Question 1

In [2]:
```python
import cv2
import numpy as np
```

In [3]:
```python
import math
```

In [4]:
```python
import matplotlib.pyplot as plt
```

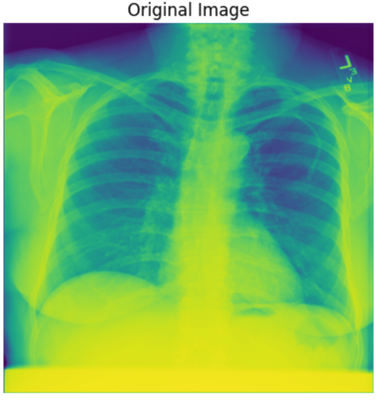
In [5]:
```python
xray_img = cv2.imread("xray.png",-1)
xray_img.shape
```

Out[5]:
```
(1024, 1024)
```

In [34]:
```python
plt.imshow(xray_img)
plt.title("Original Image")
plt.axis('off')
```

Out[34]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

Original Image



In [56]:
```python
noisy_img = cv2.imread("xray_sp.png",-1)
plt.imshow(noisy_img,cmap='gray')
plt.title("Noisy Image")
plt.axis('off')
```

Out[56]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

Noisy Image

In [182…
```python
#median filter
median_filtered_img = cv2.medianBlur(noisy_img,3)
plt.imshow(median_filtered_img,cmap='gray')
plt.axis('off')
```

Out[182]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [37]:
```python
cv2.imwrite("median_filtered_img.png",median_filtered_img)
```

Out[37]:
```
True
```

In [48]:
```python
def rmse(img1,img2):
    mse = np.square(np.subtract(img1,img2)).mean()
    rmse = math.sqrt(mse)
    return rmse
```

In [46]:
```python
rmse_median_filter = rmse(xray_img,median_filtered_img)
```

In [50]:
```python
print(f"RMSE of Median Filtered Image: {rmse_median_filter}")
```

```
RMSE of Median Filtered Image: 1.017037216780872
```

In [49]:
```python
def snr(filtered_img,original_img):
    num = np.square(filtered_img).sum()
    diff_sq = np.square(filtered_img - original_img)
    denom = np.sum(diff_sq)
    snr = num/denom
    return snr
```

In [160…
```python
snr_median_filter = snr(median_filtered_img,xray_img)
```

In [161…
```python
print(f"SNR of Median Filtered Image: {snr_median_filter}")
```

```
SNR of Median Filtered Image: 104.28556070845742
```

In [58]:
```python
def zero_padding(img,kernel_size):
    h,w = img.shape
    padding = kernel_size//2
    new_h = h+padding*2
    new_w = padding*2 + w
    temp = np.zeros((new_h,new_w))
    for i in range(h):
        for j in range(w):
            temp[i+padding,j+padding] = img[i,j]
    return temp
```

In [198…
```python
def max_filter(img,kernel_size):
    padded_img = zero_padding(img,kernel_size)
    padding = kernel_size//2
    final = np.ones((padded_img.shape[0]-2*padding,padded_img.shape[1]-2*padding))
    for i in range(padding,padded_img.shape[0]-padding):
        for j in range(padding,padded_img.shape[1]-padding): #i,j is the centre pixel
            temp = padded_img[i-padding:i+padding+1,j-padding:j+padding+1]
            max_temp = np.max(temp)
            final[i-padding,j-padding] = max_temp
    return final
```

In [199…
```python
max_filtered_img = max_filter(noisy_img,3)
```

In [200…
```python
max_filtered_img.shape
```

Out[200]:
```
(1024, 1024)
```

In [201…
```python
plt.imshow(max_filtered_img,cmap="gray")
plt.axis('off')
```

Out[201]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [202…
```python
cv2.imwrite("Max_Filter.png",max_filtered_img)
```

Out[202]:
```
True
```

In [203…
```python
rmse_max_filter = rmse(xray_img,max_filtered_img)
snr_max_filter = snr(max_filtered_img,xray_img)
```

In [204…
```python
print(f"RMSE of Max Filter: {rmse_max_filter}")
print(f"SNR of Max Filter: {snr_max_filter}")
```

```
RMSE of Max Filter: 25.95361160474386
SNR of Max Filter: 52.47334541130218
```

In [205…
```python
def min_filter(img,kernel_size):
    padded_img = zero_padding(img,kernel_size)
    padding = kernel_size//2
    final = np.ones((padded_img.shape[0]-2*padding,padded_img.shape[1]-2*padding))
    for i in range(padding,padded_img.shape[0]-padding):
        for j in range(padding,padded_img.shape[1]-padding): #i,j is the centre pixel
            temp = padded_img[i-padding:i+padding+1,j-padding:j+padding+1]
            max_temp = np.min(temp)
            final[i-padding,j-padding] = max_temp
    return final
```

In [206…
```python
min_filtered_img = min_filter(noisy_img, 3)
```

In [207…
```python
plt.imshow(min_filtered_img,cmap="gray")
plt.axis('off')
```

Out[207]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [208…
```python
cv2.imwrite("Min_Filter.png",min_filtered_img)
```

Out[208]:
```
True
```

In [209…
```python
rmse_min_filter = rmse(xray_img,min_filtered_img)
snr_min_filter = snr(min_filtered_img,xray_img)
print(f"RMSE of Min Filter: {rmse_min_filter}")
print(f"SNR of Min Filter: {snr_min_filter}")
```

```
RMSE of Min Filter: 20.75023186795397
SNR of Min Filter: 73.26402189026705
```

In [218…
```python
min_after_max_filter = min_filter(max_filtered_img,5)
```

In [219…
```python
plt.imshow(min_after_max_filter,cmap='gray')
plt.axis('off')
```

Out[219]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [220…
```python
cv2.imwrite("Min Filter on Max Filtered Image.png",min_after_max_filter)
```

Out[220]:
```
True
```

Therefore, a combination of max and min filters gives a good result for salt and pepper noise

In [222…
```python
rmse_min_max_filter = rmse(xray_img,min_after_max_filter)
snr_min_max_filter = snr(min_after_max_filter,xray_img)
print(f"RMSE of Min Filter: {rmse_min_max_filter}")
print(f"SNR of Min Filter: {snr_min_max_filter}")
```

```
RMSE of Min Filter: 12.904896921531721
SNR of Min Filter: 193.65637872382547
```

In [54]:
```python
def alpha_trim(arr,d):
    l = arr.flatten()
    l.sort()
    p = arr.flatten()
    temp = []
    count = 0
    for i in range(0, d//2):
        temp.append(l[count])
        temp.append(l[l.shape[0] - count - 1])
        count += 1
    for i in temp:
        for j in range(len(p)):
            if i == p[j]:
                p[j] = 0
    p = p.reshape(arr.shape)
    return p
```

In [59]:
```python
def alpha_trimmed_mean_filter(img,d,kernel_size):
    padded_img = zero_padding(img,kernel_size)
    padding = kernel_size//2
    final = np.ones((padded_img.shape[0]-2*padding,padded_img.shape[1]-2*padding))
    for i in range(padding,padded_img.shape[0]-padding):
        for j in range(padding,padded_img.shape[1]-padding): #i,j is the centre pixel
            temp = padded_img[i-padding:i+padding+1,j-padding:j+padding+1]
            alpha_trim_temp = alpha_trim(temp,d)
            final[i-padding,j-padding] = alpha_trim_temp.sum()/(kernel_size**2 - d)
    return final
```

In [60]:
```python
alpha_trimmed_xray_img = alpha_trimmed_mean_filter(noisy_img,0,3)
```

In [61]:
```python
alpha_trimmed_xray_img = cv2.normalize(alpha_trimmed_xray_img,None,0,255,cv2.NORM_MINMAX,cv2.CV_8UC1)
```

In [62]:
```python
plt.imshow(alpha_trimmed_xray_img,cmap='gray')
plt.axis('off')
```

Out[62]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [63]:
```python
cv2.imwrite("Alpha Trimmed Mean Filter.png",alpha_trimmed_xray_img)
```

Out[63]:
```
True
```

In [64]:
```python
rmse_alpha_filter = rmse(xray_img,alpha_trimmed_xray_img)
snr_alpha_filter = snr(alpha_trimmed_xray_img,xray_img)
print(f"RMSE of Min Filter: {rmse_alpha_filter}")
print(f"SNR of Min Filter: {snr_alpha_filter}")
```

```
RMSE of Min Filter: 2.9916151809158267
SNR of Min Filter: 11.873326829704924
```

Question 2

In [9]:
```python
noisy = cv2.imread("xray_g.png",-1)
```

In [10]:
```python
noisy_float = np.float64(noisy)
xray_fourier = cv2.dft(noisy_float,flags = cv2.DFT_COMPLEX_OUTPUT)
```

In [11]:
```python
xray_fourier_shift = np.fft.fftshift(xray_fourier)
xray_fourier_shift.shape
```

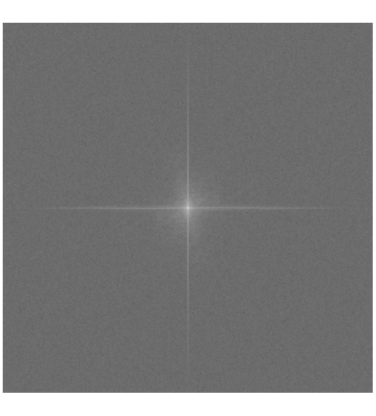
Out[11]:
```
(1024, 1024, 2)
```

In [12]:
```python
magnitude = 20*np.log(cv2.magnitude(xray_fourier_shift[:,:,0],xray_fourier_shift[:,:,1]))
```

In [13]:
```python
magnitude= cv2.normalize(magnitude,None,0,255,cv2.NORM_MINMAX,cv2.CV_8UC1)
```

In [14]:
```python
plt.imshow(magnitude,cmap='gray')
plt.axis('off')
```

Out[14]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [15]:
```python
#box filter
box = np.zeros(xray_fourier_shift.shape)
M,N = xray_img.shape
```

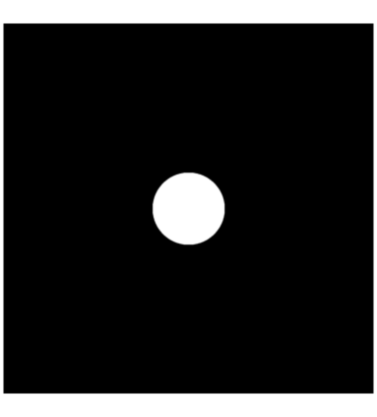
In [16]:
```python
for i in range(0, M):
    for j in range(0,N):
        z = math.sqrt(((M/2-i)**2 + (N/2-j)**2))
        if z<100:
            box[i,j] = 1
```

In [17]:
```python
box.shape
```

Out[17]:
```
(1024, 1024, 2)
```

In [315…
```python
plt.imshow(box,cmap='gray')
plt.axis('off')
```

Out[315]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [32]:
```python
#gaussian
D0 = 100
gaussian = np.zeros(xray_fourier_shift.shape)
for i in range(0, M):
    for j in range(0,N):
        z = math.sqrt(((M/2-i)**2 + (N/2-j)**2))
        gaussian[i,j] = math.exp((-z**2)/(2*D0**2))
```

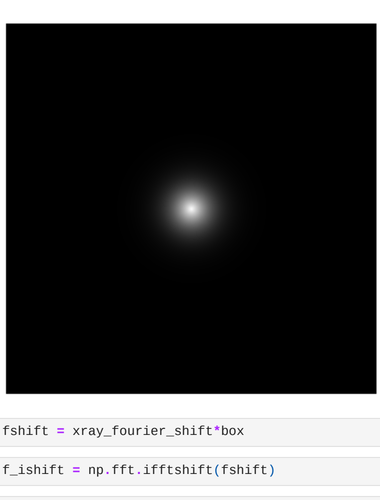
In [358…
```python
plt.imshow(gaussian,cmap='gray')
plt.axis('off')
```

Out[358]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [43]:
```python
#Butterworth LPF
D0 = 300
n = 3
butter = np.zeros(xray_fourier_shift.shape)
for i in range(0, M):
    for j in range(0,N):
        z = math.sqrt(((M/2-i)**2 + (N/2-j)**2))
        butter[i,j] = 1/(1+(math.exp(z/D0))**(2*n))
```

In [364…
```python
plt.imshow(butter,cmap='gray')
plt.axis('off')
```

Out[364]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [20]:
```python
fshift = xray_fourier_shift*box
```

In [21]:
```python
f_ishift = np.fft.ifftshift(fshift)
```

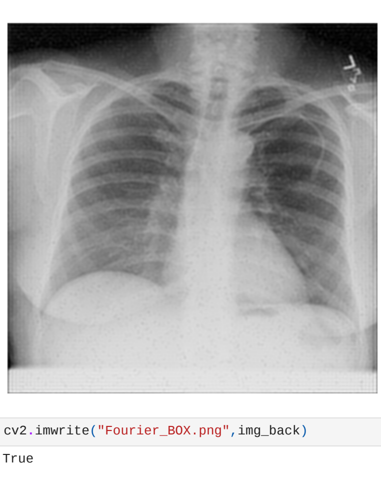
In [22]:
```python
img_back = cv2.idft(f_ishift)
```

In [23]:
```python
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
```

In [29]:
```python
img_back = cv2.normalize(img_back,None,0,255,cv2.NORM_MINMAX,cv2.CV_8UC1)
```

In [30]:
```python
plt.imshow(img_back,cmap='gray')
plt.axis('off')
```

Out[30]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



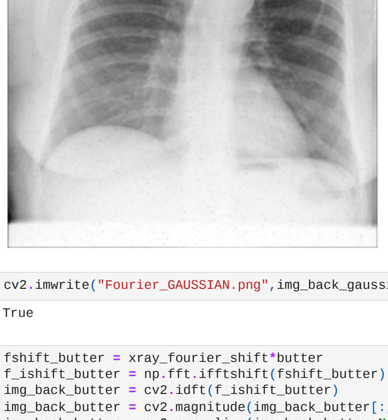
In [31]:
```python
cv2.imwrite("Fourier_BOX.png",img_back)
```

Out[31]:
```
True
```

In [34]:
```python
fshift_gaussian = xray_fourier_shift*gaussian
f_ishift_gaussian = np.fft.ifftshift(fshift_gaussian)
img_back_gaussian = cv2.idft(f_ishift_gaussian)
img_back_gaussian = cv2.magnitude(img_back_gaussian[:,:,0],img_back_gaussian[:,:,1])
img_back_gaussian = cv2.normalize(img_back_gaussian,None,0,255,cv2.NORM_MINMAX,cv2.CV_8UC1)
```

In [35]:
```python
plt.imshow(img_back_gaussian,cmap='gray')
plt.axis('off')
```

Out[35]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



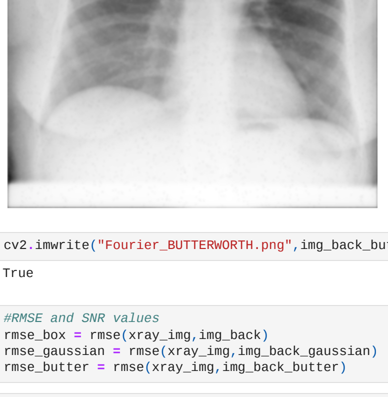
In [36]:
```python
cv2.imwrite("Fourier_GAUSSIAN.png",img_back_gaussian)
```

Out[36]:
```
True
```

In [44]:
```python
fshift_butter = xray_fourier_shift*butter
f_ishift_butter = np.fft.ifftshift(fshift_butter)
img_back_butter = cv2.idft(f_ishift_butter)
img_back_butter = cv2.magnitude(img_back_butter[:,:,0],img_back_butter[:,:,1])
img_back_butter = cv2.normalize(img_back_butter,None,0,255,cv2.NORM_MINMAX,cv2.CV_8UC1)
```

In [45]:
```python
plt.imshow(img_back_butter,cmap='gray')
plt.axis('off')
```

Out[45]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [46]:
```python
cv2.imwrite("Fourier_BUTTERWORTH.png",img_back_butter)
```

Out[46]:
```
True
```

In [51]:
```python
#RMSE and SNR values
rmse_box = rmse(xray_img,img_back)
rmse_gaussian = rmse(xray_img,img_back_gaussian)
rmse_butter = rmse(xray_img,img_back_butter)
```

In [52]:
```python
snr_box = snr(img_back,xray_img)
snr_gaussian = snr(img_back_gaussian,xray_img)
snr_butter = snr(img_back_butter,xray_img)
```

In [53]:
```python
print(f"Box Filter: RMSE = {rmse_box} | SNR = {snr_box}")
print(f"Gaussian Filter: RMSE = {rmse_gaussian} | SNR = {snr_gaussian}")
print(f"Butterworth Filter: RMSE = {rmse_butter} | SNR = {snr_butter}")
```

```
Box Filter: RMSE = 10.242325445042047 | SNR = 1.033821295998412
Gaussian Filter: RMSE = 4.013769292461323 | SNR = 6.5852544142753295
Butterworth Filter: RMSE = 4.275206706978054 | SNR = 5.871659126538234
```

In [ ]:

In [1]:
```python
import cv2
import numpy as np
```

In [2]:
```python
import math
```

In [3]:
```python
import matplotlib.pyplot as plt
```

In [4]:
```python
xray = cv2.imread('xray.png',-1)
```

In [5]:
```python
def zigzag(matrix):
    # Get the number of rows and columns in the matrix
    rows = len(matrix)
    cols = len(matrix[0])

    # Create an empty list to store the zigzag ordering
    zigzag = []
    temp = np.zeros((1,rows*cols))

    # Initialize the starting row and column
    row, col = 0, 0

    # Iterate over each diagonal of the matrix
    for i in range(rows + cols - 1):
        # If the diagonal is even, traverse it from top-right to bottom-left
        if i % 2 == 0:
            while row >= 0 and col < cols:
                zigzag.append(matrix[row][col])
                row -= 1
                col += 1
            if row < 0 and col <= cols - 1:
                row = 0
            if col == cols:
                row += 2
                col -= 1
        # If the diagonal is odd, traverse it from bottom-left to top-right
        else:
            while col >= 0 and row < rows:
                zigzag.append(matrix[row][col])
                row += 1
                col -= 1
            if col < 0 and row <= rows - 1:
                col = 0
            if row == rows:
                col += 2
                row -= 1
    zigzag = np.array(zigzag)
    zigzag = np.reshape(zigzag,(-1,matrix.shape[0]))

    return zigzag
```

In [6]:
```python
def reverse_zigzag_for8x8(matrix):
    for i in range(135):
        matrix= zigzag(matrix)
    return matrix
```

In [7]:
```python
def run_length_encode(input_string):
    # Initialize the encoded string and the counter
    encoded_string = ''
    count = 1

    # Iterate over each character in the input string
    for i in range(1, len(input_string)):
        if input_string[i] == input_string[i-1]:
            # If the current character is the same as the previous one, increment the counter
            count += 1
        else:
            # If the current character is different from the previous one, add the count and the character to the encoded string
            count_str = str(count)
            encoded_string += count_str + "," +str(input_string[i-1]) + "."
            count = 1

    # Add the final count and character to the encoded string
    encoded_string += str(count) +"," + str(input_string[-1])

    return encoded_string
```

In [8]:
```python
block_size = 4
#quantization matrix:
quantization_matrix8x8 = np.array([[16, 11, 10, 16, 24, 40, 51, 61],
                                [12, 12, 14, 19, 26, 58, 60, 55],
                                [14, 13, 16, 24, 40, 57, 69, 56],
                                [14, 17, 22, 29, 51, 87, 80, 62],
                                [18, 22, 37, 56, 68, 109, 103, 77],
                                [24, 35, 55, 64, 81, 104, 113, 92],
                                [49, 64, 78, 87, 103, 121, 120, 101],
                                [72, 92, 95, 98, 112, 100, 103, 99]])
quantization_matrix = np.array([[1,2,4,8],[2,2,4,8],[4,4,4,8],[8,8,8,8]])
```

In [9]:
```python
#calculation for number of blocks needed
h,w = xray.shape
height = np.int32(h)
width = np.int32(w)
h,w = np.float32(h),np.float32(w)

nbh = np.int32(math.ceil(h/block_size))
nbw = np.int32(math.ceil(w/block_size))

#Height of padded image
H = block_size * nbh
W = block_size * nbw

padded_img = np.zeros((H,W))
padded_img[0:height,0:width] = xray[0:height,0:width]

padded_img_minus128 = padded_img - 128
#cv2.imwrite('uncompressed.bmp',np.uint8(padded_img))
```

In [10]:
```python
plt.imshow(padded_img_minus128,cmap='gray')
plt.axis('off')
```

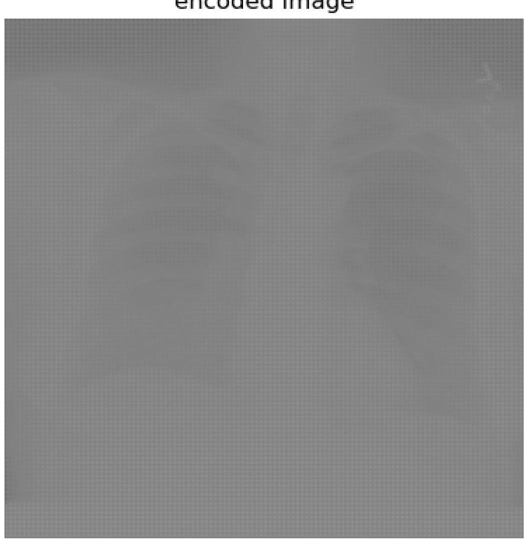
Out[10]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```

In [11]:
```python
final = np.zeros(xray.shape)
for i in range(nbh):
    #compute start and end row index of block
    row_ind_1 = i*block_size
    row_ind_2 = row_ind_1 + block_size


    for j in range(nbw):
        #compute start and end column index of block
        col_ind_1 = j*block_size
        col_ind_2 = col_ind_1 + block_size

        block = padded_img_minus128[row_ind_1 : row_ind_2, col_ind_1:col_ind_2]

        #applying 2D DCT to block
        DCT = cv2.dct(block)

        quantized = np.round(np.divide(DCT,quantization_matrix))

        reordered = zigzag(quantized)

        #reshaped = np.reshape(reordered,(block_size,block_size))

        final[row_ind_1:row_ind_2,col_ind_1:col_ind_2] = reordered
```

In [12]:
```python
plt.imshow(final,cmap='gray')
plt.axis('off')
plt.title("encoded image")
final
```

Out[12]:
```
array([[-408.,  -30.,   -6., ...,    3.,   -2.,   -1.],
       [   4.,   -7.,    1., ...,   -2.,    1.,    0.],
       [   2.,   -1.,    1., ...,   -1.,   -0.,    0.],
       ...,
       [ -25.,  -29.,    5., ...,  -29.,    8.,   -4.],
       [  11.,    5.,   -3., ...,    7.,    1.,    3.],
       [   1.,   -0.,   -1., ...,   -1.,   -1.,    0.]])
```

encoded image



In [13]:
```python
arranged = final.flatten()
```

In [14]:
```python
arranged.shape
```

Out[14]:
```
(1048576,)
```

In [15]:
```python
bitstream = run_length_encode(arranged)
#bitstream
```

In [16]:
```python
bitstream2 = str(padded_img.shape[0]) + " " + str(padded_img.shape[1]) + " " + bitstream + ";"
```

In [17]:
```python
file1 = open('test2.txt','w')
file1.write(bitstream2)
file1.close()
```

In [18]:
```python
def rl_decode(g,shape):

    l = g.split(".")
    final = []
    for i in l:
        temp = i.split(',')
        final.append(np.int32(temp))

    #final
    final_arr = []
    for i in final:
        for j in range(i[0]):
            final_arr.append(i[1])

    k = np.array(final_arr)

    return k.reshape(shape)
```

In [19]:
```python
decoded = rl_decode(bitstream,xray.shape)
decoded
```

Out[19]:
```
array([[-408,  -30,   -6, ...,    3,   -2,   -1],
       [   4,   -7,    1, ...,   -2,    1,    0],
       [   2,   -1,    1, ...,   -1,    0,    0],
       ...,
       [ -25,  -29,    5, ...,  -29,    8,   -4],
       [  11,    5,   -3, ...,    7,    1,    3],
       [   1,    0,   -1, ...,   -1,   -1,    0]])
```

In [20]:
```python
decoded.shape
```

Out[20]:
```
(1024, 1024)
```

In [21]:
```python
decoded,final #should be equal to final
```

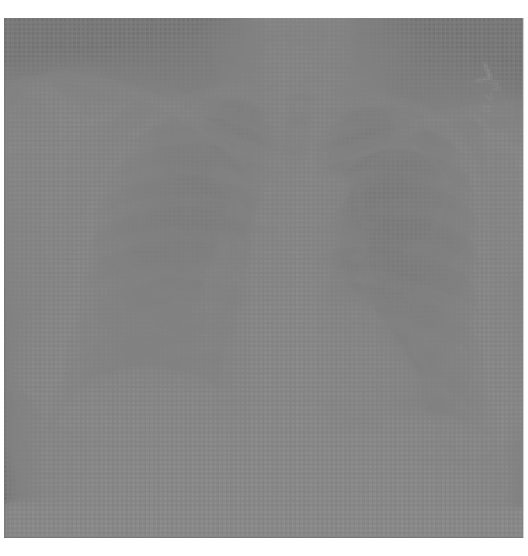
Out[21]:
```
(array([[-408,  -30,   -6, ...,    3,   -2,   -1],
        [   4,   -7,    1, ...,   -2,    1,    0],
        [   2,   -1,    1, ...,   -1,    0,    0],
        ...,
        [ -25,  -29,    5, ...,  -29,    8,   -4],
        [  11,    5,   -3, ...,    7,    1,    3],
        [   1,    0,   -1, ...,   -1,   -1,    0]]),
 array([[-408.,  -30.,   -6., ...,    3.,   -2.,   -1.],
        [   4.,   -7.,    1., ...,   -2.,    1.,    0.],
        [   2.,   -1.,    1., ...,   -1.,   -0.,    0.],
        ...,
        [ -25.,  -29.,    5., ...,  -29.,    8.,   -4.],
        [  11.,    5.,   -3., ...,    7.,    1.,    3.],
        [   1.,   -0.,   -1., ...,   -1.,   -1.,    0.]]))
```

In [22]:
```python
(decoded==final).all()
```

Out[22]:
```
True
```

In [23]:
```python
plt.imshow(decoded,cmap='gray')
plt.axis('off')
```

Out[23]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [24]:
```python
uncompressed = np.zeros(decoded.shape)
for i in range(nbh):
    #compute start and end row index of block
    row_ind_1 = i*block_size
    row_ind_2 = row_ind_1 + block_size



    for j in range(nbw):
        #compute start and end column index of block
        col_ind_1 = j*block_size
        col_ind_2 = col_ind_1 + block_size

        block = decoded[row_ind_1 : row_ind_2, col_ind_1:col_ind_2]

        #applying 2D IDCT to block

        reordered = reverse_zigzag_for8x8(block)

        dequantized_block = np.round(np.multiply(block,quantization_matrix)).astype(np.float64)
        IDCT = cv2.idct(dequantized_block)

        #reshaped = np.reshape(reordered,(block_size,block_size))

        uncompressed[row_ind_1:row_ind_2,col_ind_1:col_ind_2] = IDCT
```

In [25]:
```python
uncompressed2 =np.uint8(uncompressed) + 128
uncompressed2
```

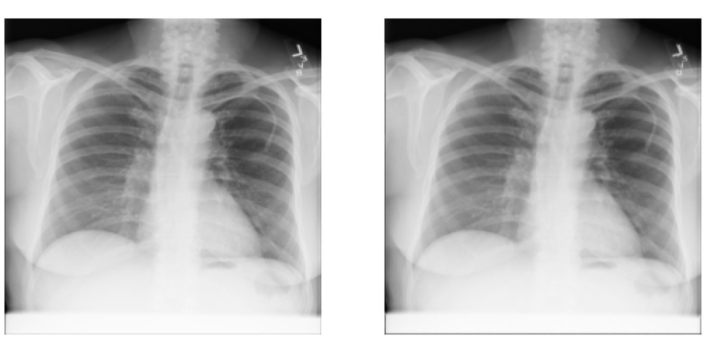
Out[25]:
```
array([[  0,  25,  44, ...,  15,  11,  14],
       [251,  32,  32, ...,  17,  10,   9],
       [251,  36,  26, ...,  19,  11,   7],
       ...,
       [ 11,  74,  67, ..., 227,  55,   8],
       [ 44, 101,  99, ..., 235,  74, 231],
       [103, 123, 144, ..., 210,  65, 166]], dtype=uint8)
```

In [26]:
```python
(xray==uncompressed2).all()
```

Out[26]:
```
False
```

In [27]:
```python
plt.subplot(1,2,1)
plt.imshow(xray,cmap='gray')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(uncompressed2,cmap='gray')
plt.axis('off')
```

Out[27]:
```
(-0.5, 1023.5, 1023.5, -0.5)
```



In [28]:
```python
def rmse(img1,img2):
    mse = np.square(np.subtract(img1,img2)).mean()
    rmse = math.sqrt(mse)
    return rmse

def snr(filtered_img,original_img):
    num = np.square(filtered_img).sum()
    diff_sq = np.square(filtered_img - original_img)
    denom = np.sum(diff_sq)
    snr = num/denom
    return snr
```

In [29]:
```python
rmse_jpeg = rmse(uncompressed2,xray)
snr_jpeg = snr(uncompressed2,xray)
print(f"For JPEG: RMSE = {rmse_jpeg} | SNR = {snr_jpeg}")
```

```
For JPEG: RMSE = 3.782245347350395 | SNR = 7.519069846572177
```

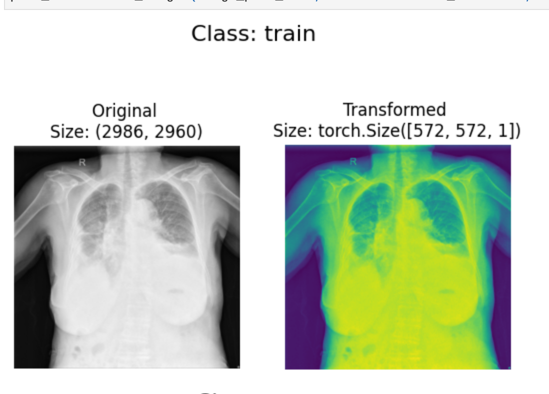
In [30]:
```python
cv2.imwrite("JPEG Reconstructed Img 4x4 block size.jpeg",uncompressed2)
```

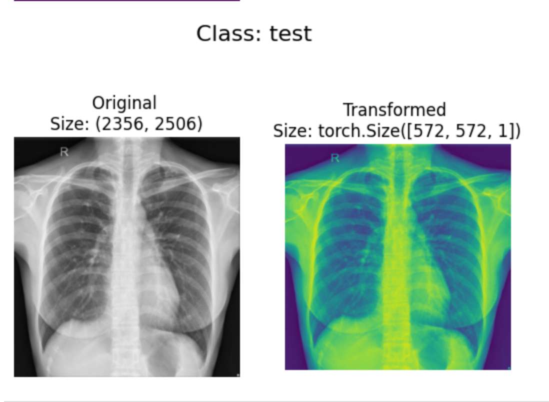
Out[30]:
```
True
```

In [31]:
```python
compression_ratio = 117020/1049654
compression_rate = 1/compression_ratio
compression_ratio, compression_rate
```

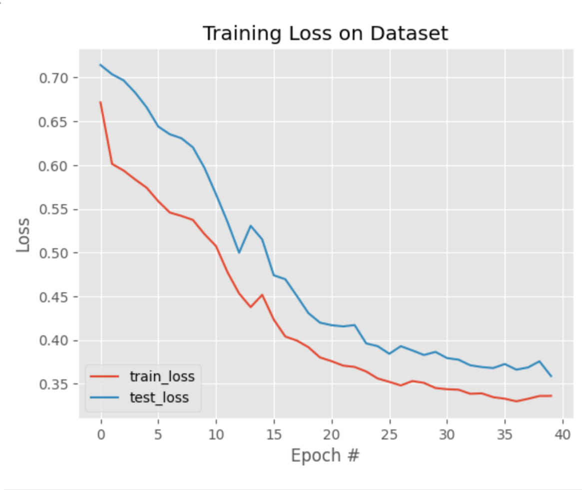
Out[31]:
```
(0.11148435579724367, 8.969868398564348)
```

In [ ]:
```python

```

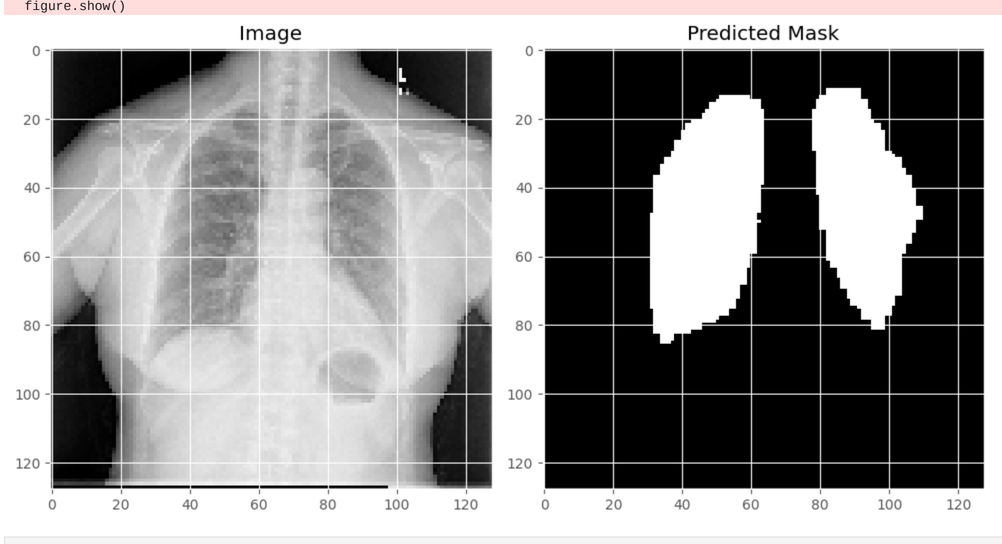
In [2]:
```python
import os
```
In [18]:
```python
import matplotlib.pyplot as plt
```
In [3]:
```python
def walk_through_dir(dir_path):
    """
    Walks through dir_path returning its contents.
    Args:
        dir_path (str or pathlib.Path): target directory

    Returns:
        A print out of:
            number of subdirectories in dir_path
            number of images (files) in each subdirectory
            name of each subdirectory
    """
    for dirpath,dirnames,filenames in os.walk(dir_path):
        print(f"There are {len(dirnames)} directories and {len(filenames)} images in {dirpath}")
```
In [4]:
```python
import requests
import zipfile
from pathlib import Path
```
In [27]:
```python
data_path = Path("data/X_ray_image_and_mask_for_U_net_training")
image_path = data_path / "CXR_png"
masks_path = data_path / "masks"
```
In [6]:
```python
walk_through_dir(image_path)
```
```
There are 2 directories and 0 images in data\X_ray_image_and_mask_for_U_net_training\CXR_png
There are 0 directories and 200 images in data\X_ray_image_and_mask_for_U_net_training\CXR_png\test
There are 0 directories and 600 images in data\X_ray_image_and_mask_for_U_net_training\CXR_png\train
```
In [7]:
```python
#setup train and testing paths
train_dir = image_path / "train"
test_dir = image_path / "test"

train_dir, test_dir
```
Out[7]:
```
(WindowsPath('data/X_ray_image_and_mask_for_U_net_training/CXR_png/train'),
 WindowsPath('data/X_ray_image_and_mask_for_U_net_training/CXR_png/test'))
```
In [10]:
```python
import random
from PIL import Image
```
In [60]:
```python
#set seed
random.seed(42)

#Get all image paths (* means 'any combination')
image_path_list = list(image_path.glob("*/*.png"))

#Get random image path
random_image_path = random.choice(image_path_list)

#Get image class from path name (name of directory)
image_class = random_image_path.parent.stem

#Open image
img = Image.open(random_image_path)

#Print metadata
print(f"Random Image Path: {random_image_path}")
print(f"Image class: {image_class}")
print(f"Image height: {img.height}")
print(f"Image width: {img.width}")
```
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[60], line 8
      5 image_path_list = list(image_path.glob("*/*.png"))
      7 #Get random image path
----> 8 random_image_path = random.choice(image_path_list)
     10 #Get image class from path name (name of directory)
     11 image_class = random_image_path.parent.stem

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.1008.0_x64__qbz5n2kfra8p0\Lib\random.py:373, in Random.choice(self, seq)
    370 # As an accommodation for NumPy, we don't use "if not seq"
    371 # because bool(numpy.array()) raises a ValueError.
    372 if not len(seq):
--> 373     raise IndexError('Cannot choose from an empty sequence')
    374 return seq[self._randbelow(len(seq))]

IndexError: Cannot choose from an empty sequence
```
In [16]:
```python
#Transforming data into tensors, then into torch.utils.data.Dataset
import torch
from torch.utils.data import DataLoader
from torchvision import datasets,transforms
```
In [124…
```python
data_transform = transforms.Compose([transforms.ToPILImage(),
    #Resize the images to 572x572
    transforms.Resize(size = (128,128)),
    #turn the imahe into torch.Tensor
    transforms.ToTensor() #this also converts all pixel values from 0 to 255 to be between 0.0 and 1.0
    ])
```
In [22]:
```python
def plot_transformed_images(image_paths, transform,n=3,seed=42):
    """Plots a series of random images from image_paths.

    Will open n image from image_paths, transform them with transform and
    plot them side by side

    Args:
        image_path (list): List of target image paths.
        transform (PyTorch Transforms): Transforms to apply to images.
        n (int,optional): Number of images to plot. Defaults to 3.
        seed (int, optional): Random seed for the random generator. Defaults to 42
    """
    random.seed(seed)
    random_image_paths = random.sample(image_paths,k=n)
    for image_path in random_image_paths:
        with Image.open(image_path) as f:
            fig, ax = plt.subplots(1,2)
            ax[0].imshow(f)
            ax[0].set_title(f"Original \nSize: {f.size}")
            ax[0].axis('off')

            #transform and plot the image
            #note: permute changes shape of image to suit matplotlib
            #Pytorch default is [C,H,W] but Matplotlib is [H,W,C]
            transformed_image = transform(f).permute(1,2,0)
            ax[1].imshow(transformed_image)
            ax[1].set_title(f"Transformed \nSize: {transformed_image.shape}")
            ax[1].axis('off')

            fig.suptitle(f'Class: {image_path.parent.stem}' , fontsize = 16)

plot_transformed_images(image_path_list,transform = data_transform, n=3)
```




In [24]:
```python
#Using ImageFolder to create datasets
from torchvision import datasets
train_data = datasets.ImageFolder(root = train_dir,
                                  transform = data_transform,
                                  target_transform = None)
test_data = datasets.ImageFolder(root = test_dir,
                                 transform = data_transform)
print(f'Train data:\n{train_data}\nTest data:\n{test_data}')
```
```
Train data:
Dataset ImageFolder
    Number of datapoints: 600
    Root location: data\X_ray_image_and_mask_for_U_net_training\CXR_png\train
    StandardTransform
Transform: Compose(
               Resize(size=(572, 572), interpolation=bilinear, max_size=None, antialias=warn)
               ToTensor()
           )
Test data:
Dataset ImageFolder
    Number of datapoints: 200
    Root location: data\X_ray_image_and_mask_for_U_net_training\CXR_png\test
    StandardTransform
Transform: Compose(
               Resize(size=(572, 572), interpolation=bilinear, max_size=None, antialias=warn)
               ToTensor()
           )
```
In [25]:
```python
train_data.classes
```
Out[25]:
```
['xray']
```
In [26]:
```python
train_data.class_to_idx
```
Out[26]:
```
{'xray': 0}
```
In [115…
```python
#defining U-Net parameters
NUM_CHANNELS = 1
NUM_CLASSES = 1
NUM_LEVELS = 3

#initialize learning rate, number of epochs to train for and batch size
INIT_LR = 0.001
NUM_EPOCHS = 40
BATCH_SIZE = 64

#define the input image dimensions
INPUT_IMAGE_WIDTH = 128
INPUT_IMAGE_HEIGHT = 128

#threshold to filter weak predictions
THRESHOLD = 0.5

#define path to the base output directory
BASE_OUTPUT = "output"

MODEL_PATH = os.path.join(BASE_OUTPUT,"unet_xray.pth")
PLOT_PATH = os.path.sep.join([BASE_OUTPUT,"plot.png"])
TEST_PATHS = os.path.sep.join([BASE_OUTPUT,"test_paths.txt"])
```
In [116…
```python
from torch.utils.data import Dataset
class SegmentationDataset(Dataset):
    def __init__(self,imagePaths,maskPaths,transforms):
        self.imagePaths = imagePaths
        self.maskPaths = maskPaths
        self.transforms = transforms

    def __len__(self):
        return len(self.imagePaths)
    def __getitem__(self,idx):
        imagePath = self.imagePaths[idx]

        image = cv2.imread(imagePath)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        mask = cv2.imread(self.maskPaths[idx],0)

        if self.transforms is not None:
            image = self.transforms(image)
            mask = self.transforms(mask)

        return (image,mask)
```
In [33]:
```python
from torch.nn import ConvTranspose2d
from torch.nn import Conv2d
from torch.nn import MaxPool2d
from torch.nn import Module
from torch.nn import ModuleList
from torch.nn import ReLU
from torchvision.transforms import CenterCrop
from torch.nn import functional as F
```
In [58]:
```python
import cv2
```
In [117…
```python
class Block(Module):
    def __init__(self,inChannels,outChannels):
        super().__init__()
        self.conv1 = Conv2d(inChannels,outChannels,3)
        self.relu = ReLU()
        self.conv2 = Conv2d(outChannels,outChannels,3)

    def forward(self,x):
        return self.conv2(self.relu(self.conv1(x)))
```
In [118…
```python
class Encoder(Module):
    def __init__(self,channels = (3,16,32,64)):
        super().__init__()
        self.encBlocks = ModuleList([Block(
        channels[i],channels[i+1]) for i in range(len(channels)-1)])
        self.pool = MaxPool2d(2)

    def forward(self,x):
        blockOutputs = []
        for block in self.encBlocks:
            x = block(x)
            blockOutputs.append(x)
            x = self.pool(x)
        return blockOutputs
```
In [119…
```python
class Decoder(Module):
        def __init__(self, channels=(64, 32, 16)):
                super().__init__()
                # initialize the number of channels, upsampler blocks, and
                # decoder blocks
                self.channels = channels
                self.upconvs = ModuleList([ConvTranspose2d(channels[i], channels[i + 1], 2, 2) for i in range(len(channels) - 1)])
                self.dec_blocks = ModuleList([Block(channels[i], channels[i + 1]) for i in range(len(channels) - 1)])

        def forward(self, x, encFeatures):
                # loop through the number of channels
                for i in range(len(self.channels) - 1):
                        # pass the inputs through the upsampler blocks
                        x = self.upconvs[i](x)
                        # crop the current features from the encoder blocks,
                        # concatenate them with the current upsampled features,
                        # and pass the concatenated output through the current
                        # decoder block
                        encFeat = self.crop(encFeatures[i], x)
                        x = torch.cat([x, encFeat], dim=1)
                        x = self.dec_blocks[i](x)
                # return the final decoder output
                return x

        def crop(self, encFeatures, x):
                # grab the dimensions of the inputs, and crop the encoder
                # features to match the dimensions
                (_, _, H, W) = x.shape
                encFeatures = CenterCrop([H, W])(encFeatures)
                # return the cropped features
                return encFeatures
```
In [90]:
```python
class UNet(Module):
    def __init__(self,encChannels = (3,16,32,64),
    decChannels = (64,32,16),
    nbClasses =1,retainDim = True,
    outSize = (INPUT_IMAGE_HEIGHT,INPUT_IMAGE_WIDTH)):
        super().__init__()
        self.encoder = Encoder(encChannels)
        self.decoder = Decoder(decChannels)

        self.head = Conv2d(decChannels[-1],nbClasses,1)
        self.retainDim = retainDim
        self.outSize = outSize

    def forward(self,x):
        encFeatures = self.encoder(x)
        decFeatures = self.decoder(encFeatures[::-1][0],
                                   encFeatures[::-1][1:])
        map = self.head(decFeatures)

        if self.retainDim:
            map = F.interpolate(map,self.outSize)

        return map
```
In [84]:
```python
from torch.nn import BCEWithLogitsLoss
from torch.optim import Adam
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
from torchvision import transforms
from imutils import paths
from tqdm import tqdm
import matplotlib.pyplot as plt
import torch
import time
import os
from torch import nn
```
In [120…
```python
imagePaths = sorted(list(paths.list_images(image_path)))
maskPaths = sorted(list(paths.list_images(masks_path)))

split = train_test_split(imagePaths,maskPaths,test_size = 0.25,random_state =42)

(trainImages,testImages) = split[:2]
(trainMasks,testMasks) = split[2:]

print("[INFO] saving testing image paths")
f = open(test_dir,"w")
f.write("\n".join(testImages))
f.close()
```
```
[INFO] saving testing image paths
```
In [125…
```python
trainDS = SegmentationDataset(imagePaths = trainImages,
                              maskPaths = trainMasks,
                              transforms = data_transform)
testDS = SegmentationDataset(imagePaths = testImages,
                              maskPaths = testMasks,
                              transforms = data_transform)

trainLoader = DataLoader(trainDS, shuffle = True,
                        batch_size = BATCH_SIZE)
testLoader = DataLoader(testDS, shuffle = False, batch_size = BATCH_SIZE)
```
In [126…
```python
len(trainDS[0][0])
```
Out[126]:
```
3
```
In [127…
```python
trainDS[0][0].size()
```
Out[127]:
```
torch.Size([3, 128, 128])
```
trainDS

In [130…
```python
unet = UNet()
lossfn = BCEWithLogitsLoss()
optimizer  = Adam(unet.parameters(),lr = INIT_LR)

trainSteps = len(trainDS) // BATCH_SIZE
testSteps = len(testDS) // BATCH_SIZE

H = {"train_loss" :[], "test_loss" :[]}
```
In [132…
```python
startTime = time.time()
for e in tqdm(range(NUM_EPOCHS)):
    unet.train()
    totalTrainLoss = 0
    totalTestLoss = 0

    for (i,(x,y)) in enumerate(trainLoader):
        pred = unet(x)
        loss = lossfn(pred,y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        totalTrainLoss += loss

    with torch.inference_mode():
        unet.eval()

        for(x,y) in testLoader:
            pred = unet(x)

            totalTestLoss += lossfn(pred,y)

    avgTrainLoss = totalTrainLoss / trainSteps
    avgTestLoss = totalTestLoss / testSteps

    H["train_loss"].append(avgTrainLoss.detach().numpy())
    H["test_loss"].append(avgTestLoss.detach().numpy())

    print("EPOCH: {}/{}".format(e+1,NUM_EPOCHS))
    print("Train loss: {:,.6f},Test loss: {:.4f}".format(avgTrainLoss,avgTestLoss))

endTime = time.time()
print("[INFO] total time taken to train the model : {:.2f}s").format(endTime - startTime)
```
```
  2%|██          | 1/40 [05:30<3:34:56, 330.69s/it]
EPOCH: 1/40
Train loss: 0.671428,Test loss: 0.7142
  5%|████        | 2/40 [10:51<3:25:51, 325.04s/it]
EPOCH: 2/40
Train loss: 0.601234,Test loss: 0.7036
  8%|██████      | 3/40 [16:12<3:19:12, 323.05s/it]
EPOCH: 3/40
Train loss: 0.593643,Test loss: 0.6966
 10%|████████    | 4/40 [21:29<3:12:29, 320.82s/it]
EPOCH: 4/40
Train loss: 0.583479,Test loss: 0.6827
 12%|██████████  | 5/40 [26:50<3:07:12, 320.93s/it]
EPOCH: 5/40
Train loss: 0.573948,Test loss: 0.6658
 15%|████████████ | 6/40 [32:08<3:01:16, 319.89s/it]
EPOCH: 6/40
Train loss: 0.558571,Test loss: 0.6439
 18%|██████████████ | 7/40 [37:24<2:55:10, 318.50s/it]
EPOCH: 7/40
Train loss: 0.545687,Test loss: 0.6351
 20%|████████████████ | 8/40 [42:40<2:49:30, 317.84s/it]
EPOCH: 8/40
Train loss: 0.541780,Test loss: 0.6306
 22%|█████████████████ | 9/40 [47:58<2:44:13, 317.84s/it]
EPOCH: 9/40
Train loss: 0.537232,Test loss: 0.6201
 25%|████████████████████ | 10/40 [53:16<2:38:55, 317.86s/it]
EPOCH: 10/40
Train loss: 0.521046,Test loss: 0.5967
 28%|██████████████████████ | 11/40 [58:33<2:33:32, 317.69s/it]
EPOCH: 11/40
Train loss: 0.507345,Test loss: 0.5665
 30%|████████████████████████ | 12/40 [1:03:52<2:28:18, 317.80s/it]
EPOCH: 12/40
Train loss: 0.477369,Test loss: 0.5348
 32%|█████████████████████████ | 13/40 [1:09:09<2:23:00, 317.81s/it]
EPOCH: 13/40
Train loss: 0.453204,Test loss: 0.4997
 35%|████████████████████████████ | 14/40 [1:14:26<2:17:35, 317.54s/it]
EPOCH: 14/40
Train loss: 0.437583,Test loss: 0.5305
 38%|██████████████████████████████ | 15/40 [1:19:43<2:12:13, 317.35s/it]
EPOCH: 15/40
Train loss: 0.451613,Test loss: 0.5149
 40%|████████████████████████████████ | 16/40 [1:25:02<2:07:09, 317.88s/it]
EPOCH: 16/40
Train loss: 0.423355,Test loss: 0.4740
 42%|█████████████████████████████████ | 17/40 [1:30:22<2:02:01, 318.33s/it]
EPOCH: 17/40
Train loss: 0.403994,Test loss: 0.4695
 45%|████████████████████████████████████ | 18/40 [1:35:38<1:56:31, 317.81s/it]
EPOCH: 18/40
Train loss: 0.399354,Test loss: 0.4504
 48%|██████████████████████████████████████ | 19/40 [1:40:55<1:51:05, 317.40s/it]
EPOCH: 19/40
Train loss: 0.391811,Test loss: 0.4307
 50%|████████████████████████████████████████ | 20/40 [1:46:11<1:45:44, 317.20s/it]
EPOCH: 20/40
Train loss: 0.380109,Test loss: 0.4199
 52%|█████████████████████████████████████████ | 21/40 [1:51:28<1:40:21, 316.92s/it]
EPOCH: 21/40
Train loss: 0.375834,Test loss: 0.4169
 55%|███████████████████████████████████████████ | 22/40 [1:56:44<1:34:59, 316.65s/it]
EPOCH: 22/40
Train loss: 0.370778,Test loss: 0.4156
 57%|█████████████████████████████████████████████ | 23/40 [2:01:59<1:29:38, 316.39s/it]
EPOCH: 23/40
Train loss: 0.369232,Test loss: 0.4171
 60%|███████████████████████████████████████████████ | 24/40 [2:07:15<1:24:16, 316.06s/it]
EPOCH: 24/40
Train loss: 0.364043,Test loss: 0.3961
 62%|█████████████████████████████████████████████████ | 25/40 [2:12:29<1:18:53, 315.59s/it]
EPOCH: 25/40
Train loss: 0.356018,Test loss: 0.3929
 65%|███████████████████████████████████████████████████ | 26/40 [2:17:43<1:13:30, 315.04s/it]
EPOCH: 26/40
Train loss: 0.352217,Test loss: 0.3843
 68%|██████████████████████████████████████████████████████ | 27/40 [2:22:59<1:08:19, 315.36s/it]
EPOCH: 27/40
Train loss: 0.348109,Test loss: 0.3929
 70%|███████████████████████████████████████████████████████ | 28/40 [2:28:16<1:03:11, 315.95s/it]
EPOCH: 28/40
Train loss: 0.353119,Test loss: 0.3881
 72%|█████████████████████████████████████████████████████████ | 29/40 [2:33:34<57:59, 316.28s/it]
EPOCH: 29/40
Train loss: 0.351022,Test loss: 0.3829
 75%|███████████████████████████████████████████████████████████ | 30/40 [2:38:52<52:50, 317.08s/it]
EPOCH: 30/40
Train loss: 0.345168,Test loss: 0.3864
 78%|██████████████████████████████████████████████████████████████ | 31/40 [2:44:09<47:31, 316.84s/it]
EPOCH: 31/40
Train loss: 0.343797,Test loss: 0.3794
 80%|███████████████████████████████████████████████████████████████ | 32/40 [2:49:26<42:16, 317.00s/it]
EPOCH: 32/40
Train loss: 0.343222,Test loss: 0.3775
 82%|█████████████████████████████████████████████████████████████████ | 33/40 [2:54:44<37:00, 317.23s/it]
EPOCH: 33/40
Train loss: 0.338577,Test loss: 0.3712
 85%|███████████████████████████████████████████████████████████████████ | 34/40 [3:00:02<31:45, 317.60s/it]
EPOCH: 34/40
Train loss: 0.339075,Test loss: 0.3691
 88%|██████████████████████████████████████████████████████████████████████ | 35/40 [3:05:22<26:31, 318.30s/it]
EPOCH: 35/40
Train loss: 0.334711,Test loss: 0.3680
 90%|███████████████████████████████████████████████████████████████████████ | 36/40 [3:10:44<21:16, 319.21s/it]
EPOCH: 36/40
Train loss: 0.332943,Test loss: 0.3726
 92%|█████████████████████████████████████████████████████████████████████████ | 37/40 [3:15:56<15:51, 317.20s/it]
EPOCH: 37/40
Train loss: 0.329948,Test loss: 0.3661
 95%|███████████████████████████████████████████████████████████████████████████ | 38/40 [3:21:12<10:33, 316.68s/it]
EPOCH: 38/40
Train loss: 0.332785,Test loss: 0.3686
 98%|█████████████████████████████████████████████████████████████████████████████ | 39/40 [3:26:29<05:16, 316.75s/it]
EPOCH: 39/40
Train loss: 0.336052,Test loss: 0.3757
100%|███████████████████████████████████████████████████████████████████████████████| 40/40 [3:31:44<00:00, 317.62s/it]
EPOCH: 40/40
Train loss: 0.336181,Test loss: 0.3589
[INFO] total time taken to train the model : {:.2f}s
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[132], line 35
     32     print("Train loss: {:,.6f},Test loss: {:.4f}".format(avgTrainLoss,avgTestLoss))
     34 endTime = time.time()
---> 35 print("[INFO] total time taken to train the model : {:.2f}s").format(endTime - startTime)

AttributeError: 'NoneType' object has no attribute 'format'
```
In [135…
```python
(endTime - startTime)/60 #minutes taken to train the model
```
Out[135]:
```
211.74997220039367
```
In [136…
```python
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"],label="train_loss")
plt.plot(H["test_loss"],label = "test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
```
Out[136]:
```
<matplotlib.legend.Legend at 0x2395c70c610>
```


In [138…
```python
torch.save(unet,MODEL_PATH)
```
Making Predictions

In [147…
```python
import numpy as np
```
In [156…
```python
def prepare_plot(origImage,predMask):
    figure, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (10,10))

    ax[0].imshow(origImage)
    #ax[1].imshow(origMask)
    ax[1].imshow(predMask,cmap='gray')

    ax[0].set_title("Image")
    #ax[1].set_title("Original Mask")
    ax[1].set_title("Predicted Mask")

    figure.tight_layout()
    figure.show()
```
In [157…
```python
unet.eval()

with torch.inference_mode():
    image = cv2.imread("data/X_ray_image_and_mask_for_U_net_training/CXR_png/CHNCXR_0035_0.png")
    image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    image = image.astype("float32")/255.0
    image = cv2.resize(image,(128,128))

    orig = image.copy()

       # filename = imagePath.split(os.path.sep)[-1]
        #groundTruthPath = os.path.join(masks_path,filename)

        #gtMask = cv2.imread(groundTruthPath,0)
        #gtMask = cv2.resize(gtMask,(INPUT_IMAGE_HEIGHT,INPUT_IMAGE_WIDTH))

    image = np.transpose(image,(2,0,1)) #to get it to [C,H,W] format
    image = np.expand_dims(image,0) #model inputs 4 dimensional tensor
    image = torch.tensor(image)

    predMask = unet(image).squeeze() #to remove extra dimension
    predMask = torch.sigmoid(predMask)
    predMask = predMask.numpy()

    predMask = (predMask>THRESHOLD) *255
    predMask = predMask.astype(np.uint8)

    prepare_plot(orig,predMask)
```
```
C:\Users\aayus\AppData\Local\Temp\ipykernel_42968\486299230.py:13: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  figure.show()
```


In [ ]:

In [1]:
```python
# Importing necessary libraries
import os
import random
import shutil
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16, VGG19, ResNet50, InceptionV3, Xception
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix

# Setting random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Define constants
NUM_CLASSES = 2
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
TRAIN_DIR = 'train'
VAL_DIR = 'val'
TEST_DIR = 'test'
TRAIN_SPLIT = 0.7
VAL_SPLIT = 0.1
TEST_SPLIT = 0.2
test_samples = 280
```

In [2]:
```python
# Define the base directory where the image data is located
BASE_DIR = r"C:\Users\mohit\Downloads\DIP Assignment 2\Data"
```

In [3]:
```python
# Set the paths for training, validation, and test data
TRAIN_DIR = os.path.join(BASE_DIR, 'train')
VAL_DIR = os.path.join(BASE_DIR, 'val')
TEST_DIR = os.path.join(BASE_DIR, 'test')
```

In [4]:
```python
# Load and preprocess the data
train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [5]:
```python
train_generator = train_datagen.flow_from_directory(TRAIN_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

val_generator = val_datagen.flow_from_directory(VAL_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

test_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical',shuffle=False)
```

```
Found 980 images belonging to 2 classes.
Found 140 images belonging to 2 classes.
Found 280 images belonging to 2 classes.
```

In [6]:
```python
# Define function for creating transfer learning models
def create_transfer_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Define VGG16 model
vgg16_base = VGG16(include_top=False, weights='imagenet', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
vgg16_model = create_transfer_model(vgg16_base, NUM_CLASSES)

# Define VGG19 model
vgg19_base = VGG19(include_top=False, weights='imagenet', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
vgg19_model = create_transfer_model(vgg19_base, NUM_CLASSES)
```

In [7]:
```python
# Compile the models
optimizer = Adam(learning_rate=0.0001)
vgg16_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
vgg19_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [8]:
```python
# Define checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
```

In [9]:
```python
# Train the models
vgg16_history = vgg16_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validation_steps
vgg19_history = vgg19_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validation_steps
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\2039408895.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  vgg16_history = vgg16_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validation_steps=val_generator.n // val_generator.batch_size, callbacks=[checkpoint])
Epoch 1/5
30/30 [==============================] - 269s 9s/step - loss: 0.6998 - accuracy: 0.5190 - val_loss: 0.7125 - val_accuracy: 0.5000
Epoch 2/5
30/30 [==============================] - 286s 10s/step - loss: 0.6402 - accuracy: 0.6118 - val_loss: 0.4709 - val_accuracy: 0.8281
Epoch 3/5
30/30 [==============================] - 470s 16s/step - loss: 0.2767 - accuracy: 0.9008 - val_loss: 0.1435 - val_accuracy: 0.9531
Epoch 4/5
30/30 [==============================] - 503s 17s/step - loss: 0.1493 - accuracy: 0.9473 - val_loss: 0.1466 - val_accuracy: 0.9609
Epoch 5/5
30/30 [==============================] - 504s 17s/step - loss: 0.1644 - accuracy: 0.9494 - val_loss: 0.0989 - val_accuracy: 0.9688
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\2039408895.py:3: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  vgg19_history = vgg19_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validation_steps=val_generator.n // val_generator.batch_size, callbacks=[checkpoint])
Epoch 1/5
30/30 [==============================] - 570s 19s/step - loss: 0.7282 - accuracy: 0.5042 - val_loss: 0.6897 - val_accuracy: 0.5156
Epoch 2/5
30/30 [==============================] - 558s 19s/step - loss: 0.6453 - accuracy: 0.6983 - val_loss: 0.4191 - val_accuracy: 0.8672
Epoch 3/5
30/30 [==============================] - 544s 18s/step - loss: 0.3698 - accuracy: 0.8576 - val_loss: 0.1742 - val_accuracy: 0.9297
Epoch 4/5
30/30 [==============================] - 547s 18s/step - loss: 0.2672 - accuracy: 0.9040 - val_loss: 0.1718 - val_accuracy: 0.9375
Epoch 5/5
30/30 [==============================] - 354s 12s/step - loss: 0.2497 - accuracy: 0.9146 - val_loss: 0.2170 - val_accuracy: 0.8984
```

In [10]:
```python
# Evaluate the models on test data
vgg16_scores = vgg16_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("VGG16 Test Loss:", vgg16_scores[0])
print("VGG16 Test Accuracy:", vgg16_scores[1])

vgg19_scores = vgg19_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("VGG19 Test Loss:", vgg19_scores[0])
print("VGG19 Test Accuracy:", vgg19_scores[1])
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\1392354301.py:2: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  vgg16_scores = vgg16_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
VGG16 Test Loss: 0.07796119153499603
VGG16 Test Accuracy: 0.97265625
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\1392354301.py:6: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  vgg19_scores = vgg19_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
VGG19 Test Loss: 0.14393511414527893
VGG19 Test Accuracy: 0.94140625
```

In [15]:
```python
# Get the number of test samples
num_test_samples = test_generator.n

# Calculate the number of steps for prediction
num_prediction_steps = num_test_samples // test_generator.batch_size + 1

# Generate predictions for all test samples
vgg16_predictions = vgg16_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)

# Convert predictions to class labels
vgg16_predicted_labels = np.argmax(vgg16_predictions, axis=1)
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\2218791254.py:8: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  vgg16_predictions = vgg16_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)
9/9 [==============================] - 23s 3s/step
```

In [16]:
```python
# Get the number of test samples
num_test_samples = test_generator.n

# Calculate the number of steps for prediction
num_prediction_steps = num_test_samples // test_generator.batch_size + 1

# Generate predictions for all test samples
vgg19_predictions = vgg19_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)

# Convert predictions to class labels
vgg19_predicted_labels = np.argmax(vgg19_predictions, axis=1)
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_25572\903632277.py:8: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  vgg19_predictions = vgg19_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)
9/9 [==============================] - 30s 3s/step
```

In [17]:
```python
# Get true class labels
true_labels = test_generator.classes
```

In [18]:
```python
# Calculate classification report
vgg16_report = classification_report(true_labels, vgg16_predicted_labels)
```

In [19]:
```python
# Calculate classification report
from sklearn.metrics import classification_report

# Get the ground truth labels
ground_truth_labels = test_generator.classes

# Get the predicted labels
vgg19_predicted_labels = np.argmax(vgg19_predictions, axis=1)

# Calculate classification report
classification_report_vgg19 = classification_report(ground_truth_labels, vgg19_predicted_labels, zero_division=1)
```

In [20]:
```python
print("vgg16 Classification Report:")
print(vgg16_report)

print("vgg19 Classification Report:")
print(classification_report_vgg19)
```

```
vgg16 Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       140
           1       0.99      0.94      0.96       140

    accuracy                           0.96       280
   macro avg       0.97      0.96      0.96       280
weighted avg       0.97      0.96      0.96       280

vgg19 Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.99      0.93       140
           1       0.98      0.86      0.92       140

    accuracy                           0.92       280
   macro avg       0.93      0.92      0.92       280
weighted avg       0.93      0.92      0.92       280
```

In [ ]:

In [5]:

```python
# Importing necessary libraries
import os
import random
import shutil
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16, VGG19, ResNet50, InceptionV3, Xception
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix

# Setting random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Define constants
NUM_CLASSES = 2
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
TRAIN_DIR = 'train'
VAL_DIR = 'val'
TEST_DIR = 'test'
TRAIN_SPLIT = 0.7
VAL_SPLIT = 0.1
TEST_SPLIT = 0.2
test_samples = 280

# Define the base directory where the image data is located
BASE_DIR = r"C:\Users\mohit\Downloads\DIP Assignment 2\Data"

# Set the paths for training, validation, and test data
TRAIN_DIR = os.path.join(BASE_DIR, 'train')
VAL_DIR = os.path.join(BASE_DIR, 'val')
TEST_DIR = os.path.join(BASE_DIR, 'test')

# Load and preprocess the data
train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(TRAIN_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

val_generator = val_datagen.flow_from_directory(VAL_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

test_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical',shuffle=False)

# Define function for creating transfer learning models
def create_transfer_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Define Xception model
xception_base = Xception(include_top=False, weights='imagenet', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
xception_model = create_transfer_model(xception_base, NUM_CLASSES)

# Compile the models
optimizer = Adam(learning_rate=0.0001)
xception_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Define checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

xception_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

xception_history = xception_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validatio

xception_scores = xception_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("Xception Test Loss:", xception_scores[0])
print("Xception Test Accuracy:", xception_scores[1])

# Get the number of test samples
num_test_samples = test_generator.n

# Calculate the number of steps for prediction
num_prediction_steps = num_test_samples // test_generator.batch_size + 1

# Generate predictions for all test samples
xception_predictions = xception_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)

# Convert predictions to class labels
xception_predicted_labels = np.argmax(xception_predictions, axis=1)

# Get true class labels
true_labels = test_generator.classes

# Calculate classification report
xception_report = classification_report(true_labels, xception_predicted_labels)

print("Xception Classification Report:")
print(xception_report)
```

```
Found 980 images belonging to 2 classes.
Found 140 images belonging to 2 classes.
Found 280 images belonging to 2 classes.
C:\Users\mohit\AppData\Local\Temp\ipykernel_33056\3483214989.py:72: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit
`, which supports generators.
  xception_history = xception_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=5, validation_data=val_generator, validat
ion_steps=val_generator.n // val_generator.batch_size, callbacks=[checkpoint])
Epoch 1/5
30/30 [==============================] - 186s 6s/step - loss: 0.1842 - accuracy: 0.9462 - val_loss: 0.1311 - val_accuracy: 0.9219
Epoch 2/5
30/30 [==============================] - 193s 6s/step - loss: 0.0225 - accuracy: 0.9926 - val_loss: 0.1422 - val_accuracy: 0.9297
Epoch 3/5
30/30 [==============================] - 190s 6s/step - loss: 0.0069 - accuracy: 0.9989 - val_loss: 0.0426 - val_accuracy: 0.9922
Epoch 4/5
30/30 [==============================] - 192s 6s/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0151 - val_accuracy: 0.9922
Epoch 5/5
30/30 [==============================] - 192s 6s/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0074 - val_accuracy: 0.9922
C:\Users\mohit\AppData\Local\Temp\ipykernel_33056\3483214989.py:74: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Mode
l.evaluate`, which supports generators.
  xception_scores = xception_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
Xception Test Loss: 0.012623521499335766
Xception Test Accuracy: 0.9921875
C:\Users\mohit\AppData\Local\Temp\ipykernel_33056\3483214989.py:85: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.
predict`, which supports generators.
  xception_predictions = xception_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)
9/9 [==============================] - 11s 1s/step
Xception Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       140
           1       1.00      0.99      0.99       140

    accuracy                           0.99       280
   macro avg       0.99      0.99      0.99       280
weighted avg       0.99      0.99      0.99       280
```

In [ ]:

In [1]:
```
# Importing necessary libraries
import os
import random
import shutil
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16, VGG19, ResNet50, InceptionV3, Xception
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix

# Setting random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Define constants
NUM_CLASSES = 2
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
TRAIN_DIR = 'train'
VAL_DIR = 'val'
TEST_DIR = 'test'
TRAIN_SPLIT = 0.7
VAL_SPLIT = 0.1
TEST_SPLIT = 0.2
test_samples = 280
```

In [2]:
```
# Define the base directory where the image data is located
BASE_DIR = r"C:\Users\mohit\Downloads\DIP Assignment 2\Data"
```

In [3]:
```
# Set the paths for training, validation, and test data
TRAIN_DIR = os.path.join(BASE_DIR, 'train')
VAL_DIR = os.path.join(BASE_DIR, 'val')
TEST_DIR = os.path.join(BASE_DIR, 'test')
```

In [4]:
```
# Load and preprocess the data
train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [5]:
```
train_generator = train_datagen.flow_from_directory(TRAIN_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

val_generator = val_datagen.flow_from_directory(VAL_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

test_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical',shuffle=False)
```

```
Found 980 images belonging to 2 classes.
Found 140 images belonging to 2 classes.
Found 280 images belonging to 2 classes.
```

In [6]:
```
# Define function for creating transfer learning models
def create_transfer_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

In [7]:
```
# Define InceptionV3 model
inceptionv3_base = InceptionV3(include_top=False, weights='imagenet', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
inceptionv3_model = create_transfer_model(inceptionv3_base, NUM_CLASSES)
```

In [8]:
```
# Compile the models
optimizer = Adam(learning_rate=0.0001)
inceptionv3_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [9]:
```
# Define checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
```

In [10]:
```
# Train the models
inceptionv3_history = inceptionv3_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=10, validation_data=val_generator, va
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_34676\918017655.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  inceptionv3_history = inceptionv3_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=10, validation_data=val_generator, validation_steps=val_generator.n // val_generator.batch_size, callbacks=[checkpoint])
Epoch 1/10
30/30 [==============================] - 110s 4s/step - loss: 0.1248 - accuracy: 0.9420 - val_loss: 0.0394 - val_accuracy: 0.9922
Epoch 2/10
30/30 [==============================] - 112s 4s/step - loss: 0.0094 - accuracy: 0.9979 - val_loss: 0.0310 - val_accuracy: 0.9922
Epoch 3/10
30/30 [==============================] - 112s 4s/step - loss: 0.0092 - accuracy: 0.9968 - val_loss: 0.0544 - val_accuracy: 0.9922
Epoch 4/10
30/30 [==============================] - 112s 4s/step - loss: 0.0117 - accuracy: 0.9979 - val_loss: 6.0826e-04 - val_accuracy: 1.0000
Epoch 5/10
30/30 [==============================] - 111s 4s/step - loss: 0.0068 - accuracy: 0.9979 - val_loss: 0.0132 - val_accuracy: 0.9922
Epoch 6/10
30/30 [==============================] - 111s 4s/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0223 - val_accuracy: 0.9844
Epoch 7/10
30/30 [==============================] - 113s 4s/step - loss: 0.0056 - accuracy: 0.9968 - val_loss: 7.1364e-04 - val_accuracy: 1.0000
Epoch 8/10
30/30 [==============================] - 113s 4s/step - loss: 0.0047 - accuracy: 0.9968 - val_loss: 0.0220 - val_accuracy: 0.9844
Epoch 9/10
30/30 [==============================] - 112s 4s/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0742 - val_accuracy: 0.9844
Epoch 10/10
30/30 [==============================] - 111s 4s/step - loss: 0.0023 - accuracy: 0.9989 - val_loss: 0.0065 - val_accuracy: 1.0000
```

In [11]:
```
inceptionv3_scores = inceptionv3_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("inceptionv3 Test Loss:", inceptionv3_scores[0])
print("inceptionv3 Test Accuracy:", inceptionv3_scores[1])
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_34676\1195483549.py:1: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  inceptionv3_scores = inceptionv3_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
inceptionv3 Test Loss: 0.0008152585942298174
inceptionv3 Test Accuracy: 1.0
```

In [12]:
```
# Get the number of test samples
num_test_samples = test_generator.n

# Calculate the number of steps for prediction
num_prediction_steps = num_test_samples // test_generator.batch_size + 1

# Generate predictions for all test samples
inceptionv3_predictions = inceptionv3_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)

# Convert predictions to class labels
inceptionv3_predicted_labels = np.argmax(inceptionv3_predictions, axis=1)
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_34676\2714831293.py:8: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  inceptionv3_predictions = inceptionv3_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)
9/9 [==============================] - 6s 613ms/step
```

In [13]:
```
# Get true class labels
true_labels = test_generator.classes
```

In [14]:
```
# Calculate classification report
inceptionv3_report = classification_report(true_labels, inceptionv3_predicted_labels)
```

In [15]:
```
print("inception Classification Report:")
print(inceptionv3_report)
```

```
inception Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       140
           1       1.00      1.00      1.00       140

    accuracy                           1.00       280
   macro avg       1.00      1.00      1.00       280
weighted avg       1.00      1.00      1.00       280
```

In [16]:
```
inceptionv3_scores = inceptionv3_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("inceptionv3 Test Loss:", inceptionv3_scores[0])
print("inceptionv3 Test Accuracy:", inceptionv3_scores[1])
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_34676\1195483549.py:1: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  inceptionv3_scores = inceptionv3_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
inceptionv3 Test Loss: 0.0008152585942298174
inceptionv3 Test Accuracy: 1.0
```

In [ ]:
```

```

In [1]:

```
# Importing necessary libraries
import os
import random
import shutil
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16, VGG19, ResNet50, InceptionV3, Xception
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix

# Setting random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Define constants
NUM_CLASSES = 2
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
TRAIN_DIR = 'train'
VAL_DIR = 'val'
TEST_DIR = 'test'
TRAIN_SPLIT = 0.7
VAL_SPLIT = 0.1
TEST_SPLIT = 0.2
test_samples = 280
```

In [2]:

```
# Define the base directory where the image data is located
BASE_DIR = r"C:\Users\mohit\Downloads\DIP Assignment 2\Data"
```

In [3]:

```
# Set the paths for training, validation, and test data
TRAIN_DIR = os.path.join(BASE_DIR, 'train')
VAL_DIR = os.path.join(BASE_DIR, 'val')
TEST_DIR = os.path.join(BASE_DIR, 'test')
```

In [4]:

```
# Load and preprocess the data
train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [5]:

```
train_generator = train_datagen.flow_from_directory(TRAIN_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

val_generator = val_datagen.flow_from_directory(VAL_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical')

test_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=IMG_SIZE,batch_size=BATCH_SIZE,class_mode='categorical',shuffle=False)
```

```
Found 980 images belonging to 2 classes.
Found 140 images belonging to 2 classes.
Found 280 images belonging to 2 classes.
```

In [6]:

```
# Define function for creating transfer learning models
def create_transfer_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Define ResNet50 model
resnet50_base = ResNet50(include_top=False, weights='imagenet', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
resnet50_model = create_transfer_model(resnet50_base, NUM_CLASSES)
```

In [7]:

```
# Compile the models
optimizer = Adam(learning_rate=0.0001)
resnet50_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [8]:

```
# Define checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
```

In [9]:

```
# Train the models
resnet50_history = resnet50_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=10, validation_data=val_generator, validati
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_33032\951620722.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  resnet50_history = resnet50_model.fit_generator(train_generator, steps_per_epoch=train_generator.n // train_generator.batch_size, epochs=10, validation_data=val_generator, validation_steps=val_generator.n // val_generator.batch_size, callbacks=[checkpoint])
Epoch 1/10
30/30 [==============================] - 358s 11s/step - loss: 0.0822 - accuracy: 0.9684 - val_loss: 3.9794 - val_accuracy: 0.5000
Epoch 2/10
30/30 [==============================] - 324s 11s/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 1.4914 - val_accuracy: 0.4922
Epoch 3/10
30/30 [==============================] - 326s 11s/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 7.1890 - val_accuracy: 0.4844
Epoch 4/10
30/30 [==============================] - 325s 11s/step - loss: 0.0068 - accuracy: 0.9979 - val_loss: 34.1761 - val_accuracy: 0.5000
Epoch 5/10
30/30 [==============================] - 358s 12s/step - loss: 0.0027 - accuracy: 0.9989 - val_loss: 14.5630 - val_accuracy: 0.5078
Epoch 6/10
30/30 [==============================] - 366s 12s/step - loss: 6.6791e-04 - accuracy: 1.0000 - val_loss: 2.8473 - val_accuracy: 0.5156
30/30 [==============================] - 391s 13s/step - loss: 2.8256e-04 - accuracy: 1.0000 - val_loss: 1.7559 - val_accuracy: 0.5000
Epoch 7/10
30/30 [==============================] - 395s 13s/step - loss: 0.0084 - accuracy: 0.9958 - val_loss: 13.6165 - val_accuracy: 0.4922
Epoch 8/10
30/30 [==============================] - 402s 13s/step - loss: 0.0079 - accuracy: 0.9968 - val_loss: 14.6081 - val_accuracy: 0.4844
Epoch 9/10
Epoch 10/10
30/30 [==============================] - 357s 12s/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 20.5592 - val_accuracy: 0.5000
```

In [10]:

```
# Evaluate the models on test data
resnet50_scores = resnet50_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
print("ResNet50 Test Loss:", resnet50_scores[0])
print("ResNet50 Test Accuracy:", resnet50_scores[1])
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_33032\556035509.py:2: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  resnet50_scores = resnet50_model.evaluate_generator(test_generator, steps=test_samples // BATCH_SIZE)
ResNet50 Test Loss: 18.02694320678711
ResNet50 Test Accuracy: 0.546875
```

In [11]:

```
# Get the number of test samples
num_test_samples = test_generator.n

# Calculate the number of steps for prediction
num_prediction_steps = num_test_samples // test_generator.batch_size + 1

# Generate predictions for all test samples
resnet50_predictions = resnet50_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)

# Convert predictions to class labels
resnet50_predicted_labels = np.argmax(resnet50_predictions, axis=1)
```

```
C:\Users\mohit\AppData\Local\Temp\ipykernel_33032\2677680890.py:8: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  resnet50_predictions = resnet50_model.predict_generator(test_generator, steps=num_prediction_steps, verbose=1)
9/9 [==============================] - 25s 3s/step
```

In [12]:

```
# Get true class labels
true_labels = test_generator.classes
```

In [13]:

```
# Calculate classification report
from sklearn.metrics import classification_report

# Get the ground truth labels
ground_truth_labels = test_generator.classes

# Get the predicted labels
resnet50_predicted_labels = np.argmax(resnet50_predictions, axis=1)

# Calculate classification report
classification_report_resnet50 = classification_report(ground_truth_labels, resnet50_predicted_labels, zero_division=1)
```

In [15]:

```
print("ResNet50 Classification Report:")
print(classification_report_resnet50)
```

```
ResNet50 Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67       140
           1       1.00      0.00      0.00       140

    accuracy                           0.50       280
   macro avg       0.75      0.50      0.33       280
weighted avg       0.75      0.50      0.33       280
```

In [ ]: