo manualmente. (Podemos mejorar nuestro algoritmo para eliminar este volteo manual).

```

def forceFlip(img):
    img = cv2.flip(img,1)
    return img

flippedImg[35]=forceFlip(flippedImg[35])
flippedImg[75]=forceFlip(flippedImg[75])
flippedImg[101]=forceFlip(flippedImg[101])

```

Para ver las 120 imágenes volteadas dirigirse a la sección 2 del anexo.

Recorte del lado izquierdo de la imagen Recortaremos el lado izquierdo de las imágenes. Todas las imágenes deberían tener un tamaño de 1024x800 después de este paso.

Algoritmo para cropLeft()

Recortar el lado izquierdo. Si el ancho de la imagen resultante es igual a 800, devolver la imagen. Si el tamaño de la imagen resultante es menor que 800, añadir píxeles en el lado derecho para hacerla de 800 píxeles de ancho. Si el ancho de la imagen resultante es mayor a 800, recortar los píxeles del lado derecho para ajustarla a 800 píxeles de ancho.

```

def cropLeft(img):
    mini = 0

```

```

pix = 0

for i in range(1024):
    if img[10][i]>10:
        mini = i
        break
newimg=np.ones((1024,800))*pix
#    print(min, img[10][200])
diff = 1024-mini
if diff<800:
    newimg[:,diff] = img[:,mini:]
#    print('in cropleft : shape ', newimg.shape)
elif diff>800:
    newimg = img[:,mini:(mini+800)]
elif diff==800:
    newimg = img[:,mini:]
#    print(newimg.shape)

#    plt.imshow(newimg)
return newimg

leftCropped = np.zeros((120,1024,800))
for i in range(0,120):
    imgforcropleft = flippedImg[i]
    leftCropped[i] = cropLeft(imgforcropleft)

```

Sacar artefactos:

Artefactos son las etiquetas en las mamografías (Los artefactos pueden observarse en la imagen de referencia anterior). Estos artefactos pueden generar resultados negativos, por lo que es importante eliminarlos.

Algoritmo para removeArt()

Identificar el inicio de la parte brillante y almacenarlo en la variable start. Identificar el final de la parte brillante y almacenarlo en la variable end. Establecer todos los píxeles a la derecha de end en cero. Devolver la imagen.

```

def removeArt(img):
    thresh = 30
#    thresh = 10
    minPix = 40

#    print(start)

    newim = np.zeros((1024,800))
    newim = img.copy()
    for i in range(1024):
        start = 0
        for k in range(800):
            if img[100][k] != 0:
                start = k
                break
        for j in range(start , 800-minPix):
#            for j in range(768-minPix):
                ar = newim[i][j:j+minPix]
                if (all(x < thresh for x in ar)):
                    newim[i][j:] = 0
                    break
#    plt.imshow(newim)
    return newim

artImg = np.zeros((120,1024,800))
for i in range(0,120):
    artImg[i] = removeArt(leftCropped[i])

```

Recortando la parte superior Algunas imágenes tienen todos los valores en cero en las filas superiores, por lo que las recortaremos.

```

def cropTop(img):
    newim = np.zeros((1024,800))

```

```

newim[:1023,:,:]=img[1,:,:]
return newim

topCropped = np.zeros((120,1024,800))
for i in range(0,120):
    topCropped[i] = cropTop(artImg[i])

```

Recorte individual de algunas imágenes Ahora, algunas imágenes tienen más filas con valores cero en la parte superior, por lo que se eliminan manualmente.

```

new1 = np.zeros((1024,800))
new1[:934,:,:] = topCropped[1][90,:,:]
topCropped[1] = new1

new1 = np.zeros((1024,800))
new1[:1002,:,:] = topCropped[10][22,:,:]
topCropped[10] = new1

for i in range(0,120):
    topCropped[i]= cv2.normalize(
        topCropped[i],
        None,
        alpha=0,
        beta=255,
        norm_type=cv2.NORM_MINMAX ,
        dtype=cv2.CV_32F ,
    )
    topCropped[i]= topCropped[i].astype("uint8")

def robust_convert_to_uint8(image):
    """
    Convierte cualquier matriz de imagen a tipo uint8, asegurando que los valores estén en el rango [0, 255]

    Args:
        image (np.ndarray): Imagen de entrada.

    Returns:
        np.ndarray: Imagen convertida a uint8.
    """
    # Verificar que sea una matriz NumPy
    if not isinstance(image, np.ndarray):
        raise TypeError("La entrada debe ser una matriz NumPy.")

    # Verificar que la matriz no esté vacía
    if image.size == 0:
        raise ValueError("La imagen de entrada está vacía.")

    # Normalizar si los valores son mayores que 255 o menores que 0
    if image.dtype != np.uint8:
        min_val, max_val = image.min(), image.max()
        print(f"Valores mínimos y máximos antes de normalizar: {min_val}, {max_val}")

        if min_val < 0 or max_val > 255:
            # Normalizar al rango [0, 255]
            image = (image - min_val) / (max_val - min_val) * 255.0
            print("La imagen fue normalizada al rango [0, 255].")

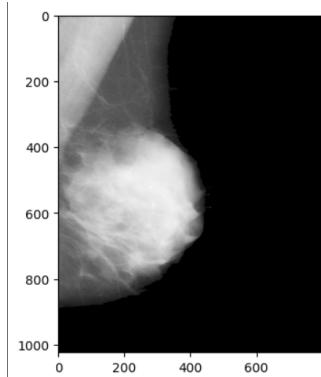
    # Convertir a tipo uint8
    image = np.clip(image, 0, 255).astype(np.uint8)
    print("Imagen convertida a uint8.")

    return image

carpetas1= np.zeros((120,1024,800))
for i in range(0,120):
    carpetas1[i] = cropTop(artImg[i])

for i in range(0,120):
    carpetas1[i] = robust_convert_to_uint8(topCropped[i])

```



Para ver las 120 imágenes con los artefactos removidos en la sección 3 del anexo.

2.3. Estimar el ruido

Con la siguiente función de rectángulo, analizamos una región de interés ROI homogénea seleccionada en cada una de las imágenes de la carpeta.

```
def get_rectangle_intensities(image, rect_start, B, H):
    rect_end = (rect_start[0] + B, rect_start[1] + H) #esquina inferior
    rect_intensities = image[rect_start[1]:rect_end[1], rect_start[0]:rect_end[0]]

    if image.dtype != np.uint8:
        image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    if len(image.shape) == 2: # Escala de grises
        image_with_rect = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    else:
        image_with_rect = image.copy()

    cv2.rectangle(image_with_rect, rect_start, rect_end, (0, 255, 0), 2)
    return image_with_rect, rect_intensities
```

A continuación, elegir un vértice del rectángulo, la base y la altura. Aplicando las funciones realizadas anteriormente, obtenemos la matriz de intensidades de píxeles dentro del rectángulo elegido, el cual es posicionado en el fondo negro de la imagen donde no se encuentra la mama.

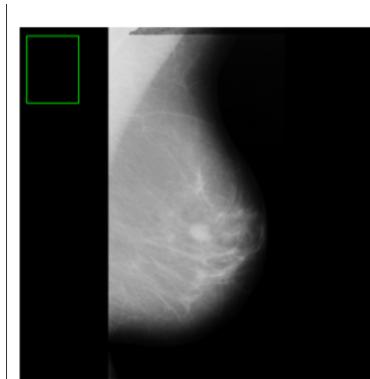
```
image_with_rect = [np.empty([1024,800])]
for i in range(0,120):
    image_with_rect.append(np.empty([1024,800]))

rect_intensities = [np.empty([1024,800])]
for i in range(0,120):
    rect_intensities.append(np.empty([1024,800]))

for i in range(0,120):
    rect_start = (20, 25)
    B, H = 150, 195
    image_with_rect[i], rect_intensities[i] = get_rectangle_intensities(carpetas[i], rect_start, B, H)
    #print("Imagen:",i+1,"Matriz de intensidades del rectángulo:\n", rect_intensities)

plt.figure(figsize=(10,520))
for i in range(0,120):
    plt.subplot(120,1, i+1)
    plt.imshow(image_with_rect[i], cmap="gray")
    plt.axis('off')

plt.show()
```



Podemos calcular las estadísticas dentro de una región de interés o dentro de toda la imagen. En este caso, voy a calcular las estadísticas de las regiones de interés. A continuación, estimamos el ruido basado en el histograma y calculando la varianza en la región de interés homogénea.

```
def calculate_statistics(image):

    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    mean_value = np.mean(image)
    variance_value = np.var(image)
    return hist, mean_value, variance_value

hist_rect = [np.empty([195,150])]
for i in range(0,120):
    hist_rect.append(np.empty([195,150]))
mean_rect = [np.empty([195,150])]
for i in range(0,120):
    mean_rect.append(np.empty([195,150]))
var_rect = [np.empty([195,150])]
for i in range(0,120):
    var_rect.append(np.empty([195,150]))
for i in range(0,120):
    hist_rect[i], mean_rect[i], var_rect[i] = calculate_statistics(rect_intensities[i])

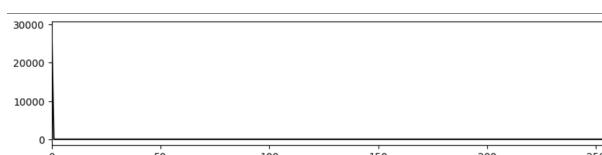
for i in range(0,120):
    print("Imagen:",i+1,"varianza:",var_rect[i],"media:",mean_rect[i])
```

1. Imagen: 1, varianza: 0.0, media: 0.0
2. Imagen: 2, varianza: 0.0, media: 0.0
-
3. Imagen: 119, varianza: 0.0, media: 0.0
4. Imagen: 120, varianza: 0.0, media: 0.0

Para ver la lista completa de las estadísticas de las 120 imágenes en la sección 4 del anexo.

```
plt.figure(figsize=(10,320))
for i in range(0,120):
    plt.subplot(120,1, i+1)
    plt.plot(hist_rect[i], color='black')
    plt.xlim([0, 256])

plt.show()
```



```
def histogram_noise_analysis(roi):
    hist = cv2.calcHist([roi], [0], None, [256], [0, 256])
```

```

hist = hist / hist.sum()
mean = np.mean(roi)
std_dev = np.std(roi)
dma = np.mean(np.abs(roi - mean))
# Calcular la frecuencia del valor de intensidad más frecuente (modo)
mode_val = np.argmax(hist)
stats = {
    'Media_(ROI)': mean,
    'Desviación_Estándar_(': std_dev,
    'Desviación_Media_Absoluta_(DMA)': dma,
    'Modo_del_Histograma': mode_val
}
return stats, hist

plt.figure(figsize=(10,320))
for i in range(0,120):
    stats, hist=histogram_noise_analysis(rect_intensities[i])
    print("Estadísticas adicionales de la imagen n°:",i+1)
    for key, value in stats.items():
        print(f"{key}: {value}")

```

Para ver los histogramas completos de las 120 imágenes ir a la sección 5 del anexo.

1. Estadísticas adicionales de la imagen número: 1

- Media (ROI): 0.0
- Desviación Estándar (σ): 0.0
- Desviación Media Absoluta (DMA): 0.0
- Modo del Histograma: 0

Estadísticas adicionales de las 120 imágenes se encuentran en la sección 4 del anexo. Como se puede ver las imágenes no parecen tener ruido relevante en las imágenes.

A continuación, estimamos el ruido a partir del espectro de potencia basado en la transformada de fourier. El espectro de potencia puede revelar la presencia de componentes de alta frecuencia que corresponderían al ruido.

Estimar el ruido con la transformada de fourier con la imagen completa:

```

def fourier_transform(image):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    power_spectrum = np.abs(fshift) ** 2
    return power_spectrum, fshift

def estimate_noise_fourier(image):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20 * np.log(np.abs(fshift))
    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2
    mask = np.ones((rows, cols), np.uint8)
    r = 30
    center_circle = np.ogrid[:rows, :cols]
    mask_area = (center_circle[0] - crow) ** 2 + (center_circle[1] - ccol) ** 2 <= r**2
    mask[mask_area] = 0
    high_freq_magnitudes = magnitude_spectrum[mask == 1]
    noise_estimate = np.mean(high_freq_magnitudes)
    return magnitude_spectrum, noise_estimate

for i in range(0,120):
    #magnitude_spectrum, noise_estimate = estimate_noise_fourier(cv2.cvtColor(carpetas[i].copy(), cv2.COLOR_
    magnitude_spectrum, noise_estimate = estimate_noise_fourier(topCropped[i])
    plt.figure(figsize=(30, 420))
    plt.subplot(120, 1, 1)
    plt.title('Espectro_de_Fourier')
    plt.imshow(magnitude_spectrum, cmap='gray')
    plt.axis('off')
    plt.subplot(120, 2, 2)

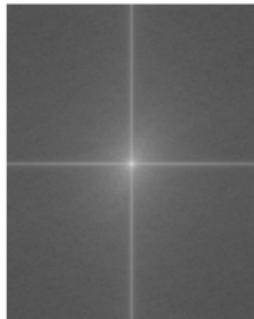
```

```

plt.text(0.5, 0.5, f'Ruido Estimado:{noise_estimate:.0f}', fontsize=12, ha='center')
plt.axis('off')
plt.show()

```

Espectro de Fourier



Ruido Estimado: 140

Las estimaciones de ruido para las 120 imágenes se pueden visualizarse en la sección 6 del anexo.

```

pip install PyWavelets

import pywt
def wavelet_noise_estimation(image, wavelet='haar', level=1):
    image = image.astype(np.float32)
    # Aplicar la transformada wavelet discreta
    coeffs = pywt.wavedec2(image, wavelet=wavelet, level=level)
    LH, HL, HH = coeffs[1]
    sigma_LH = np.std(LH)
    sigma_HL = np.std(HL)
    sigma_HH = np.std(HH)
    noise_estimation = np.mean([sigma_LH, sigma_HL, sigma_HH])

    reconstructed_image = pywt.waverec2(coeffs, wavelet)
    reconstructed_image = np.clip(reconstructed_image, 0, 255).astype(np.uint8)

    return noise_estimation, reconstructed_image

for i in range(0,120):
    noise_level, transformed_image = wavelet_noise_estimation(carpetas1[i], wavelet='haar', level=1)
    print(f"Imagen {i+1}\nEstimación del nivel de ruido:{noise_level}")

```

1. Imagen número: 1
Estimación del nivel de ruido: 1.8689807653427124
2. Imagen número: 2
Estimación del nivel de ruido: 1.802535057067871
-
3. Imagen número: 119
Estimación del nivel de ruido: 2.1040849685668945
4. Imagen número: 120
Estimación del nivel de ruido: 2.068614959716797

La tabla completa se puede ver en la sección 7 del anexo.

2.4. Denoising

Transformada Wavelet

```

# Parámetros del ruido ya calculados de la imagen
varianza_ruido = 1 # ejemplo de varianza conocida
#img=cv2.cvtColor(carpetas[2].copy(), cv2.COLOR_RGB2GRAY)
img=carpetas1[2]
# Calcular sigma a partir de la varianza existente del ruido

```

```

sigma = np.sqrt(varianza_ruido)

# Realizar la transformada wavelet discreta (DWT)
coeffs = pywt.wavedec2(img, 'haar', level=2)

# Extraer coeficientes
LL2, (LH2, HL2, HH2), (LH1, HL1, HH1) = coeffs

# Calcular el umbral utilizando la varianza existente del ruido
threshold = sigma * np.sqrt(2 * np.log2(img.size))
print(f"Threshold calculado: {threshold}")

# Umbralizar cada sub-banda utilizando soft thresholding
def soft_thresholding(coeffs, threshold):
    return pywt.threshold(coeffs, threshold, mode='soft')

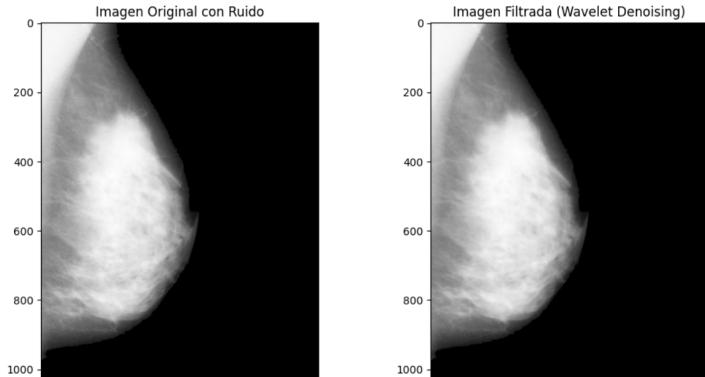
LL2_thresh = soft_thresholding(LL2, threshold)
LH2_thresh = soft_thresholding(LH2, threshold)
HL2_thresh = soft_thresholding(HL2, threshold)
HH2_thresh = soft_thresholding(HH2, threshold)

# Reconstruir la imagen a partir de los coeficientes umbralizados
coeffs_thresh = [LL2_thresh, (LH2_thresh, HL2_thresh, HH2_thresh), (LH1, HL1, HH1)]
denoised_img = pywt.waverec2(coeffs_thresh, 'haar')
denoised_img = np.clip(denoised_img, 0, 255)

# Mostrar resultados
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Imagen Original con Ruido')
plt.imshow(img, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Imagen Filtrada (Wavelet Denoising)')
plt.imshow(denoised_img, cmap='gray')
plt.show()
transformed_image1 = denoised_img

```

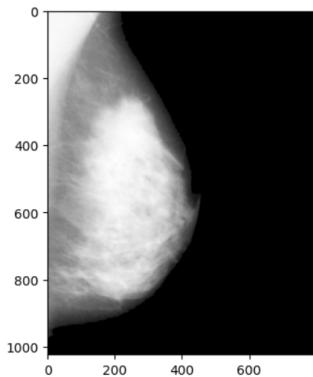


Filtro promediador

```

promediador1 = cv2.blur(carpetas1[2], (3, 3)) # Filtro promediador con kernel de 3x3
plt.imshow(promediador1, cmap='gray')

```



```

def calculate_snr(image, noise):
    signal_power = np.mean(image**2)
    noise_power = np.mean(noise**2)
    snr = abs(10 * np.log10(signal_power / noise_power))
    return snr

def calculate_psnr(original, noisy):
    mse = np.mean((original - noisy) ** 2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr = 10 * np.log10((max_pixel ** 2) / mse)
    return psnr

snr_promediador = calculate_snr(topCropped[2], promediador1)
psnr_promediador = calculate_psnr(topCropped[2], promediador1)
print("Imagen:")
print(f"PSNR (promediador): {psnr_promediador} dB")
print(f"SNR (promediador): {snr_promediador} dB")

wavelet1_r=transformed_image1
snr_wavelet = calculate_snr(topCropped[2], wavelet1_r)
psnr_wavelet = calculate_psnr(topCropped[2], wavelet1_r)
print("Imagen:")
print(f"PSNR (wavelet): {psnr_wavelet} dB")
print(f"SNR (wavelet): {snr_wavelet} dB")

```

1. Imagen:

PSNR (promediador): 43.6839861797788 dB
 SNR (promediador): 0.00049508421146909 dB
 PSNR (wavelet): 45.08653234473027 dB
 SNR (wavelet): 0.07186126112193565 dB

Decidimos usar filtro promediador aunque los resultados sean muy semejantes. Las 120 imágenes con el filtro promediador se encuentran en la sección 8 en el anexo.

```

promediador = [] # Crear una lista para almacenar los resultados

for i in range(0, 120):
    promediadores = cv2.blur(carpetas1[i], (3, 3))
    promediador.append(promediadores)

```

Utilizamos la técnica de CLAHE para mejorar el contraste

```

def clahe(img, clip=2.0, tile=(8, 8)):
    img = cv2.normalize(
        img,
        None,
        alpha=0,
        beta=255,
        norm_type=cv2.NORM_MINMAX,
        dtype=cv2.CV_32F,
    )
    img_uint8 = img.astype("uint8")

```

```

clahe_create = cv2.createCLAHE(clipLimit=clip, tileGridSize=tile)
clahe_img = clahe_create.apply(img_uint8)

return clahe_img

enhancedImg = np.zeros((120,1024,800))

for i in range(0,120):
    enhancedImg[i] = clahe(promediador[i])

plt.imshow(enhancedImg[0], cmap='gray')

```

Imagen con filtro promediador

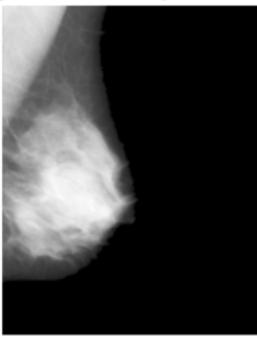


Imagen con CLAHE



Las 120 imágenes con CLAHE se encuentran en la sección 9 del anexo para visualizar libremente.

2.5. Sacar el músculo pectoral: Canny edge y transformada Hough

Eliminar el pectoral Esta parte es muy crucial.

Los músculos pectorales aparecen en las esquinas superiores izquierdas de las mamografías. Estos músculos tienen la misma densidad que los tejidos cancerosos, por lo que es necesario eliminarlos. Dado que estos músculos tienen la misma densidad que las células cancerígenas, resulta un poco complicado eliminarlos. Cada paso se realiza como una función separada. Los algoritmos de cada paso se encuentran junto con sus respectivas funciones.

```

merg = np.zeros((120,1024,800))

i = 0
start = time.time()
for image in enhancedImg:
    # Verifica si la imagen no est en uint8 y convi rtela
    if image.dtype != np.uint8:
        image = (image * 255).astype(np.uint8) if image.max() <= 1 else image.astype(np.uint8)

    # Crear kernel
    kernel = np.ones((120, 120), np.uint8)

    # Operaciones morfol gicas
    erosion = cv2.erode(image, kernel, iterations=1)
    dilation = cv2.dilate(erosion, kernel, iterations=1)

    # Aseg rate de que la m scara sea uint8
    if dilation.dtype != np.uint8:
        dilation = dilation.astype(np.uint8)

    # Aplicar bitwise_and con la m scara
    merged = cv2.bitwise_and(image, image, mask=dilation)
    merg[i] = merged

    i += 1

end = time.time()
print(f"Tiempo total:{end - start}segundos")

```

Operaciones morfológicas Tiempo total: 2.1897337436676025 segundos

Transformada Hough y Canny Edges

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab as pylab
from skimage import io
from skimage import color

from skimage.feature import canny
from skimage.filters import sobel
def apply_canny(image):
    sigma = 0.5 # Ajusta el sigma para suavizar m s/menos
    canny_img = canny(image, sigma=sigma)
    return sobel(canny_img)

from skimage.transform import hough_line, hough_line_peaks

def get_hough_lines(canny_img):
    h, theta, d = hough_line(canny_img)
    lines = []
    print('\nAll hough lines')
    for _, angle, dist in zip(*hough_line_peaks(h, theta, d)):
        if np.sin(angle) == 0: # Evitar division por cero
            print(f"Skipping line with angle {np.degrees(angle):.2f} where sin(angle)=0")
            continue
        print(f"Angle:{np.degrees(angle):.2f}, Dist:{dist:.2f}")

        x1 = 0
        y1 = int((dist - x1 * np.cos(angle)) / np.sin(angle))
        x2 = canny_img.shape[1]
        y2 = (dist - x2 * np.cos(angle)) / np.sin(angle)
        lines.append({
            'dist': float(dist),
            'angle': np.degrees(float(angle)),
            'point1': [x1, y1],
            'point2': [x2, y2]
        })
    return lines

def shortlist_lines(lines):
    MIN_ANGLE = 10
    MAX_ANGLE = 70
    MIN_DIST = 5
    MAX_DIST = 256
    print(f"Total detected lines: {len(lines)}")
    shortlisted_lines = [
        x for x in lines if
        (x['dist'] >= MIN_DIST) and
        (x['dist'] <= MAX_DIST) and
        (x['angle'] >= MIN_ANGLE) and
        (x['angle'] <= MAX_ANGLE)
    ]
    print(f"Shortlisted lines: {len(shortlisted_lines)}")
    for i in shortlisted_lines:
        print(f"Angle:{i['angle']:.2f}, Dist:{i['dist']:.2f}")
    return shortlisted_lines

from skimage.draw import polygon
def remove_pectoral(shortlisted_lines):
    if not shortlisted_lines:
        print("No se encontraron lineas para eliminar el sculo pectoral.")
        return [], []
    shortlisted_lines.sort(key=lambda x: x['dist'])
    pectoral_line = shortlisted_lines[0]
    d = pectoral_line['dist']
    theta = np.radians(pectoral_line['angle'])
    x_intercept = d / np.cos(theta)
    y_intercept = d / np.sin(theta)
```

```

    return polygon([0, 0, y_intercept], [0, x_intercept, 0])
def draw_hough_lines(image, lines):
    """
    Dibuja l neas Hough sobre la imagen usando OpenCV.
    """
    # Crea una copia en formato BGR para superponer l neas de colores
    hough_image = cv2.cvtColor((image * 255).astype(np.uint8), cv2.COLOR_GRAY2BGR)

    for line in lines:
        x1, y1 = map(int, line['point1'])
        x2, y2 = map(int, line['point2'])
        cv2.line(hough_image, (x1, y1), (x2, y2), (0, 255, 0), 2) # Verde
    return hough_image

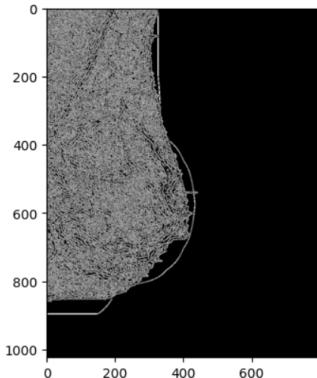
def display_image(filename):
    image = read_image(filename)
    canny_image = apply_canny(image)
    lines = get_hough_lines(canny_image)
    shortlisted_lines = shortlist_lines(lines)
    rr, cc = remove_pectoral(shortlisted_lines)
    image[rr, cc] = 0
    cv2.imwrite('image13.png', image)

canny_vector = np.zeros((120, 1024, 800))

for i in range(0, 120):
    canny_vector[i] = apply_canny(merg[i])

canny_image = apply_canny(merg[0])
plt.imshow(canny_image, cmap='gray')

```



```

lines_vector = []
short_vector = []

for i in range(120):
    image = canny_vector[i]
    if image.dtype != np.uint8:
        image = (image * 255).astype(np.uint8) if image.max() <= 1 else image.astype(np.uint8)

    lines = get_hough_lines(image) # Lista de diccionarios
    shortlisted = shortlist_lines(lines) # Lista filtrada

    lines_vector.append(lines) # Almacena la lista completa
    short_vector.append(shortlisted) # Almacena la lista filtrada

lines = get_hough_lines(canny_image)
shortlisted_lines = shortlist_lines(lines)

# Procesar cada copia y almacenar el resultado
for i in range(120):
    if not short_vector[i]: # Verificar si no hay l neas
        print(f"No se encontraron l neas para la imagen {i}.")
        remove_vector.append(copia_merg[i]) # Guardar la imagen sin modificar
        continue

```

```

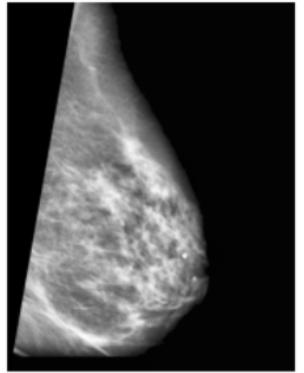
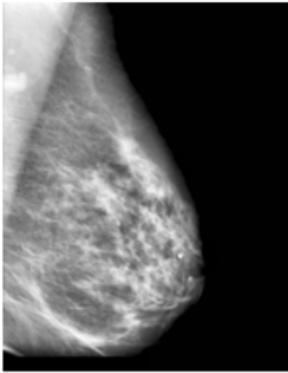
rr, cc = remove_pectoral(short_vector[i]) # Coordenadas del m sculo pectoral

# Validar que los ndices est n dentro de los l mites de la imagen
rr = np.clip(rr, 0, copia_merg[i].shape[0] - 1).astype(int)
cc = np.clip(cc, 0, copia_merg[i].shape[1] - 1).astype(int)

copia_merg[i][rr, cc] = 0 # Eliminar la regi n del m sculo
remove_vector.append(copia_merg[i]) # Guardar la imagen procesada

for i in range(0,120):
    plt.figure(figsize=(30, 420))
    plt.subplot(120, 1, 1)
    plt.title('Imagen original')
    plt.imshow(merg[i], cmap='gray')
    plt.axis('off')
    plt.subplot(120, 2, 2)
    plt.title(f'M sculo pectoral sacado {i}')
    plt.imshow(remove_vector[i], cmap='gray')
    plt.axis('off')
plt.show()

```



2.6. Extracci n de caracter sticas para T cnicas de Machine Learning

```

import os
import math
import h5py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from scipy.stats import skew
from scipy.stats import kurtosis
import scipy.ndimage as ndi
import sklearn.metrics as metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn import svm
from skimage.measure import shannon_entropy
import cv2
from skimage import io
from skimage import color
from skimage.draw import polygon
#from skimage.feature import greycomatrix, greycoprops
#from skimage.measure import greycomatrix, greycoprops
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import h5py
from skimage.feature import canny
from skimage.filters import sobel

```

```

from skimage.transform import hough_line, hough_line_peaks
from sklearn.metrics import classification_report, accuracy_score

with h5py.File('../input/final.h5', 'r') as scan_H5:
    print("Keys: %s" % scan_H5.keys())
    severity = scan_H5['severity'][:]
    scanH5 = scan_H5['img'][:]
    print('\n||||| File read successfully|||||')

```

Extracción de características: Pasar de categoricas a numerica

```

df = pd.DataFrame({'severity':severity})
df['severity'] = df['severity'].str.decode("utf-8")
df['severity'].replace('B',1,inplace=True)
df['severity'].replace('M',2,inplace=True)
df['severity'].replace('nan',3,inplace=True)

```

Sacar células nromales

```

li = []
for i in range(330):
    if df['severity'][i] != 3:
        li.append(i)

deletedScan = np.take(scanH5, li, axis=0)
df = df[df['severity'] != 3]
Y = df.values
Y = Y.flatten()

finalImage = deletedScan

```

Características estadísticas: featureVector = pd.DataFrame() Media

```

row=35
means = np.zeros((row,3))
for i in range(row):
    means[i] = cv2.mean(finalImage[i])[:3]

featureVector['means'] = means[:,0]

```

Desviación Estándar

```

stds = np.zeros((row,3))
for i in range(row):
    stds[i] = cv2.meanStdDev(finalImage[i])

featureVector['stds'] = stds[:,0]

```

Kurtosis

```

kurtos = np.zeros(row,)

for i in range(row):
    kurtos[i] = np.average(kurtosis(finalImage[i]))

featureVector['kurtosis'] = kurtos

```

Asimetría

```

skewness = np.zeros(row,)

for i in range(row):
    skewness[i] = np.average(skew(finalImage[i]))

featureVector['skewness'] = skewness

```

Suavidad

```

smoothness = np.zeros((row,))

for i in range(row):
    smoothness[i] = np.average(np.absolute(ndi.filters.laplace(finalImage[i]).astype(float) / 255.0))
featureVector['smooth'] = smoothness

```

Entropía

```

entropyFromImage = np.zeros(row,)
for i in range(row):
    entropyFromImage[i] = shannon_entropy(finalImage[i])
featureVector['entropy'] = entropyFromImage

```

Características GLCM

```

entropies = np.zeros((row,))
dissimilarity = np.zeros((row,))
correlation = np.zeros((row,))
homogeneity = np.zeros((row,))
energy = np.zeros((row,))
contrast= np.zeros((row,))
ASM= np.zeros((row,))

def getGlcM():
    for i in range(row):
        glcm = greycomatrix(finalImage[i].astype('uint8'), distances=[1],
                            angles=[0], symmetric=True,
                            normed=True)
    #        print(glcm.shape)

        dissimilarity[i]= greycoprops(glcm, 'dissimilarity')[0, 0]
        correlation[i]= greycoprops(glcm, 'correlation')[0, 0]
        homogeneity[i] = greycoprops(glcm, 'homogeneity')[0, 0]
        energy[i] = greycoprops(glcm, 'energy')[0, 0]
        contrast[i]= greycoprops(glcm, 'contrast')[0, 0]
        ASM[i] = greycoprops(glcm, 'ASM')[0,0]
        glcm1 = np.squeeze(glcm)
        entropies[i] = -np.sum(glcm1*np.log2(glcm1 + (glcm1==0)))

getGlcM()

featureVector['entropies'] = entropies
featureVector['dissimilarity'] = dissimilarity
featureVector['correlation'] = correlation
featureVector['homogeneity'] =homogeneity
featureVector['energy'] = energy
featureVector['contrast'] = contrast
featureVector['ASM'] =ASM

featureVector.head(5)

```

	means	stds	kurtosis	skewness	smooth	entropy	entropies	dissimilarity	correlation	homogeneity	energy	contrast
ASM												
0	34.640250	69.033719	-1.315060	0.421231	0.010550	2.368539	3.324533	1.030793	0.997564	0.834578	0.779962	23.242229
0.608341												
1	37.305714	72.430764	-1.757026	0.393993	0.010108	2.440434	3.380427	0.950110	0.997965	0.836059	0.772250	21.371413
0.596371												
2	73.054004	78.400490	-0.287723	-0.546568	0.025749	4.106596	6.224676	2.319036	0.996399	0.632705	0.523306	44.290387
0.273849												
3	73.054004	78.400490	-0.287723	-0.546568	0.025749	4.106596	6.224676	2.319036	0.996399	0.632705	0.523306	44.290387
0.273849												
4	47.341917	75.290774	-1.717753	0.011414	0.015706	2.885099	4.210470	1.525751	0.996866	0.770632	0.706395	35.551051
0.498994												

Cuadro 1: Estadísticas y características de las imágenes.

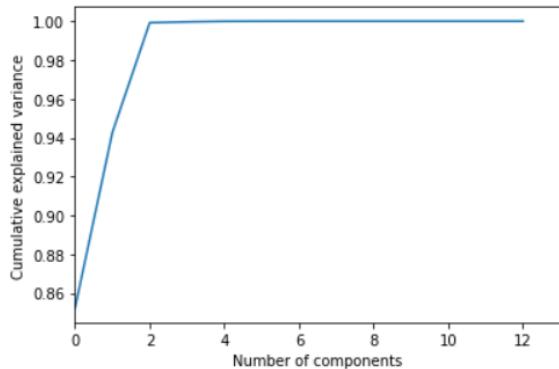
Feature selection

```

pca = PCA().fit(featureVector)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,13,1)

```

```
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```



```
sklearn_pca = PCA(n_components=3)
X = sklearn_pca.fit_transform(featureVector)

# test
X = featureVector
```

Entrenamiento y testeo

```
X_train, X_test = train_test_split(X,
                                    test_size = 0.25,
                                    random_state = 2021)

Y_train, Y_test = train_test_split(Y, test_size = 0.25, random_state = 2021)

print('train', Y_train.shape[0], 'Test', Y_test.shape[0])
```

train 92 Test 31

3. Resultados

3.1. k-Nearest Neighbors (k-NN)

Creación del modelo

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

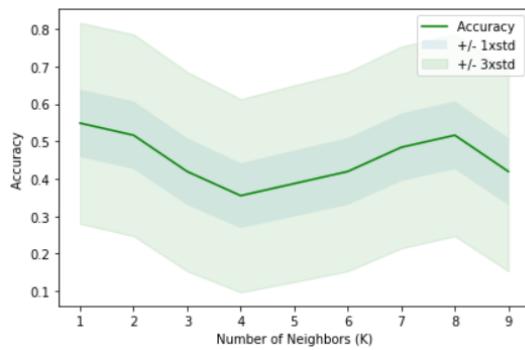
    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,Y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(Y_test, yhat)

    std_acc[n-1]=np.std(yhat==Y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

array([0.5483871, 0.51612903, 0.41935484, 0.35483871, 0.38709677, 0.41935484, 0.48387097, 0.51612903, 0.41935484])

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,color="green")
plt.legend(('Accuracy','+/-_1xstd','+/-_3xstd'))
plt.ylabel('Accuracy')
plt.xlabel('Number of Neighbors(K)')
plt.tight_layout()
plt.show()
```



```
print("El mejor accuracy es con", mean_acc.max(), "con k=", mean_acc.argmax()+1)
```

El mejor accuracy es con 0.5483870967741935 con k= 1

```
# should find K
knn = KNeighborsClassifier(1)
knn.fit(X_train, Y_train)
Y_pred_knn = knn.predict(X_test)
print('Accuracy %2.2f%' % (100*accuracy_score(Y_test, Y_pred_knn)))
print(classification_report(Y_test, Y_pred_knn))
```

Clase	Precision	Recall	F1-Score	Support
1	0.58	0.65	0.61	17
2	0.50	0.43	0.46	14
Accuracy			0.55	31
Macro avg	0.54	0.54	0.54	31
Weighted avg	0.54	0.55	0.54	31

Cuadro 2: Métricas de rendimiento del modelo

3.2. Random Forest

```
rfc = RandomForestClassifier()
rfc.fit(X_train, Y_train)
Y_pred_rff = rfc.predict(X_test)
print('Accuracy %2.2f%' % (100*accuracy_score(Y_test, Y_pred_rff)))
print(classification_report(Y_test, Y_pred_rff))
```

Clase	Precision	Recall	F1-Score	Support
1	0.57	0.71	0.63	17
2	0.50	0.36	0.42	14
Accuracy			0.55	31
Macro avg	0.54	0.53	0.52	31
Weighted avg	0.54	0.55	0.53	31

Cuadro 3: Métricas de rendimiento del modelo

4. Conclusión

5. Anexo

5.1. Sección 1

```
import os
import math
```

```

from skimage.segmentation import flood as flood
import h5py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from scipy.stats import skew
from scipy.stats import kurtosis
import scipy.ndimage as ndi
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn import svm
from skimage.measure import shannon_entropy
import cv2
from skimage import io
from skimage import color
from skimage.draw import polygon
#from skimage.feature import greycomatrix, greycoprops
from sklearn.decomposition import PCA

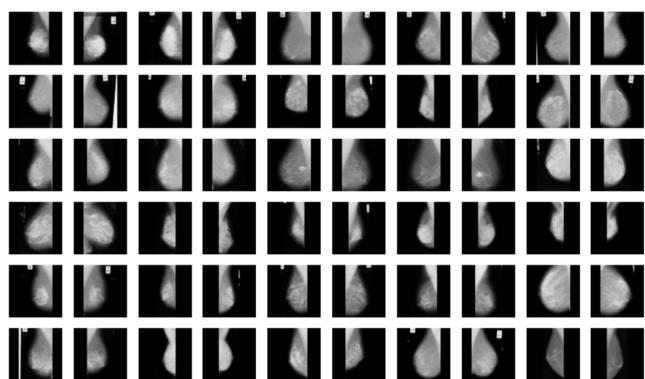
from skimage.feature import canny
from skimage.filters import sobel
from skimage.transform import hough_line, hough_line_peaks

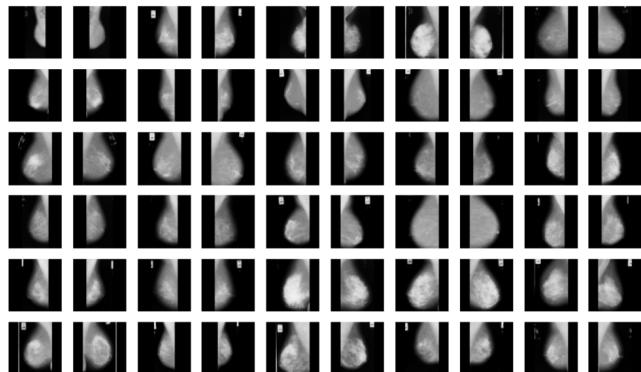
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

from IPython.display import Image
#Lectura de imágenes
carpetas=[]
for i in range(0,120):
    if i<9:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TP_final_imagenes/mdb0{i+1}.pgm')
        carpetas.append(carpeta)
    elif i<99:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TP_final_imagenes/mdb0{i+1}.pgm')
        carpetas.append(carpeta)
    else:
        carpeta = cv2.imread(f'/content/drive/MyDrive/PAByB/TP_final_imagenes/mdb{i+1}.pgm')
        carpetas.append(carpeta)

plt.imshow(carpetas[1], cmap="gray")

```





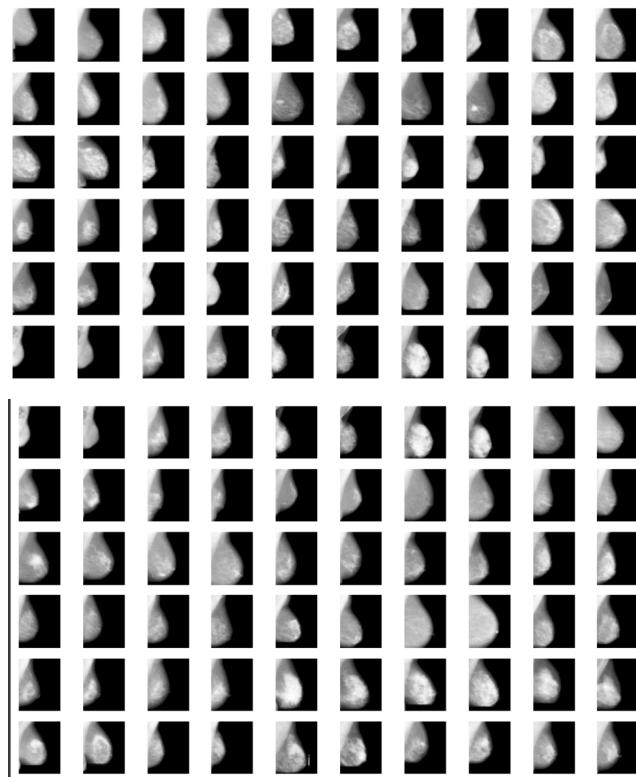
5.2. Sección 2

Los resultados de las 120 imágenes volteadas después del algortimos de volteo automático y manual:



5.3. Sección 3

Los resultados de las 120 imágenes volteadas después del algortimos de remoción de artefactos y recorte:



5.4. Sección 4

Las estadísticas de las 120 imágenes:

1. Imagen: 1, varianza: 0.0, media: 0.0
2. Imagen: 2, varianza: 0.0, media: 0.0
3. Imagen: 3, varianza: 0.0, media: 0.0
4. Imagen: 4, varianza: 0.0, media: 0.0
5. Imagen: 5, varianza: 0.0, media: 0.0
6. Imagen: 6, varianza: 0.0, media: 0.0
7. Imagen: 7, varianza: 0.0, media: 0.0
8. Imagen: 8, varianza: 0.0, media: 0.0
9. Imagen: 9, varianza: 0.0, media: 0.0
10. Imagen: 10, varianza: 0.0, media: 0.0
11. Imagen: 11, varianza: 0.0, media: 0.0
12. Imagen: 12, varianza: 0.0, media: 0.0
13. Imagen: 13, varianza: 0.0, media: 0.0
14. Imagen: 14, varianza: 0.0, media: 0.0
15. Imagen: 15, varianza: 0.0, media: 0.0
16. Imagen: 16, varianza: 0.0, media: 0.0
17. Imagen: 17, varianza: 0.0, media: 0.0
18. Imagen: 18, varianza: 0.0, media: 0.0

19. Imagen: 19, varianza: 0.0, media: 0.0
20. Imagen: 20, varianza: 0.0, media: 0.0
21. Imagen: 21, varianza: 0.0, media: 0.0
22. Imagen: 22, varianza: 0.0, media: 0.0
23. Imagen: 23, varianza: 0.0, media: 0.0
24. Imagen: 24, varianza: 0.0, media: 0.0
25. Imagen: 25, varianza: 0.0, media: 0.0
26. Imagen: 26, varianza: 0.0, media: 0.0
27. Imagen: 27, varianza: 0.0, media: 0.0
28. Imagen: 28, varianza: 0.0, media: 0.0
29. Imagen: 29, varianza: 0.0, media: 0.0
30. Imagen: 30, varianza: 0.0, media: 0.0
31. Imagen: 31, varianza: 0.0, media: 0.0
32. Imagen: 32, varianza: 0.0, media: 0.0
33. Imagen: 33, varianza: 0.0, media: 0.0
34. Imagen: 34, varianza: 0.0, media: 0.0
35. Imagen: 35, varianza: 0.0, media: 0.0
36. Imagen: 36, varianza: 0.0, media: 0.0
37. Imagen: 37, varianza: 0.0, media: 0.0
38. Imagen: 38, varianza: 0.0, media: 0.0
39. Imagen: 39, varianza: 0.0, media: 0.0
40. Imagen: 40, varianza: 0.0, media: 0.0
41. Imagen: 41, varianza: 0.0, media: 0.0
42. Imagen: 42, varianza: 0.0, media: 0.0
43. Imagen: 43, varianza: 0.0, media: 0.0
44. Imagen: 44, varianza: 0.0, media: 0.0
45. Imagen: 45, varianza: 0.0, media: 0.0
46. Imagen: 46, varianza: 0.0, media: 0.0
47. Imagen: 47, varianza: 0.0, media: 0.0
48. Imagen: 48, varianza: 0.0, media: 0.0
49. Imagen: 49, varianza: 0.0, media: 0.0
50. Imagen: 50, varianza: 0.0, media: 0.0
51. Imagen: 51, varianza: 0.0, media: 0.0
52. Imagen: 52, varianza: 0.0, media: 0.0
53. Imagen: 53, varianza: 0.0, media: 0.0
54. Imagen: 54, varianza: 0.0, media: 0.0
55. Imagen: 55, varianza: 0.0, media: 0.0

56. Imagen: 56, varianza: 0.0, media: 0.0
57. Imagen: 57, varianza: 0.0, media: 0.0
58. Imagen: 58, varianza: 0.0, media: 0.0
59. Imagen: 59, varianza: 0.0, media: 0.0
60. Imagen: 60, varianza: 0.0, media: 0.0
61. Imagen: 61, varianza: 0.0, media: 0.0
62. Imagen: 62, varianza: 0.0, media: 0.0
63. Imagen: 63, varianza: 0.0, media: 0.0
64. Imagen: 64, varianza: 0.0, media: 0.0
65. Imagen: 65, varianza: 0.0, media: 0.0
66. Imagen: 66, varianza: 0.0, media: 0.0
67. Imagen: 67, varianza: 0.0, media: 0.0
68. Imagen: 68, varianza: 0.0, media: 0.0
69. Imagen: 69, varianza: 0.0, media: 0.0
70. Imagen: 70, varianza: 0.0, media: 0.0
71. Imagen: 71, varianza: 0.0, media: 0.0
72. Imagen: 72, varianza: 0.0, media: 0.0
73. Imagen: 73, varianza: 0.0, media: 0.0
74. Imagen: 74, varianza: 0.0, media: 0.0
75. Imagen: 75, varianza: 0.0, media: 0.0
76. Imagen: 76, varianza: 0.0, media: 0.0
77. Imagen: 77, varianza: 0.0, media: 0.0
78. Imagen: 78, varianza: 0.0, media: 0.0
79. Imagen: 79, varianza: 0.0, media: 0.0
80. Imagen: 80, varianza: 0.0, media: 0.0
81. Imagen: 81, varianza: 0.0, media: 0.0
82. Imagen: 82, varianza: 0.0, media: 0.0
83. Imagen: 83, varianza: 0.0, media: 0.0
84. Imagen: 84, varianza: 0.0, media: 0.0
85. Imagen: 85, varianza: 0.0, media: 0.0
86. Imagen: 86, varianza: 0.0, media: 0.0
87. Imagen: 87, varianza: 0.0, media: 0.0
88. Imagen: 88, varianza: 0.0, media: 0.0
89. Imagen: 89, varianza: 0.0, media: 0.0
90. Imagen: 90, varianza: 0.0, media: 0.0
91. Imagen: 91, varianza: 0.0, media: 0.0
92. Imagen: 92, varianza: 0.0, media: 0.0

93. Imagen: 93, varianza: 0.0, media: 0.0
94. Imagen: 94, varianza: 0.0, media: 0.0
95. Imagen: 95, varianza: 0.0, media: 0.0
96. Imagen: 96, varianza: 0.0, media: 0.0
97. Imagen: 97, varianza: 0.0, media: 0.0
98. Imagen: 98, varianza: 0.0, media: 0.0
99. Imagen: 99, varianza: 0.0, media: 0.0
100. Imagen: 100, varianza: 0.0, media: 0.0
101. Imagen: 101, varianza: 0.0, media: 0.0
102. Imagen: 102, varianza: 0.0, media: 0.0
103. Imagen: 103, varianza: 0.0, media: 0.0
104. Imagen: 104, varianza: 0.0, media: 0.0
105. Imagen: 105, varianza: 0.0, media: 0.0
106. Imagen: 106, varianza: 0.0, media: 0.0
107. Imagen: 107, varianza: 0.0, media: 0.0
108. Imagen: 108, varianza: 0.0, media: 0.0
109. Imagen: 109, varianza: 0.0, media: 0.0
110. Imagen: 110, varianza: 0.0, media: 0.0
111. Imagen: 111, varianza: 0.0, media: 0.0
112. Imagen: 112, varianza: 0.0, media: 0.0
113. Imagen: 113, varianza: 0.0, media: 0.0
114. Imagen: 114, varianza: 0.0, media: 0.0
115. Imagen: 115, varianza: 0.0, media: 0.0
116. Imagen: 116, varianza: 0.0, media: 0.0
117. Imagen: 117, varianza: 0.0, media: 0.0
118. Imagen: 118, varianza: 0.0, media: 0.0
119. Imagen: 119, varianza: 0.0, media: 0.0
120. Imagen: 120, varianza: 0.0, media: 0.0

article enumerate

1. Imagen: 1, varianza: 0.0, media: 0.0
2. Imagen: 2, varianza: 0.0, media: 0.0
3. Imagen: 3, varianza: 0.0, media: 0.0
4. Imagen: 4, varianza: 0.0, media: 0.0
5. Imagen: 5, varianza: 0.0, media: 0.0
6. Imagen: 6, varianza: 0.0, media: 0.0
7. Imagen: 7, varianza: 0.0, media: 0.0
8. Imagen: 8, varianza: 0.0, media: 0.0

9. Imagen: 9, varianza: 0.0, media: 0.0
10. Imagen: 10, varianza: 0.0, media: 0.0
11. Imagen: 11, varianza: 0.0, media: 0.0
12. Imagen: 12, varianza: 0.0, media: 0.0
13. Imagen: 13, varianza: 0.0, media: 0.0
14. Imagen: 14, varianza: 0.0, media: 0.0
15. Imagen: 15, varianza: 0.0, media: 0.0
16. Imagen: 16, varianza: 0.0, media: 0.0
17. Imagen: 17, varianza: 0.0, media: 0.0
18. Imagen: 18, varianza: 0.0, media: 0.0
19. Imagen: 19, varianza: 0.0, media: 0.0
20. Imagen: 20, varianza: 0.0, media: 0.0
21. Imagen: 21, varianza: 0.0, media: 0.0
22. Imagen: 22, varianza: 0.0, media: 0.0
23. Imagen: 23, varianza: 0.0, media: 0.0
24. Imagen: 24, varianza: 0.0, media: 0.0
25. Imagen: 25, varianza: 0.0, media: 0.0
26. Imagen: 26, varianza: 0.0, media: 0.0
27. Imagen: 27, varianza: 0.0, media: 0.0
28. Imagen: 28, varianza: 0.0, media: 0.0
29. Imagen: 29, varianza: 0.0, media: 0.0
30. Imagen: 30, varianza: 0.0, media: 0.0
31. Imagen: 31, varianza: 0.0, media: 0.0
32. Imagen: 32, varianza: 0.0, media: 0.0
33. Imagen: 33, varianza: 0.0, media: 0.0
34. Imagen: 34, varianza: 0.0, media: 0.0
35. Imagen: 35, varianza: 0.0, media: 0.0
36. Imagen: 36, varianza: 0.0, media: 0.0
37. Imagen: 37, varianza: 0.0, media: 0.0
38. Imagen: 38, varianza: 0.0, media: 0.0
39. Imagen: 39, varianza: 0.0, media: 0.0
40. Imagen: 40, varianza: 0.0, media: 0.0
41. Imagen: 41, varianza: 0.0, media: 0.0
42. Imagen: 42, varianza: 0.0, media: 0.0
43. Imagen: 43, varianza: 0.0, media: 0.0
44. Imagen: 44, varianza: 0.0, media: 0.0
45. Imagen: 45, varianza: 0.0, media: 0.0

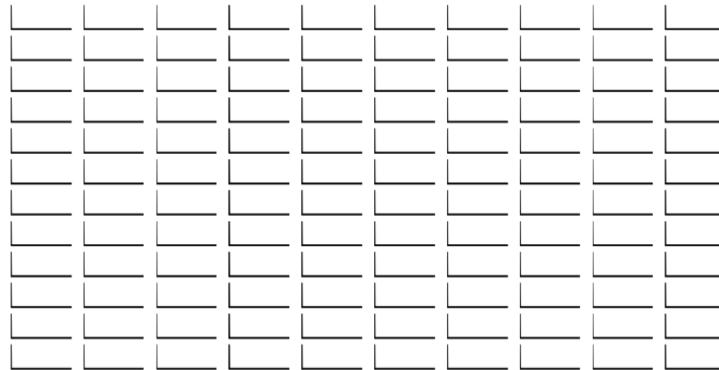
46. Imagen: 46, varianza: 0.0, media: 0.0
47. Imagen: 47, varianza: 0.0, media: 0.0
48. Imagen: 48, varianza: 0.0, media: 0.0
49. Imagen: 49, varianza: 0.0, media: 0.0
50. Imagen: 50, varianza: 0.0, media: 0.0
51. Imagen: 51, varianza: 0.0, media: 0.0
52. Imagen: 52, varianza: 0.0, media: 0.0
53. Imagen: 53, varianza: 0.0, media: 0.0
54. Imagen: 54, varianza: 0.0, media: 0.0
55. Imagen: 55, varianza: 0.0, media: 0.0
56. Imagen: 56, varianza: 0.0, media: 0.0
57. Imagen: 57, varianza: 0.0, media: 0.0
58. Imagen: 58, varianza: 0.0, media: 0.0
59. Imagen: 59, varianza: 0.0, media: 0.0
60. Imagen: 60, varianza: 0.0, media: 0.0
61. Imagen: 61, varianza: 0.0, media: 0.0
62. Imagen: 62, varianza: 0.0, media: 0.0
63. Imagen: 63, varianza: 0.0, media: 0.0
64. Imagen: 64, varianza: 0.0, media: 0.0
65. Imagen: 65, varianza: 0.0, media: 0.0
66. Imagen: 66, varianza: 0.0, media: 0.0
67. Imagen: 67, varianza: 0.0, media: 0.0
68. Imagen: 68, varianza: 0.0, media: 0.0
69. Imagen: 69, varianza: 0.0, media: 0.0
70. Imagen: 70, varianza: 0.0, media: 0.0
71. Imagen: 71, varianza: 0.0, media: 0.0
72. Imagen: 72, varianza: 0.0, media: 0.0
73. Imagen: 73, varianza: 0.0, media: 0.0
74. Imagen: 74, varianza: 0.0, media: 0.0
75. Imagen: 75, varianza: 0.0, media: 0.0
76. Imagen: 76, varianza: 0.0, media: 0.0
77. Imagen: 77, varianza: 0.0, media: 0.0
78. Imagen: 78, varianza: 0.0, media: 0.0
79. Imagen: 79, varianza: 0.0, media: 0.0
80. Imagen: 80, varianza: 0.0, media: 0.0
81. Imagen: 81, varianza: 0.0, media: 0.0
82. Imagen: 82, varianza: 0.0, media: 0.0

83. Imagen: 83, varianza: 0.0, media: 0.0
84. Imagen: 84, varianza: 0.0, media: 0.0
85. Imagen: 85, varianza: 0.0, media: 0.0
86. Imagen: 86, varianza: 0.0, media: 0.0
87. Imagen: 87, varianza: 0.0, media: 0.0
88. Imagen: 88, varianza: 0.0, media: 0.0
89. Imagen: 89, varianza: 0.0, media: 0.0
90. Imagen: 90, varianza: 0.0, media: 0.0
91. Imagen: 91, varianza: 0.0, media: 0.0
92. Imagen: 92, varianza: 0.0, media: 0.0
93. Imagen: 93, varianza: 0.0, media: 0.0
94. Imagen: 94, varianza: 0.0, media: 0.0
95. Imagen: 95, varianza: 0.0, media: 0.0
96. Imagen: 96, varianza: 0.0, media: 0.0
97. Imagen: 97, varianza: 0.0, media: 0.0
98. Imagen: 98, varianza: 0.0, media: 0.0
99. Imagen: 99, varianza: 0.0, media: 0.0
100. Imagen: 100, varianza: 0.0, media: 0.0
101. Imagen: 101, varianza: 0.0, media: 0.0
102. Imagen: 102, varianza: 0.0, media: 0.0
103. Imagen: 103, varianza: 0.0, media: 0.0
104. Imagen: 104, varianza: 0.0, media: 0.0
105. Imagen: 105, varianza: 0.0, media: 0.0
106. Imagen: 106, varianza: 0.0, media: 0.0
107. Imagen: 107, varianza: 0.0, media: 0.0
108. Imagen: 108, varianza: 0.0, media: 0.0
109. Imagen: 109, varianza: 0.0, media: 0.0
110. Imagen: 110, varianza: 0.0, media: 0.0
111. Imagen: 111, varianza: 0.0, media: 0.0
112. Imagen: 112, varianza: 0.0, media: 0.0
113. Imagen: 113, varianza: 0.0, media: 0.0
114. Imagen: 114, varianza: 0.0, media: 0.0
115. Imagen: 115, varianza: 0.0, media: 0.0
116. Imagen: 116, varianza: 0.0, media: 0.0
117. Imagen: 117, varianza: 0.0, media: 0.0
118. Imagen: 118, varianza: 0.0, media: 0.0
119. Imagen: 119, varianza: 0.0, media: 0.0

120. Imagen: 120, varianza: 0.0, media: 0.0

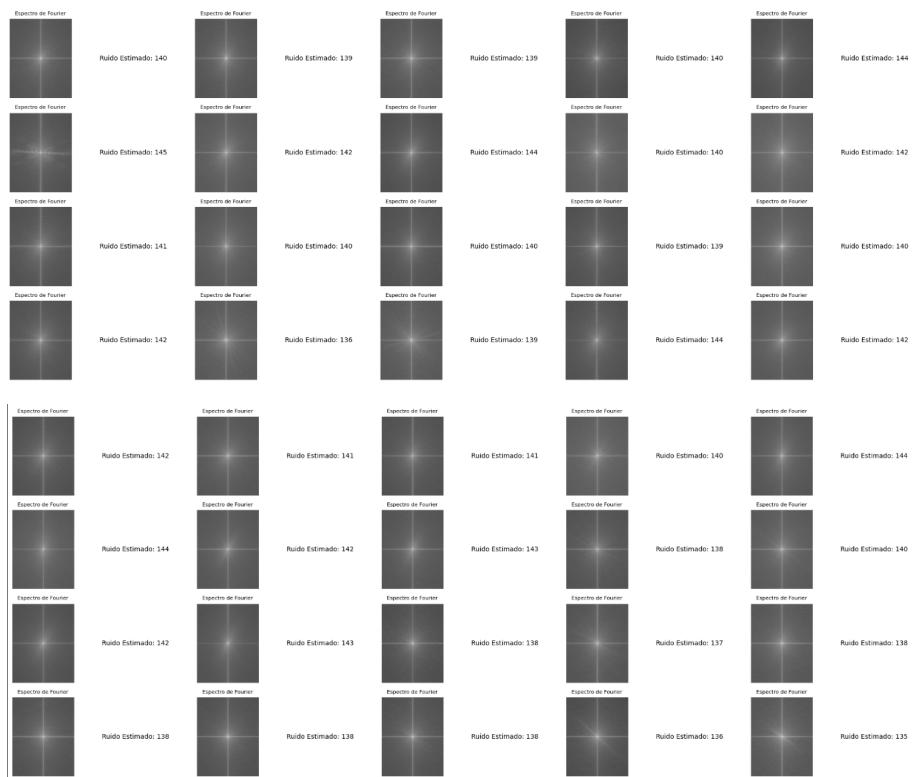
5.5. Sección 5

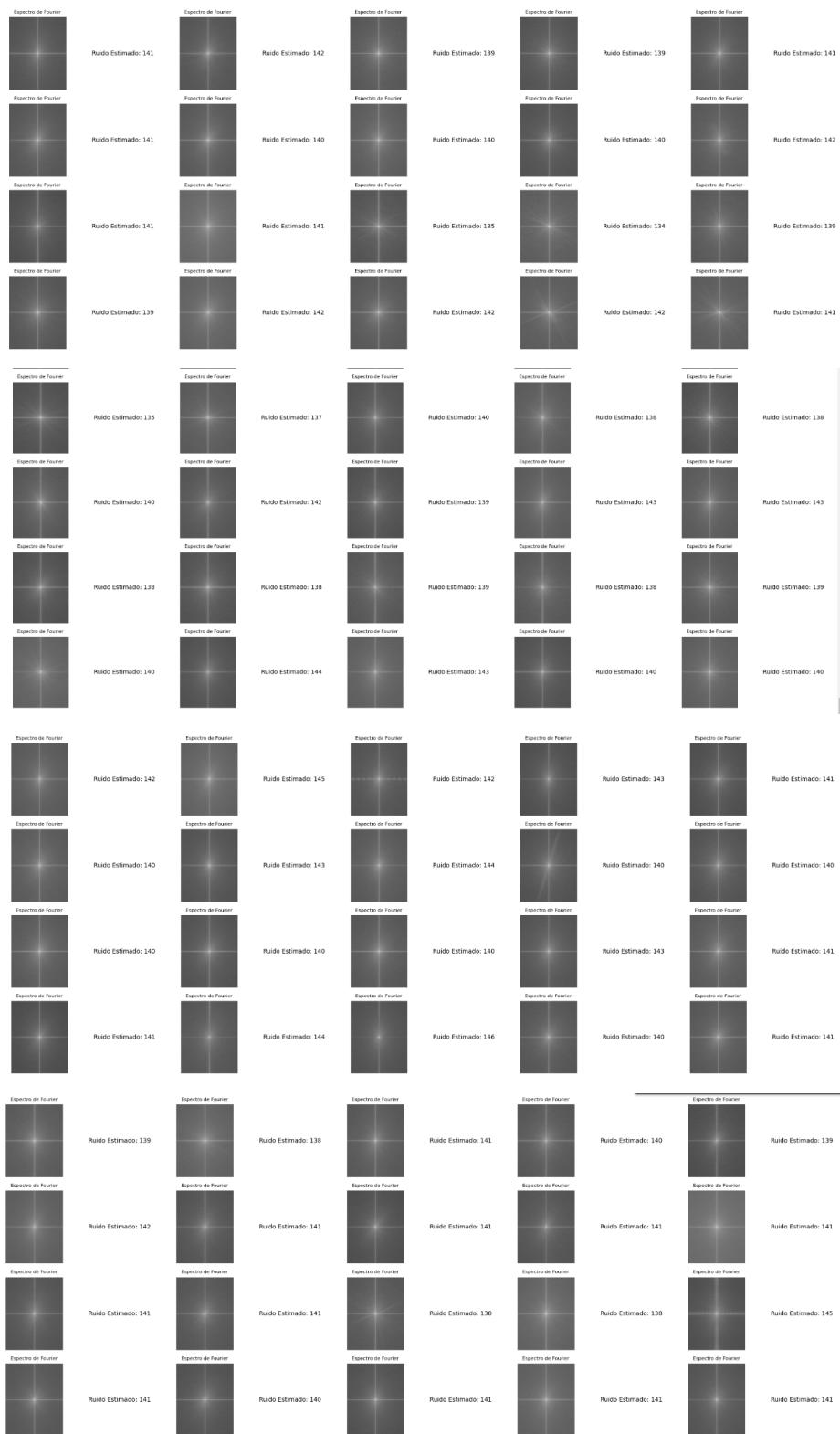
Los histogramas de las 120 imágenes:



5.6. Sección 6

Estimar el ruido de las 120 imágenes con la transformada de Fourier:





5.7. Sección 7

Estimación del ruido para las 120 imágenes con la transformada wavelet:

1. Imagen número: 1 - Estimación del nivel de ruido: 1.8689807653427124
2. Imagen número: 2 - Estimación del nivel de ruido: 1.802535057067871

3. Imagen número: 3 - Estimación del nivel de ruido: 1.8874021768569946
4. Imagen número: 4 - Estimación del nivel de ruido: 2.0666253566741943
5. Imagen número: 5 - Estimación del nivel de ruido: 2.4214725494384766
6. Imagen número: 6 - Estimación del nivel de ruido: 3.074233055114746
7. Imagen número: 7 - Estimación del nivel de ruido: 2.2409183979034424
8. Imagen número: 8 - Estimación del nivel de ruido: 2.6394755840301514
9. Imagen número: 9 - Estimación del nivel de ruido: 2.126112937927246
10. Imagen número: 10 - Estimación del nivel de ruido: 2.2248759269714355
11. Imagen número: 11 - Estimación del nivel de ruido: 2.234548330307007
12. Imagen número: 12 - Estimación del nivel de ruido: 2.115666627883911
13. Imagen número: 13 - Estimación del nivel de ruido: 2.0661253929138184
14. Imagen número: 14 - Estimación del nivel de ruido: 1.8446522951126099
15. Imagen número: 15 - Estimación del nivel de ruido: 2.5071961879730225
16. Imagen número: 16 - Estimación del nivel de ruido: 2.13735294342041
17. Imagen número: 17 - Estimación del nivel de ruido: 1.9961323738098145
18. Imagen número: 18 - Estimación del nivel de ruido: 1.7712582349777222
19. Imagen número: 19 - Estimación del nivel de ruido: 4.070398330688477
20. Imagen número: 20 - Estimación del nivel de ruido: 2.12402606010437
21. Imagen número: 21 - Estimación del nivel de ruido: 2.667194128036499
22. Imagen número: 22 - Estimación del nivel de ruido: 2.0977752208709717
23. Imagen número: 23 - Estimación del nivel de ruido: 2.7396633625030518
24. Imagen número: 24 - Estimación del nivel de ruido: 1.9767929315567017
25. Imagen número: 25 - Estimación del nivel de ruido: 2.3265624046325684
26. Imagen número: 26 - Estimación del nivel de ruido: 2.506366014480591
27. Imagen número: 27 - Estimación del nivel de ruido: 2.4394209384918213
28. Imagen número: 28 - Estimación del nivel de ruido: 2.08256459236145
29. Imagen número: 29 - Estimación del nivel de ruido: 2.207547664642334
30. Imagen número: 30 - Estimación del nivel de ruido: 1.990684151649475
31. Imagen número: 31 - Estimación del nivel de ruido: 2.1412241458892822
32. Imagen número: 32 - Estimación del nivel de ruido: 2.6578333377838135
33. Imagen número: 33 - Estimación del nivel de ruido: 2.0014541149139404
34. Imagen número: 34 - Estimación del nivel de ruido: 1.9445034265518188
35. Imagen número: 35 - Estimación del nivel de ruido: 1.8432408571243286
36. Imagen número: 36 - Estimación del nivel de ruido: 1.825203537940979
37. Imagen número: 37 - Estimación del nivel de ruido: 2.127866268157959
38. Imagen número: 38 - Estimación del nivel de ruido: 1.8153581619262695
39. Imagen número: 39 - Estimación del nivel de ruido: 1.7432446479797363

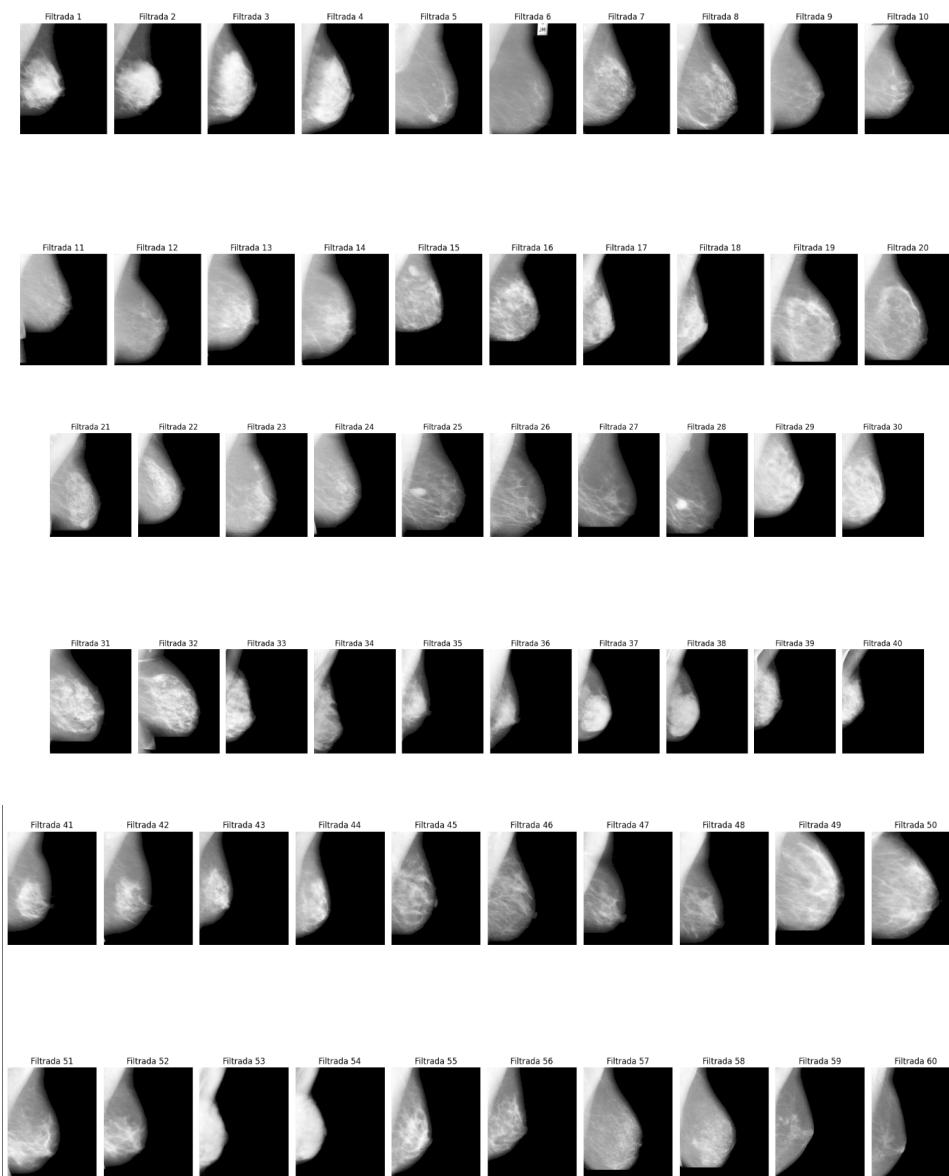
40. Imagen número: 40 - Estimación del nivel de ruido: 1.7187353372573853
41. Imagen número: 41 - Estimación del nivel de ruido: 2.2822506427764893
42. Imagen número: 42 - Estimación del nivel de ruido: 1.9695792198181152
43. Imagen número: 43 - Estimación del nivel de ruido: 1.7710380554199219
44. Imagen número: 44 - Estimación del nivel de ruido: 1.9147462844848633
45. Imagen número: 45 - Estimación del nivel de ruido: 2.0536909103393555
46. Imagen número: 46 - Estimación del nivel de ruido: 2.014547109603882
47. Imagen número: 47 - Estimación del nivel de ruido: 2.3213260173797607
48. Imagen número: 48 - Estimación del nivel de ruido: 1.988856315612793
49. Imagen número: 49 - Estimación del nivel de ruido: 2.4313416481018066
50. Imagen número: 50 - Estimación del nivel de ruido: 2.1680057048797607
51. Imagen número: 51 - Estimación del nivel de ruido: 2.301175117492676
52. Imagen número: 52 - Estimación del nivel de ruido: 2.024676561355591
53. Imagen número: 53 - Estimación del nivel de ruido: 2.0260775089263916
54. Imagen número: 54 - Estimación del nivel de ruido: 1.740662932395935
55. Imagen número: 55 - Estimación del nivel de ruido: 1.9878877401351929
56. Imagen número: 56 - Estimación del nivel de ruido: 1.8398040533065796
57. Imagen número: 57 - Estimación del nivel de ruido: 2.1044318675994873
58. Imagen número: 58 - Estimación del nivel de ruido: 2.110410690307617
59. Imagen número: 59 - Estimación del nivel de ruido: 2.0633766651153564
60. Imagen número: 60 - Estimación del nivel de ruido: 2.0635201930999756
61. Imagen número: 61 - Estimación del nivel de ruido: 1.6553648710250854
62. Imagen número: 62 - Estimación del nivel de ruido: 1.6961761713027954
63. Imagen número: 63 - Estimación del nivel de ruido: 2.167093515396118
64. Imagen número: 64 - Estimación del nivel de ruido: 1.9893079996109009
65. Imagen número: 65 - Estimación del nivel de ruido: 1.971517562866211
66. Imagen número: 66 - Estimación del nivel de ruido: 1.8222814798355103
67. Imagen número: 67 - Estimación del nivel de ruido: 2.9530441761016846
68. Imagen número: 68 - Estimación del nivel de ruido: 1.9184436798095703
69. Imagen número: 69 - Estimación del nivel de ruido: 2.5601069927215576
70. Imagen número: 70 - Estimación del nivel de ruido: 2.308398962020874
71. Imagen número: 71 - Estimación del nivel de ruido: 2.1295979022979736
72. Imagen número: 72 - Estimación del nivel de ruido: 1.8300957679748535
73. Imagen número: 73 - Estimación del nivel de ruido: 2.3841640949249268
74. Imagen número: 74 - Estimación del nivel de ruido: 2.045501947402954
75. Imagen número: 75 - Estimación del nivel de ruido: 1.848164677619934
76. Imagen número: 76 - Estimación del nivel de ruido: 2.033203363418579

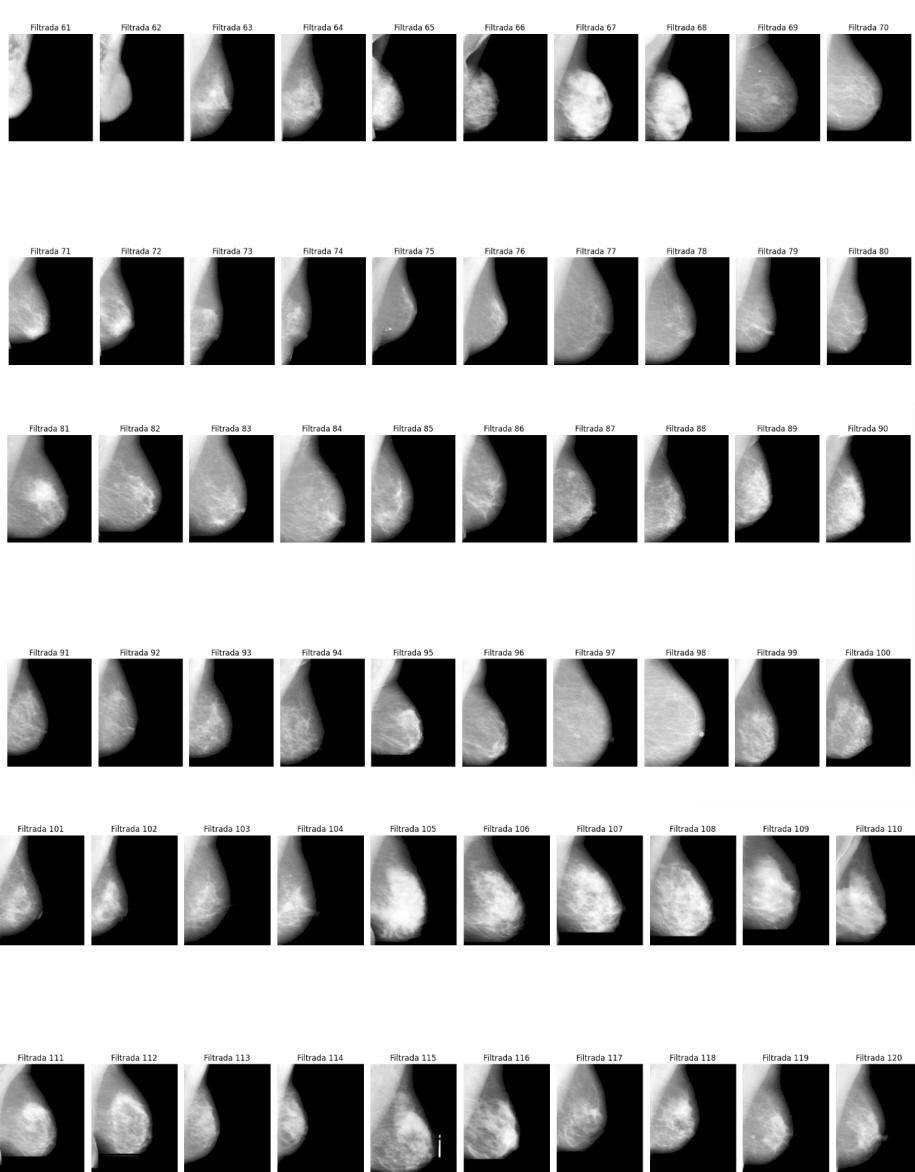
77. Imagen número: 77 - Estimación del nivel de ruido: 2.3889577388763428
78. Imagen número: 78 - Estimación del nivel de ruido: 2.1368560791015625
79. Imagen número: 79 - Estimación del nivel de ruido: 1.9782567024230957
80. Imagen número: 80 - Estimación del nivel de ruido: 2.006558418273926
81. Imagen número: 81 - Estimación del nivel de ruido: 2.3163273334503174
82. Imagen número: 82 - Estimación del nivel de ruido: 2.3216593265533447
83. Imagen número: 83 - Estimación del nivel de ruido: 2.3052332401275635
84. Imagen número: 84 - Estimación del nivel de ruido: 3.0524799823760986
85. Imagen número: 85 - Estimación del nivel de ruido: 2.354590654373169
86. Imagen número: 86 - Estimación del nivel de ruido: 1.8919099569320679
87. Imagen número: 87 - Estimación del nivel de ruido: 2.220329523086548
88. Imagen número: 88 - Estimación del nivel de ruido: 2.365697145462036
89. Imagen número: 89 - Estimación del nivel de ruido: 1.98248291015625
90. Imagen número: 90 - Estimación del nivel de ruido: 2.026078224182129
91. Imagen número: 91 - Estimación del nivel de ruido: 1.9340434074401855
92. Imagen número: 92 - Estimación del nivel de ruido: 1.9067901372909546
93. Imagen número: 93 - Estimación del nivel de ruido: 2.0456106662750244
94. Imagen número: 94 - Estimación del nivel de ruido: 2.1941678524017334
95. Imagen número: 95 - Estimación del nivel de ruido: 2.466956853866577
96. Imagen número: 96 - Estimación del nivel de ruido: 2.0937047004699707
97. Imagen número: 97 - Estimación del nivel de ruido: 3.742621660232544
98. Imagen número: 98 - Estimación del nivel de ruido: 2.978510856628418
99. Imagen número: 99 - Estimación del nivel de ruido: 2.5331292152404785
100. Imagen número: 100 - Estimación del nivel de ruido: 2.115816116333008
101. Imagen número: 101 - Estimación del nivel de ruido: 1.9253875017166138
102. Imagen número: 102 - Estimación del nivel de ruido: 1.8380743265151978
103. Imagen número: 103 - Estimación del nivel de ruido: 2.264296770095825
104. Imagen número: 104 - Estimación del nivel de ruido: 2.027695417404175
105. Imagen número: 105 - Estimación del nivel de ruido: 2.309577465057373
106. Imagen número: 106 - Estimación del nivel de ruido: 2.0446033477783203
107. Imagen número: 107 - Estimación del nivel de ruido: 3.1517837047576904
108. Imagen número: 108 - Estimación del nivel de ruido: 1.98811674118042
109. Imagen número: 109 - Estimación del nivel de ruido: 2.381478786468506
110. Imagen número: 110 - Estimación del nivel de ruido: 2.0648281574249268
111. Imagen número: 111 - Estimación del nivel de ruido: 2.113908290863037
112. Imagen número: 112 - Estimación del nivel de ruido: 2.5404443740844727
113. Imagen número: 113 - Estimación del nivel de ruido: 2.0609776973724365

114. Imagen número: 114 - Estimación del nivel de ruido: 1.839152216911316
115. Imagen número: 115 - Estimación del nivel de ruido: 3.500232696533203
116. Imagen número: 116 - Estimación del nivel de ruido: 2.4591991901397705
117. Imagen número: 117 - Estimación del nivel de ruido: 2.058652639389038
118. Imagen número: 118 - Estimación del nivel de ruido: 2.029325485229492
119. Imagen número: 119 - Estimación del nivel de ruido: 2.1040849685668945
120. Imagen número: 120 - Estimación del nivel de ruido: 2.068614959716797

5.8. Sección 8

Los resultados de los filtros promediadores de las 120 imágenes:





5.9. Sección 9

Los resultados de las 120 imágenes con CLAHE:

