import os

os.chdir("/content/drive/MyDrive/Ceren/MSc/Digital Image Processing/src")

# Import necessary packages

# bu çalışmada ihtiyaç duyulacak olan kütüphaneler yüklendi.

# bu kütüphanelerin her biri kendisi için spesifik bir amaç doğrultusunda kullanılmıştır.

# pandas kütüphanesi, veriseti içerisinde kullanılmış olan görseller hakkında detaylı bilgiler içeren CSV dosyasını okumak ve etkili kullanmak amacıyla yüklenmiştir.

# numpy kütüphanesi çok boyutlu dizileri ve matrisleri kullanmak için, bu diziler üzerinde çalışacak üst düzey matematiksel işlevler yapmak için kullanılacaktır.

# png dosyalarının gösterilmesi ve veri hakkındaki istatistiklerin incelenmesi amacıyla matplotlib ve seaborn kütüphaneleri kullanılmıştır

# os kütüphanesi sayesinde verilerin bulunduğu klasör ve dosyalar üzerinde çalışmalar yapılmıştır.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import os

import seaborn as sns

sns.set()

# Bu iki komutun bir arada kullanılmasıyla, GitHub üzerindeki ChestX-ray8 dataset veri setine erişilerek yüklenmiştir.

!git clone https://github.com/hardik0/AI-for-Medicine-Specialization

%cd AI-for-Medicine-Specialization/AI-for-Medical-Diagnosis/

%ls nih/

# Read csv file containing training datadata

train\_df = pd.read\_csv("nih/train-small.csv")

# Print first 5 rows

print(f'There are {train\_df.shape[0]} rows and {train\_df.shape[1]} columns in this data frame')

train\_df.head()

# bu aşamada görüntüleri açıklayan ve her bir png dosyası için de tanımlayıcı bilgiler içeren csv dosyası yüklenmiştir.

# yüklenen csv dosyası Görüntü Atelektazi Kardiyomegali Konsolidasyon Ödem Efüzyon Amfizem Fibroz Fıtık İnfiltrasyon Kitle Nodül Hasta Plevral Kalınlaşma Zatürre Pnömotoraks bilgilerini içermektedir.

# bu sütunlar 1 veya 0 ile doldurulmaktadır. Eğer söz konusu parametre hastada mevcutsa 1 değilse 0 olarak doldurulmuştur.

# Look at the data type of each column and whether null values are present

# veri setinin boyutları

# veri tipleri

# eksik değerler

# bellek kullanımı

# veri hakkında genel bilgi elde etmek amacıyla kullanılmıştır.

train\_df.info()

# Elde edilen DataFrame'de 0'dan 999'a kadar dizine eklenmiş 1000 giriş (satır) vardır.

# Veri Sütunları: DataFrame'de 16 sütun (özellik) vardır.

# Boş Olmayan Sayım: Bu durumda, tüm sütunlarda 1000 boş olmayan değer vardır, yani eksik giriş yoktur.

# Dtype: Her sütunun veri türü. Burada, 15 sütun int64 (tamsayılar) türündedir ve 1 sütun (object) nesne türündedir. sütun adı Image

# Bellek Kullanımı: DataFrame tarafından kullanılan toplam bellek yaklaşık 125,1 KB'dir.

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Image 1000 non-null object

1 Atelectasis 1000 non-null int64

2 Cardiomegaly 1000 non-null int64

3 Consolidation 1000 non-null int64

4 Edema 1000 non-null int64

5 Effusion 1000 non-null int64

6 Emphysema 1000 non-null int64

7 Fibrosis 1000 non-null int64

8 Hernia 1000 non-null int64

9 Infiltration 1000 non-null int64

10 Mass 1000 non-null int64

11 Nodule 1000 non-null int64

12 PatientId 1000 non-null int64

13 Pleural\_Thickening 1000 non-null int64

14 Pneumonia 1000 non-null int64

15 Pneumothorax 1000 non-null int64

dtypes: int64(15), object(1)

memory usage: 125.1+ KB

# train\_df DataFrame'indeki toplam hasta kimliği sayısını sayar.

# Aynı DataFrame'deki benzersiz hasta kimliği sayısını sayar.

# Her iki sayımı da görüntüleyen biçimlendirilmiş bir dize yazdırır.

print(f"The total patient ids are {train\_df['PatientId'].count()}, from those the unique ids are {train\_df['PatientId'].value\_counts().shape[0]} ")

# bu kod çalıştıktan sonra 1000 satır veri içerisinde 928 adet hastanın verisinin olduğu gözlemlenmiştir. bu demek olur ki bazı hastalara ait birden fazla kayıt bulunmaktadır.

# bir hastanın birden fazla kez hastane ziyaretinin olması beklenen bir durumdur. ek olarak ilk hastane ziyaretinde sağlıklı olan birisi ikinci ziyaretinde hasta olabilir.

# her görüntü ayrı olarak CSV dosyasında tutulduğundan ve her görüntüye ait detaylar da csv dosyası içerisinde sağlandığından, bu durum bir sorun oluşturmaz.

# yapılan etiketlemeler bir hastaya ait birden fazla görüntü olması durumunda oluşacak sorunları engellemiştir.

# veri setinde yer alan sütunlar bize yapılan etiketleme işlemleri hakkında bilgi verir.

# bu sütunların hepsi ayrı ayrı etiket değildir. sütunlar arasında tanımlayıcı bilgiler de yer almaktadır.

columns = train\_df.keys()

columns = list(columns)

print(columns)

# veri setindeki sütunlar arasından kimlik tanımlayıcı bilgiler atıldığında elde edilen sütunlar etiketlenmiş verileri içerir

# Remove unnecesary elements

columns.remove('Image')

columns.remove('PatientId')

# Get the total classes

print(f"There are {len(columns)} columns of labels for these conditions: {columns}")

# her bir etiket için elimizde ne kadar veri olduğu önemli bir durumdur.

# eğer elimizde bir veri için az veya dengesiz miktarda veri varsa o zaman bu durum ek geliştirme gerektirebileceğinden çalışmanın başlarında kontrol edilmesi gereken bir durumdur.

# Print out the number of positive labels for each class

for column in columns:

print(f"The class {column} has {train\_df[column].sum()} samples")

# elde edilen veri seti dengeli değildir.

# Dengeli bir veri setinde her sınıfta yaklaşık olarak eşit sayıda örnek olurdu.

# Ancak bu durumda önemli bir dengesizlik vardır.

# "İnfiltrasyon" ve "Efüzyon" gibi bazı sınıflar, "Fıtık" ve "Zatürre" gibi sınıflara kıyasla çok daha fazla sayıda örneğe sahiptir.

# Bu dengesizlik, makine öğrenimi modellerinin performansını etkiler,

# çünkü bunlar çoğunluk sınıflarına doğru önyargılı olacaktır.

# Eğitim sırasında aşırı örnekleme, yetersiz örnekleme veya sınıf ağırlıklandırma gibi tekniklerle bu dengesizliği gidermek önemlidir.

The class Atelectasis has 106 samples

The class Cardiomegaly has 20 samples

The class Consolidation has 33 samples

The class Edema has 16 samples

The class Effusion has 128 samples

The class Emphysema has 13 samples

The class Fibrosis has 14 samples

The class Hernia has 2 samples

The class Infiltration has 175 samples

The class Mass has 45 samples

The class Nodule has 54 samples

The class Pleural\_Thickening has 21 samples

The class Pneumonia has 10 samples

The class Pneumothorax has 38 samples

# train\_df DataFrame'inden 'Image' sütunu seçilir.

# Bu sütun, muhtemelen tüm görüntülerin dosya yollarını veya benzersiz kimliklerini içerir.

# Seçilen 'Image' sütunundaki değerler, daha sonraki işlemler için kullanılabilecek bir NumPy dizisine dönüştürülür.

# Bu dizi, images adlı bir değişkene atanır.

# Bu kodun temel amacı, görüntülerin dosya yollarını veya kimliklerini içeren bir listeyi elde etmektir.

# Bu liste, daha sonra görüntülerin yüklenmesi, işlenmesi veya bir makine öğrenimi modeline beslenmesi gibi işlemler için kullanılabilecektir.

# Extract numpy values from Image column in data frame

images = train\_df['Image'].values

# bu resimlerin içerisinden rasgele 9 örnek listelenmiştir.

# Extract 9 random images from it

random\_images = [np.random.choice(images) for i in range(9)]

# Bu kod, bir veri setindeki görüntülerin rastgele bir örneğini hızlıca görselleştirmek için kullanılır.

# Bu sayede, veri setinin kalitesi, çeşitliliği ve herhangi bir anomali olup olmadığı hakkında genel bir fikir edinilebilir.

# Location of the image dir

img\_dir = 'nih/images\_small/'

print('Display Random Images')

# Adjust the size of your images

plt.figure(figsize=(20,10))

# Iterate and plot random images

for i in range(9):

plt.subplot(3, 3, i + 1)

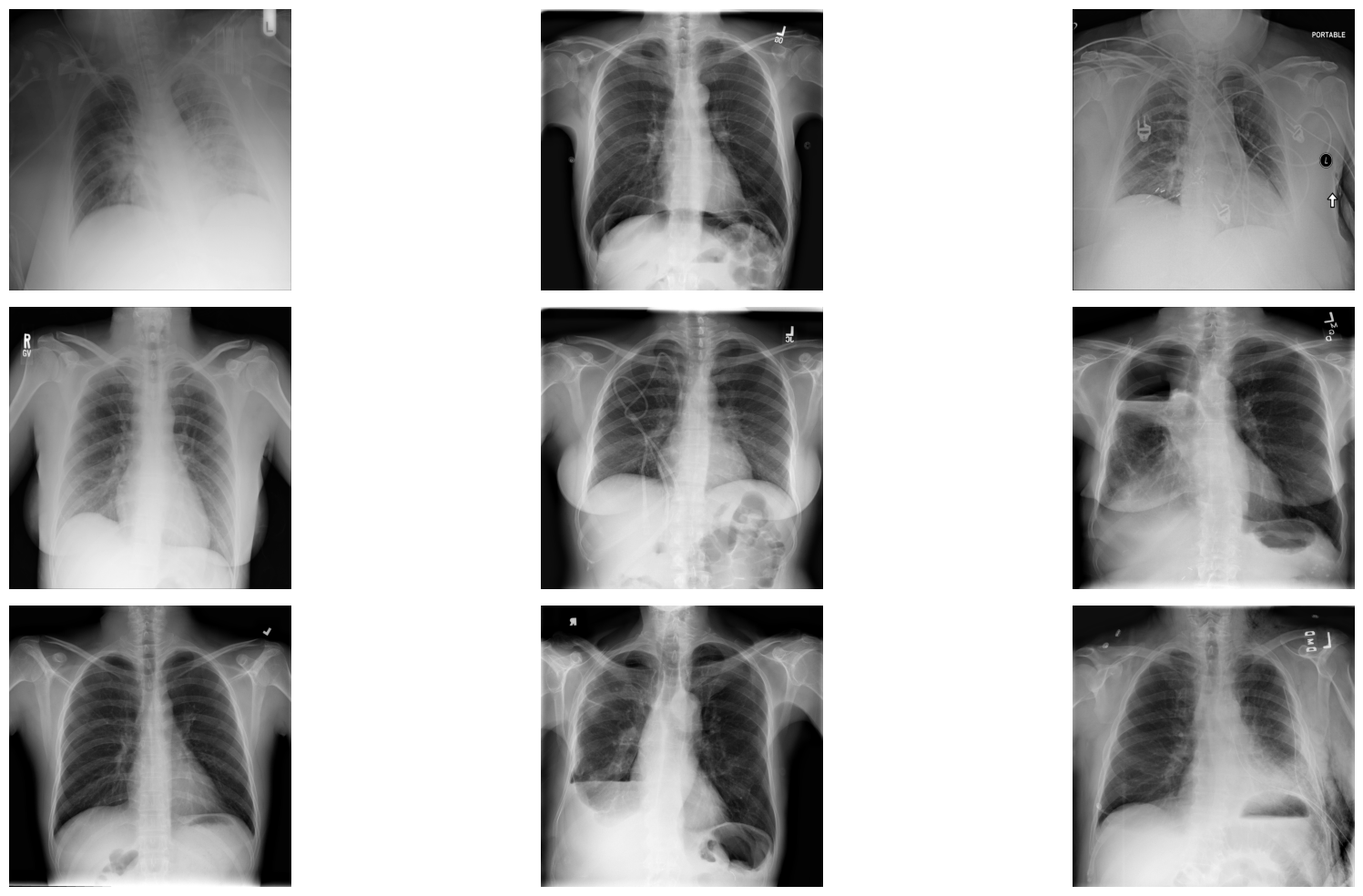
img = plt.imread(os.path.join(img\_dir, random\_images[i]))

plt.imshow(img, cmap='gray')

plt.axis('off')

# Adjust subplot parameters to give specified padding

plt.tight\_layout()



# belirli bir veri çerçevesinden (train\_df) alınan ilk görüntüyü yükler, görüntüler ve temel görüntü özelliklerini ekrana yazdırır.

# train\_df DataFrame'inden ilk görüntü yolunu alır ve sample\_img değişkenine atar.

# img\_dir dizinindeki sample\_img yolunda bulunan görüntüyü yükler ve raw\_image değişkenine atar.

# yüklenen görüntüyü gri tonlamalı olarak ekranda gösterir.

# Görüntünün sağ tarafına bir renk çubuğu ekler, böylece piksel yoğunlukları ile renkler arasındaki ilişkiyi görselleştirir.

# Grafiğin başlığı olarak "Raw Chest X Ray Image" yazar.

# Görüntünün genişlik, yükseklik ve renk kanallarını ekrana yazdırır.

# Görüntüdeki en büyük ve en küçük piksel değerlerini ekrana yazdırır.

# Görüntüdeki piksel değerlerinin ortalamasını ve standart sapmasını ekrana yazdırır.

sample\_img = train\_df.Image[0]

raw\_image = plt.imread(os.path.join(img\_dir, sample\_img))

plt.imshow(raw\_image, cmap='gray')

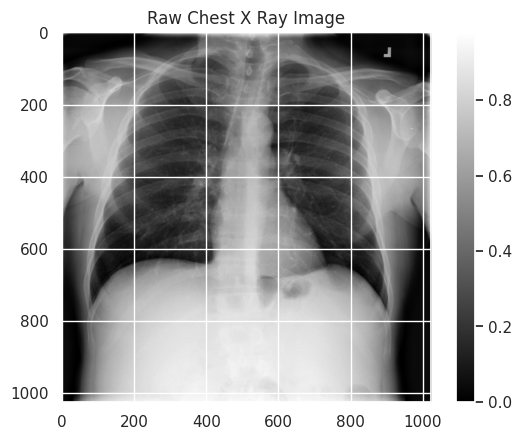
plt.colorbar()

plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {raw\_image.shape[0]} pixels width and {raw\_image.shape[1]} pixels height, one single color channel")

print(f"The maximum pixel value is {raw\_image.max():.4f} and the minimum is {raw\_image.min():.4f}")

print(f"The mean value of the pixels is {raw\_image.mean():.4f} and the standard deviation is {raw\_image.std():.4f}")



img\_dir = 'nih/images\_small/'

random\_images = [np.random.choice(images) for i in range(9)]

for i in range(9):

sample\_img = random\_images[i]

print(sample\_img)

raw\_image = plt.imread(os.path.join(img\_dir, sample\_img))

print(f"The dimensions of the image are {raw\_image.shape[0]} pixels width and {raw\_image.shape[1]} pixels height, one single color channel")

print(f"The maximum pixel value is {raw\_image.max():.4f} and the minimum is {raw\_image.min():.4f}")

print(f"The mean value of the pixels is {raw\_image.mean():.4f} and the standard deviation is {raw\_image.std():.4f}")

print("="\*50)

# bir görüntüdeki piksel yoğunluklarının dağılımını görselleştirmek için kullanılır.

# görüntüyü tek boyutlu bir diziye dönüştürür. Yani, görüntüdeki tüm pikseller tek bir sıraya dizilir.

# Seaborn kütüphanesini kullanarak bir histogram oluşturur. Histogram, piksel yoğunluklarının frekans dağılımını gösterir. kde=False parametresi, yoğunluk eğrisinin çizdirilmesini engeller.

# x eksenine "Pixel Intensity" (Piksel Yoğunluğu) etiketi ekler.

# y eksenine "# Pixels in Image" (Görüntüdeki Piksel Sayısı) etiketi ekler.

# Grafiğin başlığı olarak "Distribution of Pixel Intensities in the Image" (Görüntüdeki Piksel Yoğunluklarının Dağılımı) yazar.

# Grafiğin üst merkezine bir efsane ekler. Efsane, histogramın altında yazılan metni içerir. Bu metin, görüntünün ortalama piksel yoğunluğunu ve standart sapmasını gösterir.

sns.distplot(raw\_image.ravel(),

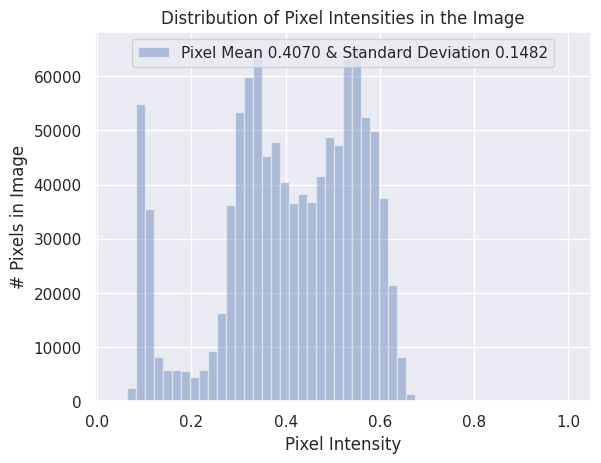
label=f'Pixel Mean {np.mean(raw\_image):.4f} & Standard Deviation {np.std(raw\_image):.4f}', kde=False)

plt.legend(loc='upper center')

plt.title('Distribution of Pixel Intensities in the Image')

plt.xlabel('Pixel Intensity')

plt.ylabel('# Pixels in Image')



img\_dir = 'nih/images\_small/'

random\_images = [np.random.choice(images) for i in range(9)]

for i in range(9):

sample\_img = random\_images[i]

print(sample\_img)

raw\_image = plt.imread(os.path.join(img\_dir, sample\_img))

sns.distplot(raw\_image.ravel(),

label=f'Pixel Mean {np.mean(raw\_image):.4f} & Standard Deviation {np.std(raw\_image):.4f}', kde=False)

plt.legend(loc='upper center')

plt.title(f'Distribution of Pixel Intensities in the Image: {sample\_img}')

plt.xlabel('Pixel Intensity')

plt.ylabel('# Pixels in Image')

plt.show()

# tıbbi görüntüler üzerinde çeşitli görüntü işleme tekniklerini uygulamaktadır.

# Bu teknikler, görüntülerin kontrastını, yoğunluk dağılımını ve genel görünümünü iyileştirmek için kullanılır.

# Görüntüdeki en küçük ve en büyük piksel değerleri belirlenir.

# Piksel değerleri, 0 ile 1 arasında bir aralığa normalize edilir. Bu, görüntünün dinamik aralığını sıkıştırır.

# Normalize edilmiş piksel değerleri, 0 ile 255 arasında bir aralığa ölçeklenir. Bu, görüntüyü 8-bitlik bir formata dönüştürür.

# skimage.exposure.equalize\_hist fonksiyonu kullanılarak, görüntünün histogramı eşitlenir.

# Bu, görüntünün kontrastını artırır ve yoğunluk dağılımını daha düzgün hale getirir.

# Eşitleme işleminden sonra, piksel değerleri 0 ile 255 arasında bir aralığa ölçeklenir.

# gamma değişkeni, gama düzeltme işlemi için bir parametredir.

# Genellikle 1'den büyük değerler, görüntüyü daha parlak hale getirirken, 1'den küçük değerler, görüntüyü daha karanlık hale getirir.

# np.power fonksiyonu kullanılarak, piksel değerleri gama değerine göre güçlendirilir.

# Ardından, sonuç 0 ile 255 arasında bir aralığa ölçeklenir.

# Görüntülerin daha net ve anlaşılır hale getirilmesi için kullanılır.

# Görüntü sınıflandırma veya nesne tespiti gibi görevlerde, daha iyi performans elde etmek için kullanılır.

# Genel görüntü işleme uygulamalarında, görüntülerin kalitesini artırmak için kullanılır.

from skimage import exposure

# Calculate minimum and maximum for contrast stretching

min\_val, max\_val = raw\_image.min(), raw\_image.max()

# Step 1: Apply contrast stretching

min\_val, max\_val = raw\_image.min(), raw\_image.max()

contrast\_stretched = (raw\_image - min\_val) / (max\_val - min\_val) # Normalize to [0, 1]

contrast\_stretched = (contrast\_stretched \* 255).astype(np.uint8) # Scale to [0, 255] for visualization

# Step 2: Apply histogram equalization after contrast stretching

# Step 2: Apply histogram equalization using skimage

equalized\_image = exposure.equalize\_hist(contrast\_stretched) # Returns values in [0, 1]

equalized\_image = (equalized\_image \* 255).astype(np.uint8) # Scale to [0, 255]

# Step 3: Apply gamma correction on the equalized image

gamma = 1.2 # Adjust gamma value as needed

gamma\_corrected = np.power(equalized\_image / 255.0, gamma) \* 255

gamma\_corrected = gamma\_corrected.astype(np.uint8)

# Plot original image and histogram

plt.figure(figsize=(18, 10))

plt.subplot(2, 4, 1)

plt.imshow(raw\_image, cmap='gray')

plt.title('Original Image')

plt.colorbar()

plt.subplot(2, 4, 5)

plt.hist(raw\_image.ravel(), bins=256, color='gray')

plt.title("Original Image Histogram")

# Plot contrast-stretched image and histogram

plt.subplot(2, 4, 2)

plt.imshow(contrast\_stretched, cmap='gray')

plt.title('Contrast-Stretched Image')

plt.colorbar()

plt.subplot(2, 4, 6)

plt.hist(contrast\_stretched.ravel(), bins=256, color='gray')

plt.title("Contrast-Stretched Histogram")

# Plot contrast-equalized image and histogram

plt.subplot(2, 4, 3)

plt.imshow(equalized\_image, cmap='gray')

plt.title('Contrast Equalized Image')

plt.colorbar()

plt.subplot(2, 4, 7)

plt.hist(equalized\_image.ravel(), bins=256, color='gray')

plt.title("Equalized Histogram")

# Plot gamma-corrected image and histogram

plt.subplot(2, 4, 4)

plt.imshow(gamma\_corrected, cmap='gray')

plt.title(f'Gamma-Corrected Image (Gamma={gamma})')

plt.colorbar()

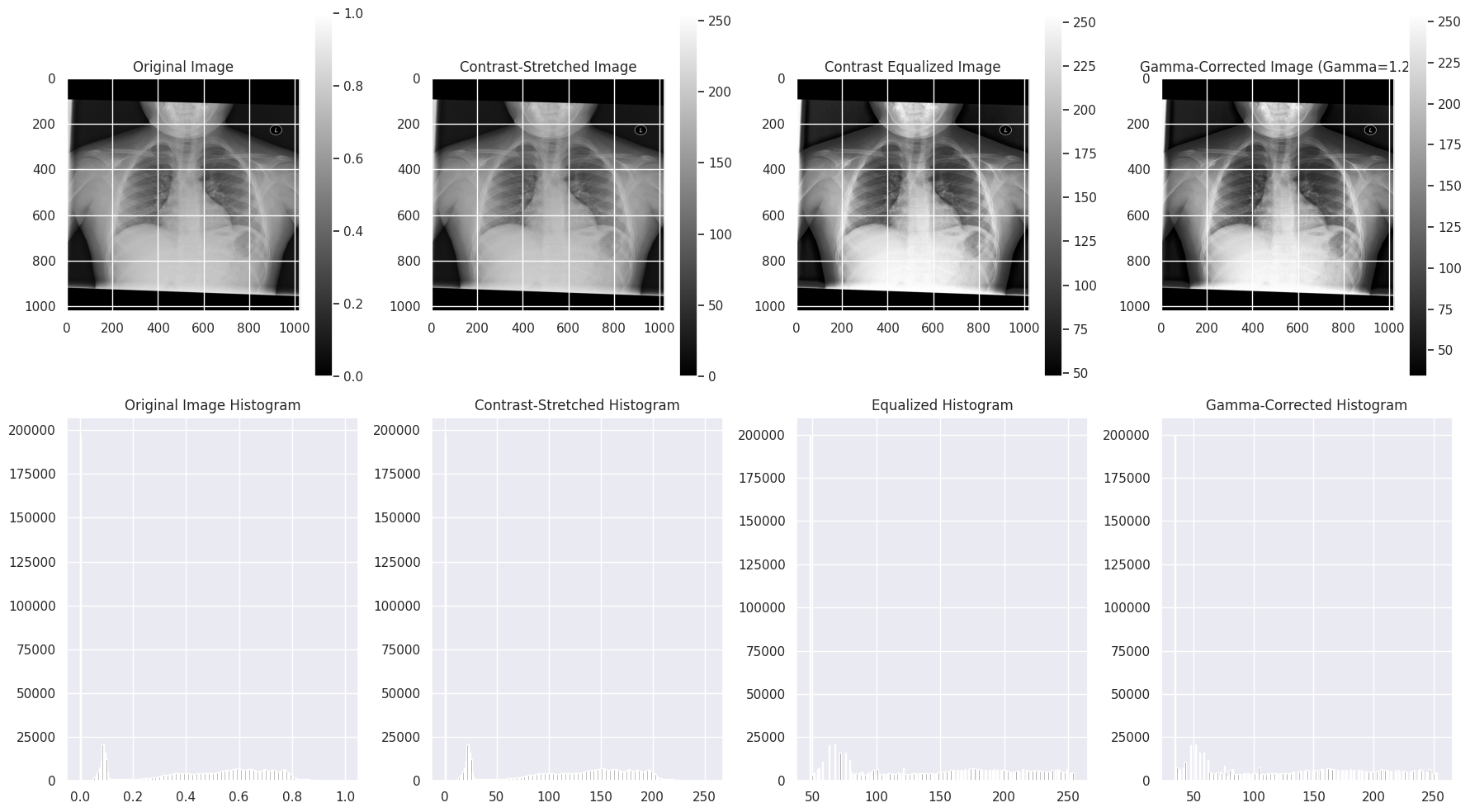
plt.subplot(2, 4, 8)

plt.hist(gamma\_corrected.ravel(), bins=256, color='gray')

plt.title("Gamma-Corrected Histogram")

plt.tight\_layout()

plt.show()



from tqdm import tqdm

img\_dir = 'nih/images\_small/'

gamma\_corrected\_images = []

for i in tqdm(range(len(images))):

sample\_img = images[i]

raw\_image = plt.imread(os.path.join(img\_dir, sample\_img))

min\_val, max\_val = raw\_image.min(), raw\_image.max()

min\_val, max\_val = raw\_image.min(), raw\_image.max()

contrast\_stretched = (raw\_image - min\_val) / (max\_val - min\_val) # Normalize to [0, 1]

contrast\_stretched = (contrast\_stretched \* 255).astype(np.uint8) # Scale to [0, 255] for visualization

equalized\_image = exposure.equalize\_hist(contrast\_stretched) # Returns values in [0, 1]

equalized\_image = (equalized\_image \* 255).astype(np.uint8) # Scale to [0, 255]

gamma = 1.2 # Adjust gamma value as needed

gamma\_corrected = np.power(equalized\_image / 255.0, gamma) \* 255

gamma\_corrected = gamma\_corrected.astype(np.uint8)

gamma\_corrected\_images.append(gamma\_corrected)

# Plot a histogram of the distribution of the pixels

sns.distplot(gamma\_corrected.ravel(),

label=f'Pixel Mean {np.mean(gamma\_corrected):.4f} & Standard Deviation {np.std(gamma\_corrected):.4f}', kde=False)

plt.legend(loc='upper center')

plt.title('Distribution of Pixel Intensities in the gamma\_corrected image')

plt.xlabel('Pixel Intensity')

plt.ylabel('# Pixels in Image')

# OpenCV) kütüphanesini kullanarak, gama düzeltmesi uygulanmış görüntüye gürültü azaltma ve filtreleme teknikleri uygulanmıştır.

# Gürültü Azaltma (Median Filtering)

# Filtreleme (Gaussian Blur)

# işlemleri yapılır.

import cv2 # opencv for python package

# Apply Median Filtering

median\_filtered = cv2.medianBlur(gamma\_corrected, 5)

# Apply Gaussian Blur

gaussian\_filtered = cv2.GaussianBlur(gamma\_corrected, (5, 5), 0)

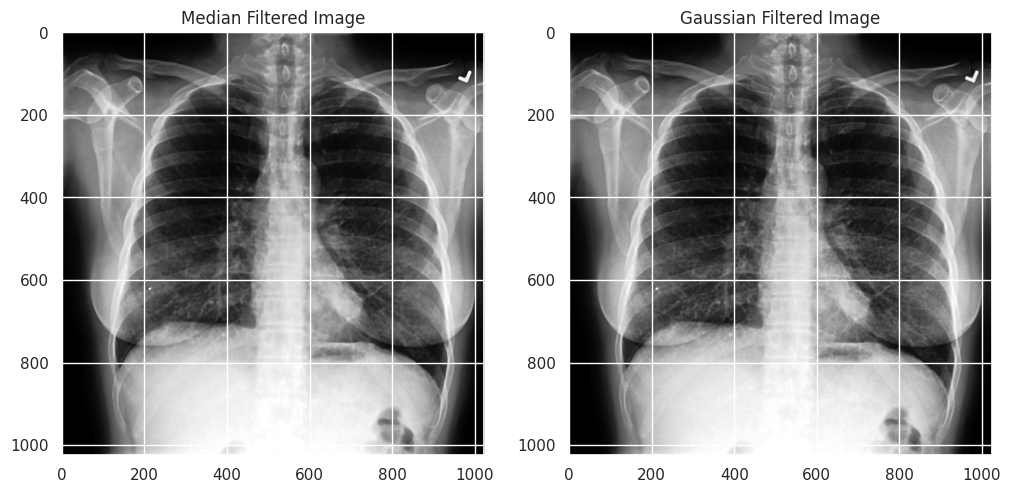
# Display Results

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1), plt.imshow(median\_filtered, cmap='gray'), plt.title("Median Filtered Image")

plt.subplot(1, 2, 2), plt.imshow(gaussian\_filtered, cmap='gray'), plt.title("Gaussian Filtered Image")

plt.show()



median\_filter\_applied\_images = []

gaussian\_filter\_applied\_images = []

for g\_c\_image in tqdm(gamma\_corrected\_images):

median\_filtered = cv2.medianBlur(g\_c\_image, 5)

gaussian\_filtered = cv2.GaussianBlur(g\_c\_image, (5, 5), 0)

median\_filter\_applied\_images.append(median\_filtered)

gaussian\_filter\_applied\_images.append(gaussian\_filtered)

print("Median Filtering Completed", len(median\_filter\_applied\_images))

print("Gaussian Filtering Applied", len(gaussian\_filter\_applied\_images))

# bir görüntüyü belirli bir açıyla döndürmek ve yatay olarak ters çevirmek işlemleri uygulanmıştır.

# Görüntünün merkez noktası hesaplanır.

# Döndürme işlemi için gerekli dönüşüm matrisi oluşturulur.

# Döndürme matrisi kullanılarak, median\_filtered görüntüsü döndürülür ve yeni bir görüntü olan rotated\_image oluşturulur.

# median\_filtered görüntüsü yatay olarak ters çevrilir ve flipped\_horizontal değişkenine atanır. 1 parametresi, yatay ters çevirmeyi belirtir.

# Example: Rotate the image by 10 degrees

center = (median\_filtered.shape[1] // 2, median\_filtered.shape[0] // 2)

rotation\_matrix = cv2.getRotationMatrix2D(center, angle=10, scale=1.0)

rotated\_image = cv2.warpAffine(median\_filtered, rotation\_matrix, (median\_filtered.shape[1], median\_filtered.shape[0]))

# Horizontal Flip

flipped\_horizontal = cv2.flip(median\_filtered, 1)

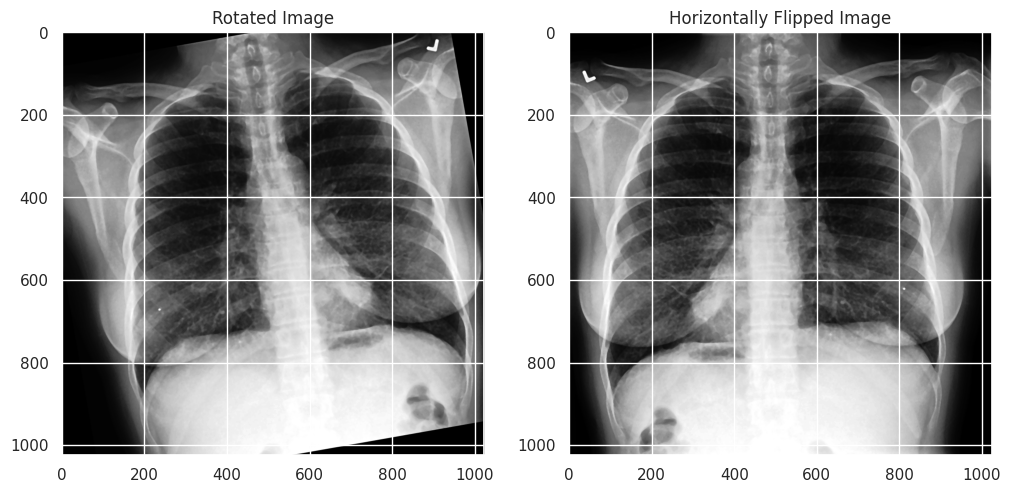
# Display Results

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1), plt.imshow(rotated\_image, cmap='gray'), plt.title("Rotated Image")

plt.subplot(1, 2, 2), plt.imshow(flipped\_horizontal, cmap='gray'), plt.title("Horizontally Flipped Image")

plt.show()



rotated\_images = []

flipped\_images = []

for median\_filtered in tqdm(median\_filter\_applied\_images):

center = (median\_filtered.shape[1] // 2, median\_filtered.shape[0] // 2)

rotation\_matrix = cv2.getRotationMatrix2D(center, angle=10, scale=1.0)

rotated\_image = cv2.warpAffine(median\_filtered, rotation\_matrix, (median\_filtered.shape[1], median\_filtered.shape[0]))

flipped\_horizontal = cv2.flip(median\_filtered, 1)

rotated\_images.append(rotated\_image)

flipped\_images.append(flipped\_horizontal)

print()

print("Rotation Completed", len(rotated\_images))

print("Flipping Completed", len(flipped\_images))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1), plt.imshow(rotated\_image, cmap='gray'), plt.title("Rotated Image")

plt.subplot(1, 2, 2), plt.imshow(flipped\_horizontal, cmap='gray'), plt.title("Horizontally Flipped Image")

plt.show()

# bir görüntüyü Fourier dönüşümü kullanarak frekans uzayına dönüştürür,

# belirli bir frekans bandını filtreler ve ardından ters Fourier dönüşümü ile gerçek uzaya geri dönüştürür.

# Bu işlem, görüntü gürültüsünü azaltmak, kenarları vurgulamak veya diğer görüntü işleme görevleri için kullanılmaktadır

# Apply Fourier Transform and shift it to center low frequencies

dft = cv2.dft(np.float32(rotated\_image), flags=cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shifted = np.fft.fftshift(dft)

# Create a mask for filtering

rows, cols = rotated\_image.shape

crow, ccol = rows // 2 , cols // 2

mask = np.zeros((rows, cols, 2), np.uint8)

mask[crow-30:crow+30, ccol-30:ccol+30] = 1 # Center low frequencies pass

# Apply mask and inverse DFT

fshift = dft\_shifted \* mask

f\_ishift = np.fft.ifftshift(fshift)

filtered\_image = cv2.idft(f\_ishift)

filtered\_image = cv2.magnitude(filtered\_image[:, :, 0], filtered\_image[:, :, 1])

# Normalize and display the filtered image

filtered\_image = cv2.normalize(filtered\_image, None, 0, 255, cv2.NORM\_MINMAX)

plt.imshow(filtered\_image, cmap='gray')

plt.title("Frequency Domain Filtered Image")

plt.axis("off")

plt.show()



filtered\_images = []

for rotated\_image in tqdm(rotated\_images):

dft = cv2.dft(np.float32(rotated\_image), flags=cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shifted = np.fft.fftshift(dft)

rows, cols = rotated\_image.shape

crow, ccol = rows // 2 , cols // 2

mask = np.zeros((rows, cols, 2), np.uint8)

mask[crow-30:crow+30, ccol-30:ccol+30] = 1 # Center low frequencies pass

fshift = dft\_shifted \* mask

f\_ishift = np.fft.ifftshift(fshift)

filtered\_image = cv2.idft(f\_ishift)

filtered\_image = cv2.magnitude(filtered\_image[:, :, 0], filtered\_image[:, :, 1])

filtered\_image = cv2.normalize(filtered\_image, None, 0, 255, cv2.NORM\_MINMAX)

filtered\_images.append(filtered\_image)

print("Filtering Completed", len(filtered\_images))

# bir görüntüyü frekans uzayına dönüştürerek belirli frekans bileşenlerini filtreleyen bir işlem yapar.

# Bu işlem, görüntü gürültüsünü azaltmak,

# kenarları vurgulamak veya görüntüdeki belirli detayları öne çıkarmak için kullanılır

# rotated\_image adlı görüntüyü kompleks Fourier dönüşümü uygulayarak frekans uzayına dönüştürür

# frekans spektrumunu kaydırarak düşük frekans bileşenlerini merkeze taşır

# Görüntünün merkezi hesaplanır

# İki kanallı bir maske oluşturulur.

# Maskenin merkezinde bir dairesel bölge oluşturulur.

# Frekans spektrumu ile maske çarpılarak, istenen frekans bandı filtrelenir.

# Filtrelenmiş frekans spektrumu tekrar orijinal konumuna kaydırılır.

# Ters Fourier dönüşümü uygulanarak, frekans uzayından gerçek uzaya geri dönülür

# Gerçek ve sanal kısımların büyüklüğü alınarak, filtrelenmiş görüntü elde edilir.

# Apply Fourier Transform and shift it to center low frequencies

dft = cv2.dft(np.float32(rotated\_image), flags=cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shifted = np.fft.fftshift(dft)

# Create a mask for filtering

rows, cols = rotated\_image.shape

crow, ccol = rows // 2 , cols // 2

mask = np.zeros((rows, cols, 2), np.uint8)

mask[crow-30:crow+30, ccol-30:ccol+30] = 1 # Center low frequencies pass

# Apply mask and inverse DFT

fshift = dft\_shifted \* mask

f\_ishift = np.fft.ifftshift(fshift)

filtered\_image = cv2.idft(f\_ishift)

filtered\_image = cv2.magnitude(filtered\_image[:, :, 0], filtered\_image[:, :, 1])

# bir görüntüyü daha net ve keskin hale getirmek için iki temel görüntü işleme tekniği olan keskinleştirme ve büyütme işlemlerini uygulanmıştır.

# Görüntünün kenarlarını vurgulamak için tasarlanmış bir matris (çekirdek) tanımlanır.

# cv2.filter2D fonksiyonu kullanılarak, keskinleştirme çekirdeği, görüntü üzerinde kaydırılarak her piksel için bir ağırlıklı ortalama hesaplanır.

# np.clip fonksiyonu ile piksel değerleri 0 ile 255 arasında sınırlanır, böylece piksel değerleri geçerli bir aralıkta kalır.

# scale\_factor değişkeni ile görüntü kaç kat büyütüleceği belirlenir.

# scipy.ndimage.zoom fonksiyonu kullanılarak, keskinleştirilmiş görüntü, bikübik enterpolasyon yöntemi ile büyütülür.

# matplotlib kütüphanesi kullanılarak, orijinal görüntü, keskinleştirilmiş görüntü ve büyütülmüş görüntü yan yana gösterilir.

from scipy.ndimage import zoom

# Step 4: Apply sharpening using an unsharp masking kernel

sharpening\_kernel = np.array([[-1, -1, -1],

[-1, 9, -1],

[-1, -1, -1]])

sharpened\_image = np.clip(cv2.filter2D(gamma\_corrected, -1, sharpening\_kernel), 0, 255).astype(np.uint8)

# Step 5: Apply bicubic interpolation to upscale the image by a factor of 2

scale\_factor = 2

resized\_image = zoom(sharpened\_image, scale\_factor, order=3) # Bicubic interpolation with `order=3`

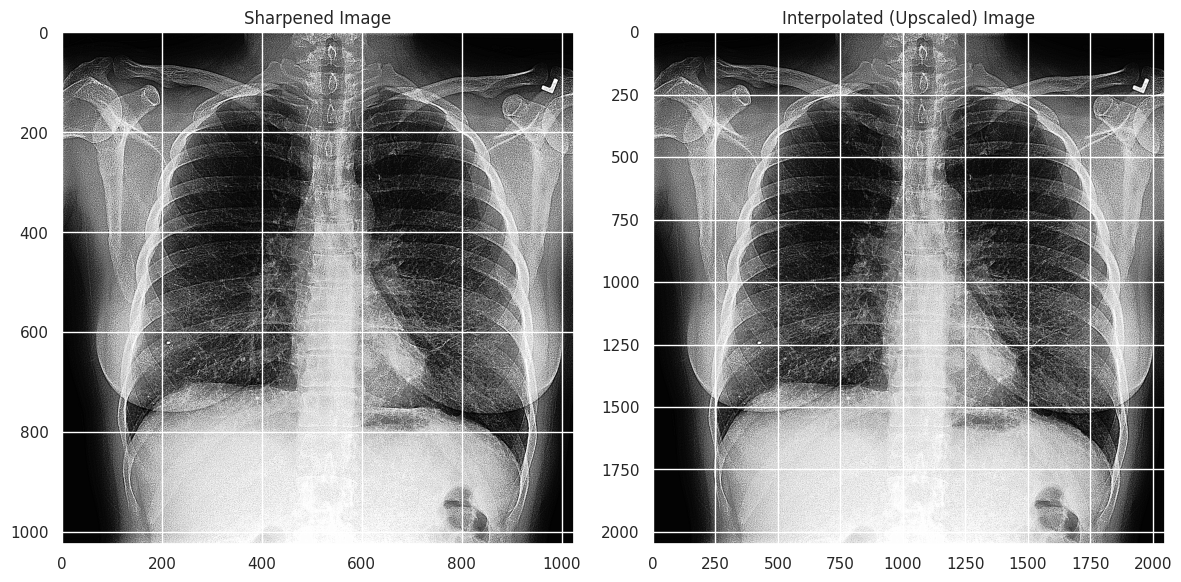
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1), plt.imshow(sharpened\_image, cmap='gray'),plt.title("Sharpened Image")

plt.subplot(1, 2, 2), plt.imshow(resized\_image, cmap='gray'),plt.title("Interpolated (Upscaled) Image")

plt.tight\_layout()

plt.show()



#plt.savefig(f'/content/drive/MyDrive/Ceren/MSc/Digital Image Processing/data/processed/{gamma\_corrected\_imName}'): İşlenmiş görüntü belirtilen dizine kaydedilir.

import cv2

from scipy.ndimage import zoom

sharpening\_kernel = np.array([[-1, -1, -1],

[-1, 9, -1],

[-1, -1, -1]])

for gamma\_corrected\_imName in tqdm(gamma\_corrected\_images.keys()):

sharpened\_image = np.clip(cv2.filter2D(gamma\_corrected\_images[gamma\_corrected\_imName], -1, sharpening\_kernel), 0, 255).astype(np.uint8)

scale\_factor = 2

resized\_image = zoom(sharpened\_image, scale\_factor, order=3) # Bicubic interpolation with `order=3`

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1), plt.imshow(sharpened\_image, cmap='gray'),plt.title("Sharpened Image")

plt.subplot(1, 2, 2), plt.imshow(resized\_image, cmap='gray'),plt.title("Interpolated (Upscaled) Image")

plt.tight\_layout()

plt.savefig(f'/content/drive/MyDrive/Ceren/MSc/Digital Image Processing/data/processed/{gamma\_corrected\_imName}')