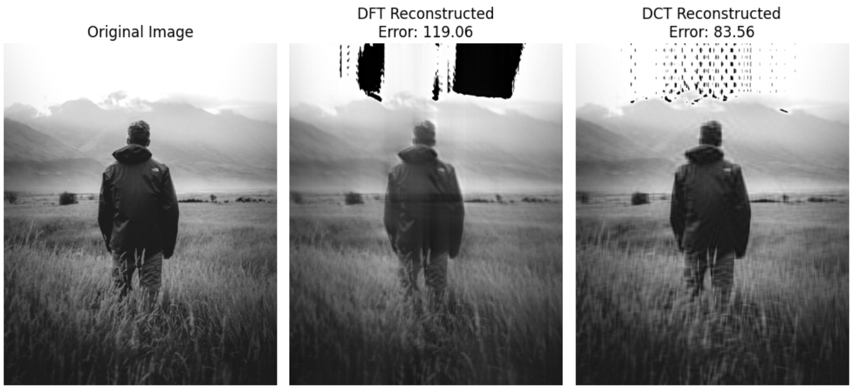
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('photo1.jpeg')

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image using matplotlib
plt.imshow(gray_image, cmap='gray')
plt.axis('off')  # Turn off axis labels
plt.show()
```

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('photo1.jpeg')

# Define the size of the blocks (sub-images)
block_size = 64  # Adjust this size according to your needs

# Get the dimensions of the image
height, width, _ = image.shape

# Initialize a list to store the sub-images
sub_images = []

# Iterate over the image in blocks
for y in range(0, height, block_size):
    for x in range(0, width, block_size):
        # Extract the sub-image (block)
        sub_image = image[y:y+block_size, x:x+block_size]
        sub_images.append(sub_image)

# Display or save the sub-images
for i, sub_image in enumerate(sub_images):
    cv2.imwrite(f'sub_image_{i}.jpeg', sub_image)  # Save each sub-image with a unique name
    # To display sub-images, you can use cv2.imshow here.

# Note: This code saves each sub-image as separate files. You can adjust it as needed.
```

```
import cv2
import numpy as np
from scipy.fftpack import dct

# Load the image and divide it into blocks (as shown in previous answers)
image = cv2.imread('photo1.jpeg')
block_size = 8
height, width, _ = image.shape

sub_images = []
for y in range(0, height, block_size):
    for x in range(0, width, block_size):
        sub_image = image[y:y+block_size, x:x+block_size]
        sub_images.append(sub_image)

# Initialize a list to store DCT-transformed blocks
dct_transformed_blocks = []

# Perform DCT on each sub-image (block)
for sub_image in sub_images:
    # Convert the sub-image to grayscale
    gray_sub_image = cv2.cvtColor(sub_image, cv2.COLOR_BGR2GRAY)

    # Apply DCT
    dct_block = dct(dct(gray_sub_image, axis=0, norm='ortho'), axis=1, norm='ortho')

    dct_transformed_blocks.append(dct_block)



# Define the quantization step size (adjust as needed)
quantization_step = 10

# Initialize a list to store quantized blocks
quantized_blocks = []

# Perform quantization on each DCT-transformed block
for dct_block in dct_transformed_blocks:
    quantized_block = np.round(dct_block / quantization_step)
    quantized_blocks.append(quantized_block)


import cv2
import numpy as np
from scipy.fftpack import idct
import matplotlib.pyplot as plt

# Load the image and divide it into blocks (as shown in previous answers)
image = cv2.imread('photo1.jpeg')
block_size = 8
height, width, _ = image.shape

sub_images = []
for y in range(0, height, block_size):
    for x in range(0, width, block_size):
        sub_image = image[y:y+block_size, x:x+block_size]
        sub_images.append(sub_image)

# Initialize a list to store the reconstructed blocks
reconstructed_blocks = []

# Perform inverse quantization and inverse DCT on each quantized block
for quantized_block in quantized_blocks:
    # Inverse quantization
    reconstructed_dct_block = quantized_block * quantization_step

    # Inverse DCT
    reconstructed_gray_sub_image = idct(idct(reconstructed_dct_block, axis=0, norm='ortho'), axis=1, norm='ortho')

    # Create a 3-channel image (for color images)
    reconstructed_sub_image = cv2.cvtColor(reconstructed_gray_sub_image.astype(np.uint8), cv2.COLOR_GRAY2BGR)

    reconstructed_blocks.append(reconstructed_sub_image)

# Reconstruct the image by arranging the blocks
reconstructed_image = np.zeros_like(image)

block_idx = 0
for y in range(0, height, block_size):
    for x in range(0, width, block_size):
        block = reconstructed_blocks[block_idx]
        reconstructed_image[y:y+block_size, x:x+block_size] = block
        block_idx += 1

# Display the reconstructed image using matplotlib
plt.imshow(cv2.cvtColor(reconstructed_image, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Turn off axis labels
plt.show()
```

```
import cv2
import numpy as np
from scipy.fftpack import fft2, ifft2, dct, idct

# Load the image
image = cv2.imread('photo1.jpeg')

# Define the number of coefficients to retain
num_coefficients_to_retain = 50  # Adjust this number as needed

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Perform DFT
dft_image = fft2(gray_image)

# Perform DCT
dct_image = dct(dct(gray_image, axis=0, norm='ortho'), axis=1, norm='ortho')

# Retain only the specified number of coefficients for both transforms
dft_image_reduced = np.copy(dft_image)
dft_image_reduced[num_coefficients_to_retain:, num_coefficients_to_retain:] = 0

dct_image_reduced = np.copy(dct_image)
dct_image_reduced[num_coefficients_to_retain:, num_coefficients_to_retain:] = 0

# Reconstruct the images from the reduced coefficients
reconstructed_image_dft = np.abs(ifft2(dft_image_reduced)).astype(np.uint8)
reconstructed_image_dct = idct(idct(dct_image_reduced, axis=0, norm='ortho'), axis=1, norm='ortho').astype(np.uint8)

# Calculate the error between the original and reconstructed images
error_dft = np.mean(np.abs(gray_image - reconstructed_image_dft))
error_dct = np.mean(np.abs(gray_image - reconstructed_image_dct))

# Display the original, DFT-reconstructed, and DCT-reconstructed images
import matplotlib.pyplot as plt


plt.show()

plt.figure(figsize=(10, 5))
plt.subplot(131)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(132)
plt.imshow(reconstructed_image_dft, cmap='gray')
plt.title(f'DFT Reconstructed\nError: {error_dft:.2f}')
plt.axis('off')

plt.subplot(133)
plt.imshow(reconstructed_image_dct, cmap='gray')
plt.title(f'DCT Reconstructed\nError: {error_dct:.2f}')
plt.axis('off')

plt.tight_layout()
plt.show()
```


Original Image
DFT Reconstructed
Error: 119.06
DCT Reconstructed
Error: 83.56