Signals Project Report

NAME	ID	SEC
Mina Hany William	9220895	2
Nour Khaled	9220921	2

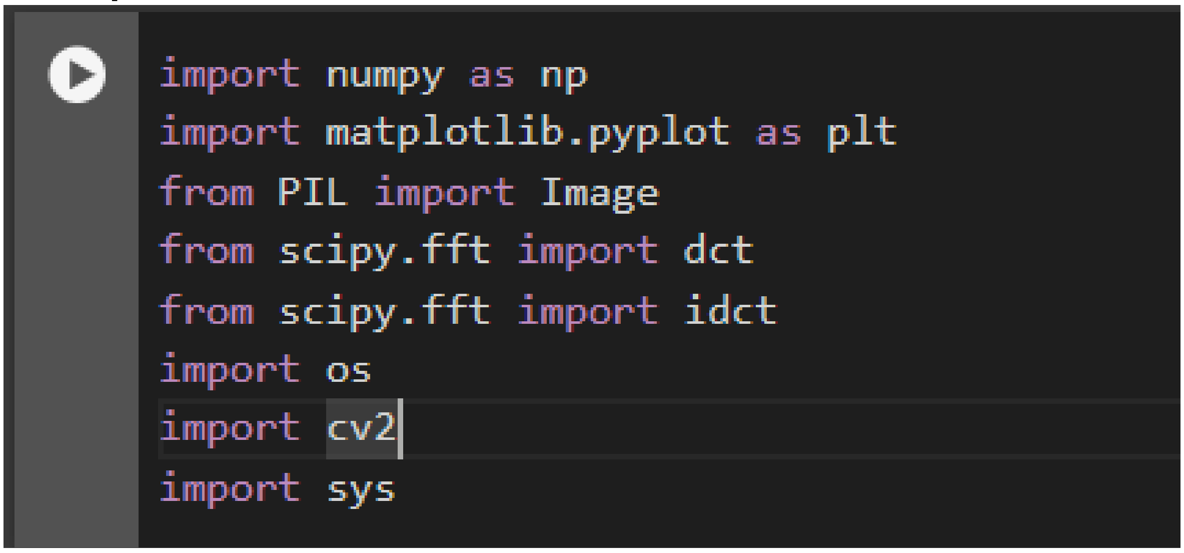
Presented To

Dr. Micheal Melek

Eng. Sayed Kamel

Contents

1- Imports



```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy.fft import dct
from scipy.fft import idct
import os
import cv2
import sys
```

Image 1: Importing the used libraries in project.

1- Numpy (np)

Library in python that deals with mathematical operations on vectors and scalars.

2- Matplotlib

Library in python for plotting data.

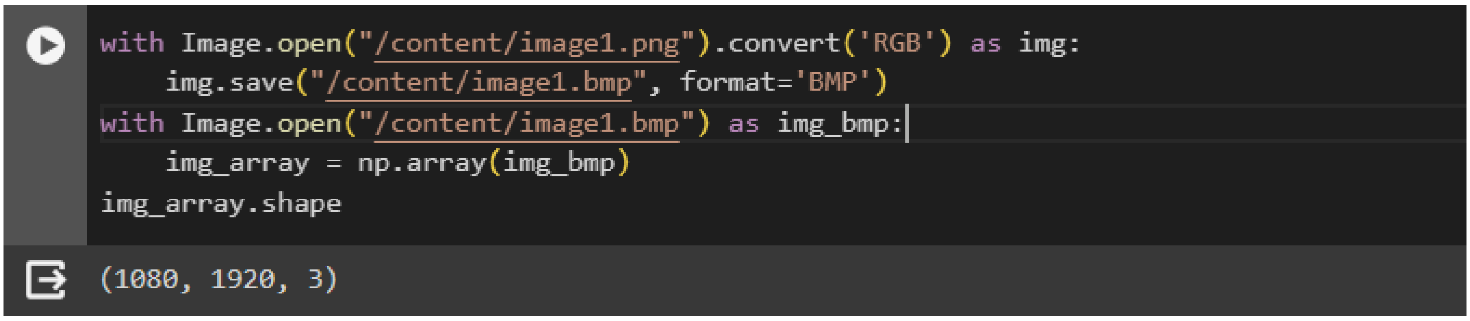
3- Scipy

Library in python that deals with operations on signals like DCT and IDCT.

4- PIL (Image)

Library in python that deals with images and processing it.

2- Read the image file



```
with Image.open("/content/image1.png").convert('RGB') as img:
    img.save("/content/image1.bmp", format='BMP')
with Image.open("/content/image1.bmp") as img_bmp:
    img_array = np.array(img_bmp)
img_array.shape
```

```
(1080, 1920, 3)
```

Image 2: Getting Image Data.

By utilizing PIL to read the image, we can extract its pixels as a 3D array. The array's first dimension corresponds to the pixel's row, the second to its column, and the third to its color channel (0-3).
The value in each cell represents the intensity of the RGB color, ranging from 0 to 255.

3- display each of its three color components.

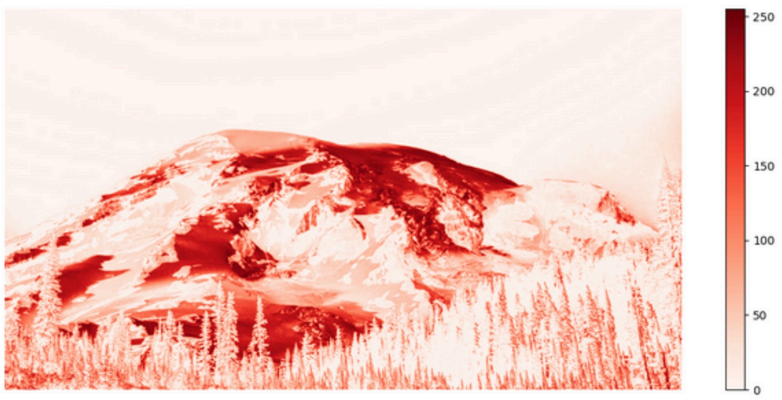


Image 3: Red component of Image.

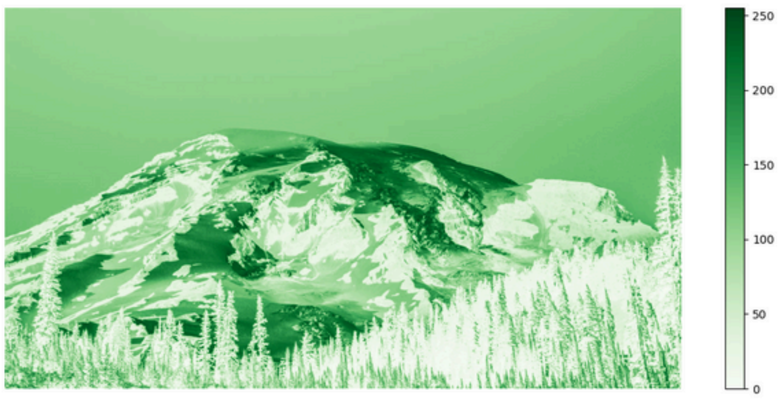


Image 4: Green component of Image.

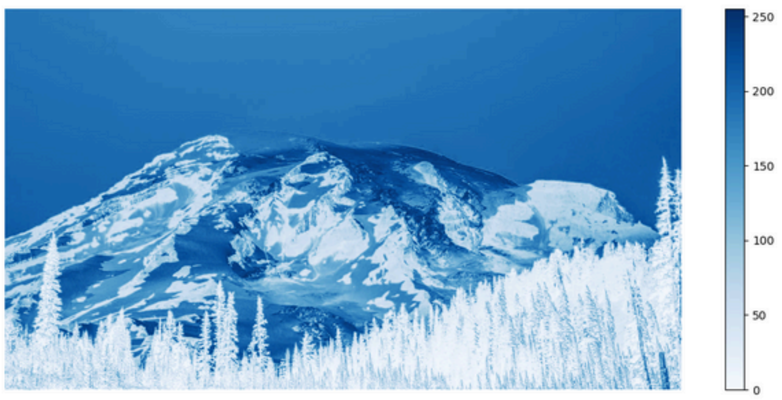
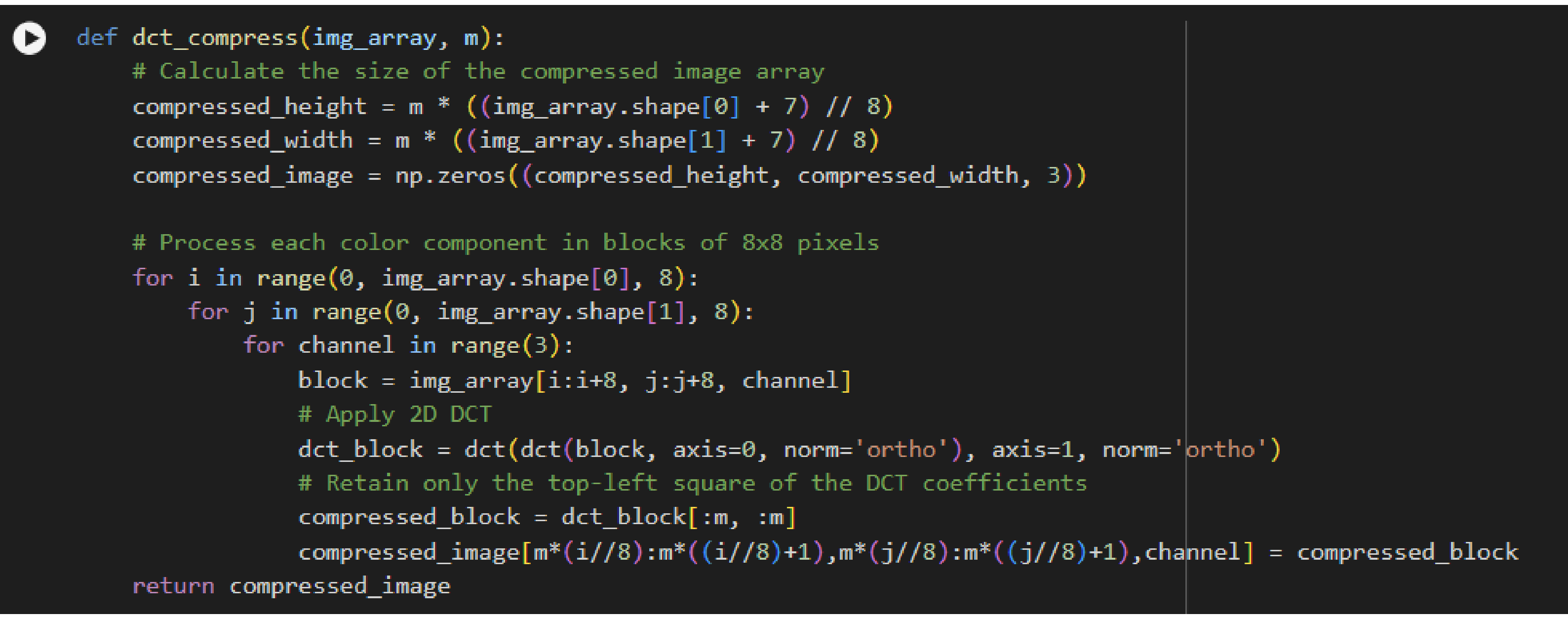


Image 5: Blue component of Image.

4- Compressing Image Using Discrete Cosines Transform (DCT)

As outlined in the project documentation, we treat each color channel of the image matrix as a collection of blocks, with each block consisting of 8x8 pixels. For each block, we apply the Discrete Cosine Transform (DCT) to extract the lower frequencies, which have a more pronounced effect on the human eye. This process involves focusing on the top left square of the 2D DCT matrix for various M values, while disregarding the higher frequencies.



```python
def dct_compress(img_array, m):
    # Calculate the size of the compressed image array
    compressed_height = m * ((img_array.shape[0] + 7) // 8)
    compressed_width = m * ((img_array.shape[1] + 7) // 8)
    compressed_image = np.zeros((compressed_height, compressed_width, 3))

    # Process each color component in blocks of 8x8 pixels
    for i in range(0, img_array.shape[0], 8):
        for j in range(0, img_array.shape[1], 8):
            for channel in range(3):
                block = img_array[i:i+8, j:j+8, channel]
                # Apply 2D DCT
                dct_block = dct(dct(block, axis=0, norm='ortho'), axis=1, norm='ortho')
                # Retain only the top-left square of the DCT coefficients
                compressed_block = dct_block[:m, :m]
                compressed_image[m*(i//8):m*((i//8)+1),m*(j//8):m*((j//8)+1),channel] = compressed_block
    return compressed_image
```

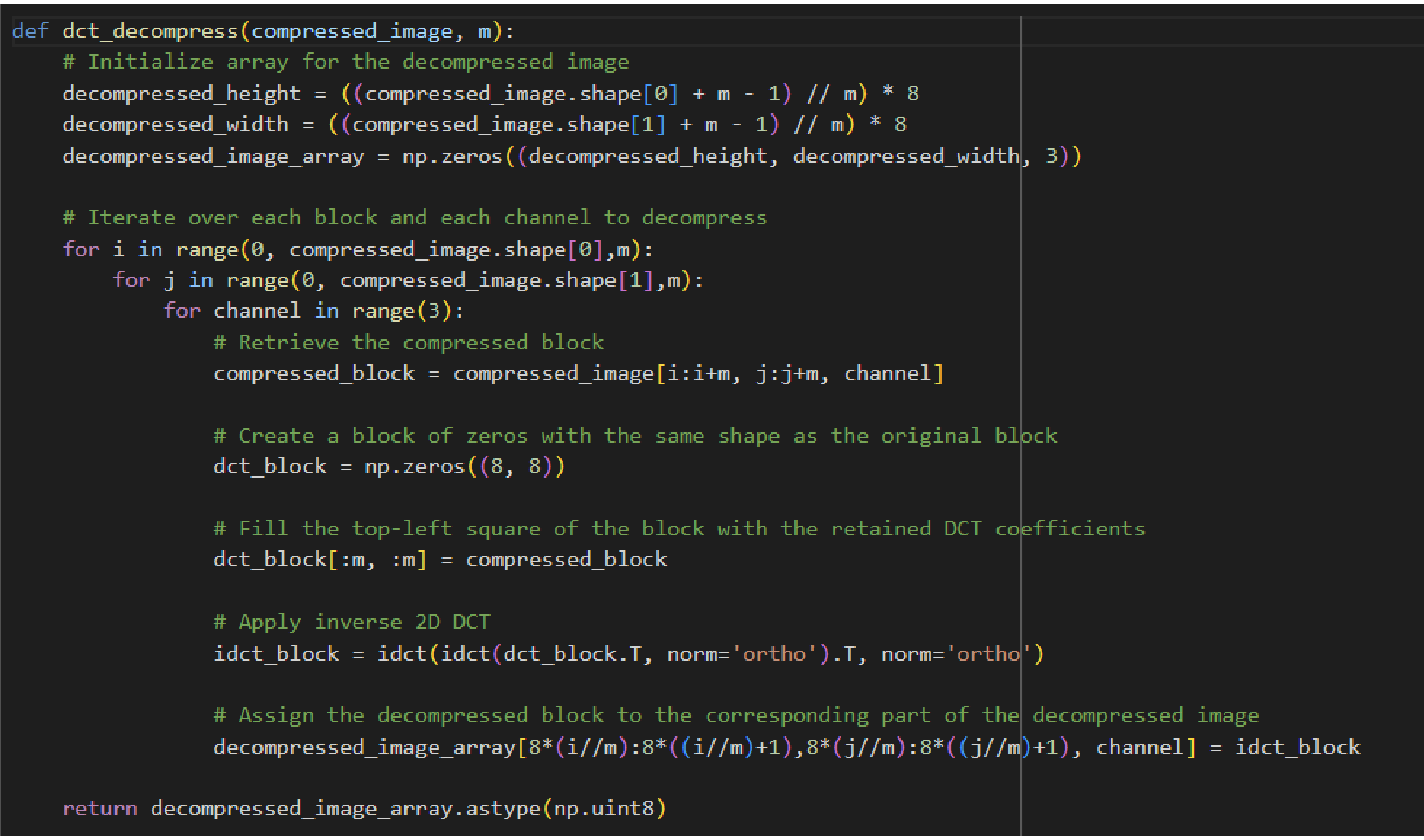
Image 5: Code For Compressing Image using DCT.

5- Compare the size of the original and compressed images.

Image	**array size**	**image size (MB)**
Original	6220800	5.9
Compressed with m = 1	97200	0.04
Compressed with m = 2	388800	0.2
Compressed with m = 3	874800	0.4
Compressed with m = 4	1555200	0.6

6- Decompress the image by applying inverse 2D DCT

To reconstruct the original image, we decompress the compressed image using the Inverse Discrete Cosine Transform (IDCT). However, we only apply the IDCT to the retained mxm block, setting all other values to zero. This process ensures that only the retained lower frequency components contribute to the final image reconstruction.



```python
def dct_decompress(compressed_image, m):
    # Initialize array for the decompressed image
    decompressed_height = ((compressed_image.shape[0] + m - 1) // m) * 8
    decompressed_width = ((compressed_image.shape[1] + m - 1) // m) * 8
    decompressed_image_array = np.zeros((decompressed_height, decompressed_width, 3))

    # Iterate over each block and each channel to decompress
    for i in range(0, compressed_image.shape[0],m):
        for j in range(0, compressed_image.shape[1],m):
            for channel in range(3):
                # Retrieve the compressed block
                compressed_block = compressed_image[i:i+m, j:j+m, channel]

                # Create a block of zeros with the same shape as the original block
                dct_block = np.zeros((8, 8))

                # Fill the top-left square of the block with the retained DCT coefficients
                dct_block[:m, :m] = compressed_block

                # Apply inverse 2D DCT
                idct_block = idct(idct(dct_block.T, norm='ortho').T, norm='ortho')

                # Assign the decompressed block to the corresponding part of the decompressed image
                decompressed_image_array[8*(i//m):8*((i//m)+1),8*(j//m):8*((j//m)+1), channel] = idct_block

    return decompressed_image_array.astype(np.uint8)
```

Image 6: Code For Decompressing The Compressed Image.

7- Showing images after decompression

Image 7: Original Image

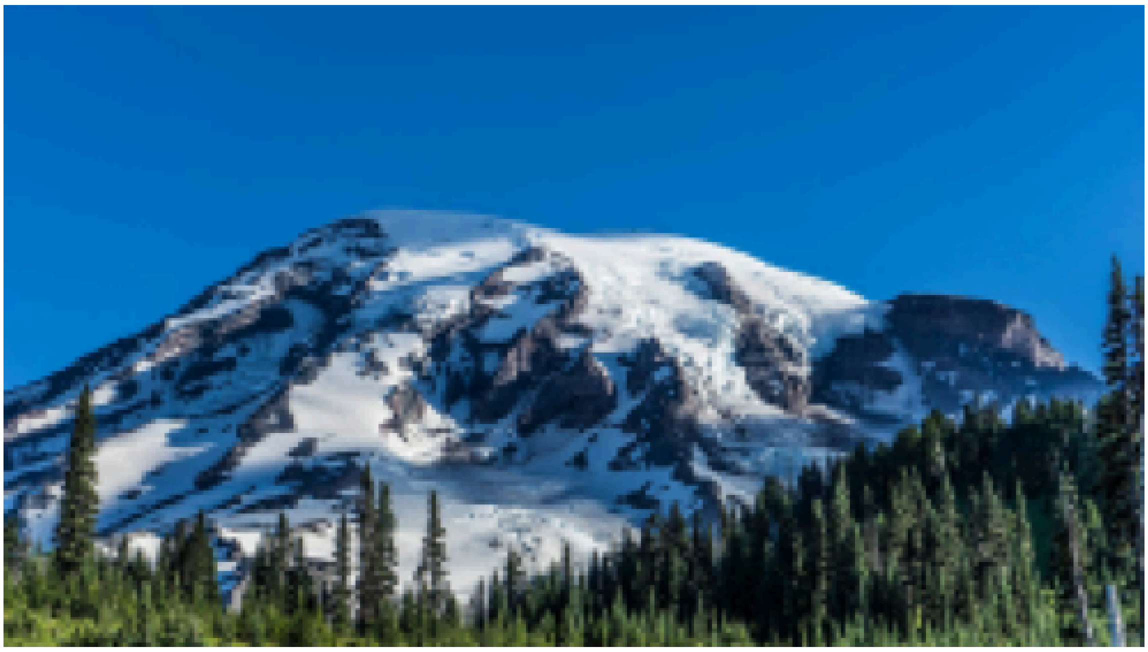


Image 8: Decompressed image using m = 1

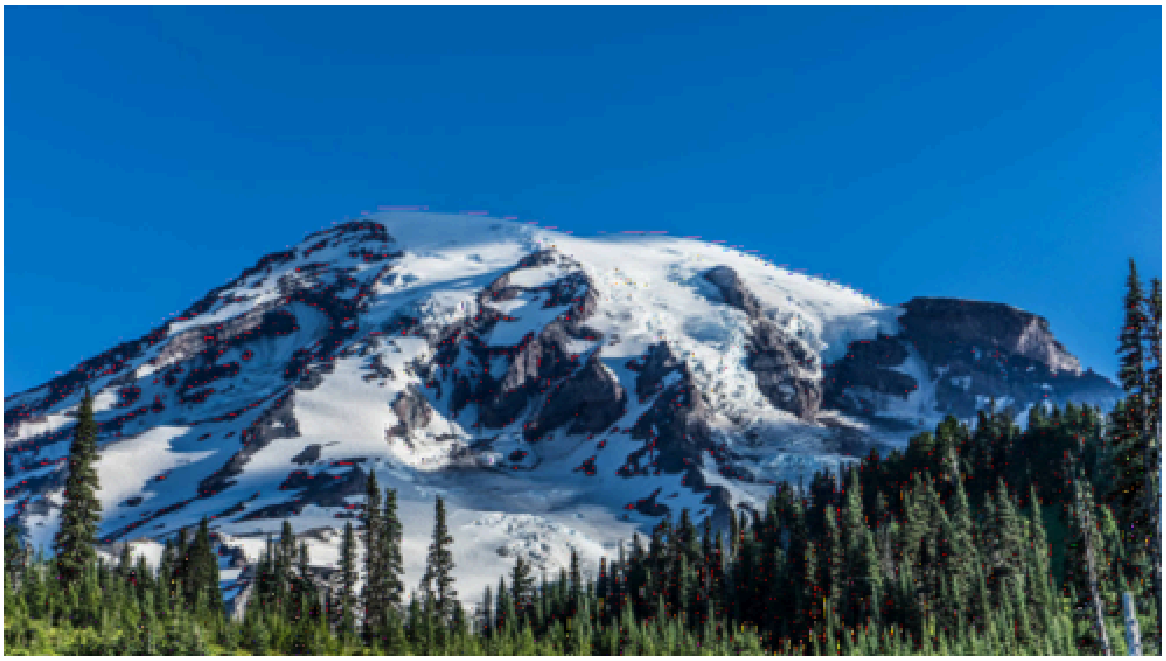


Image 9: Decompressed image using m = 2

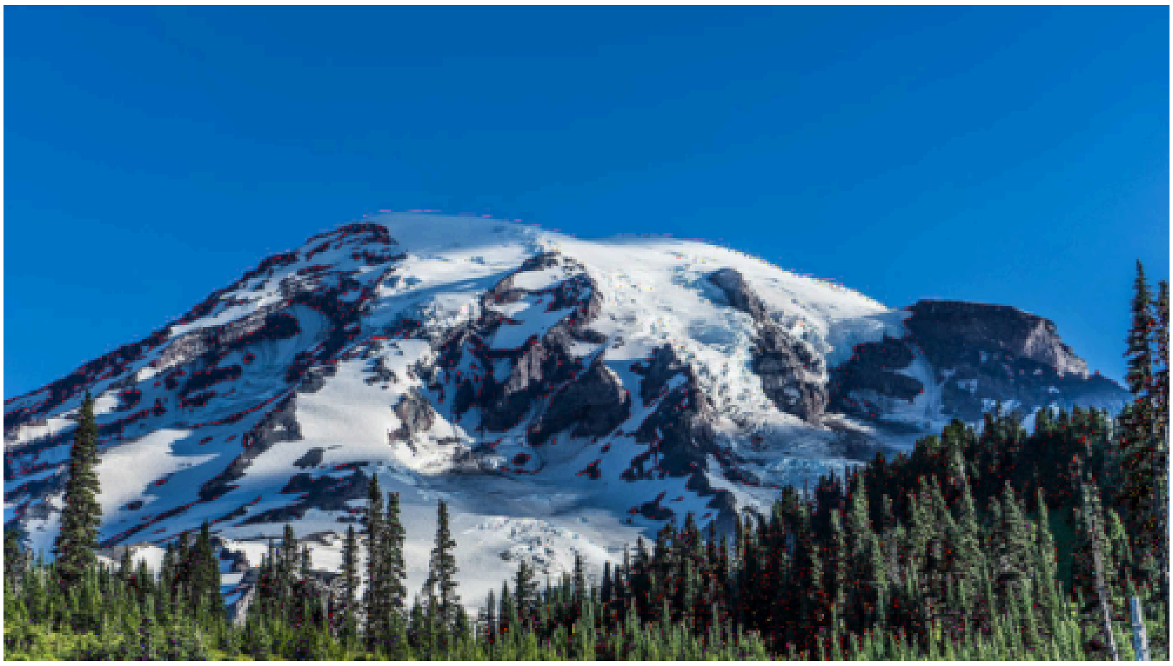


Image 10: Decompressed image using m = 3

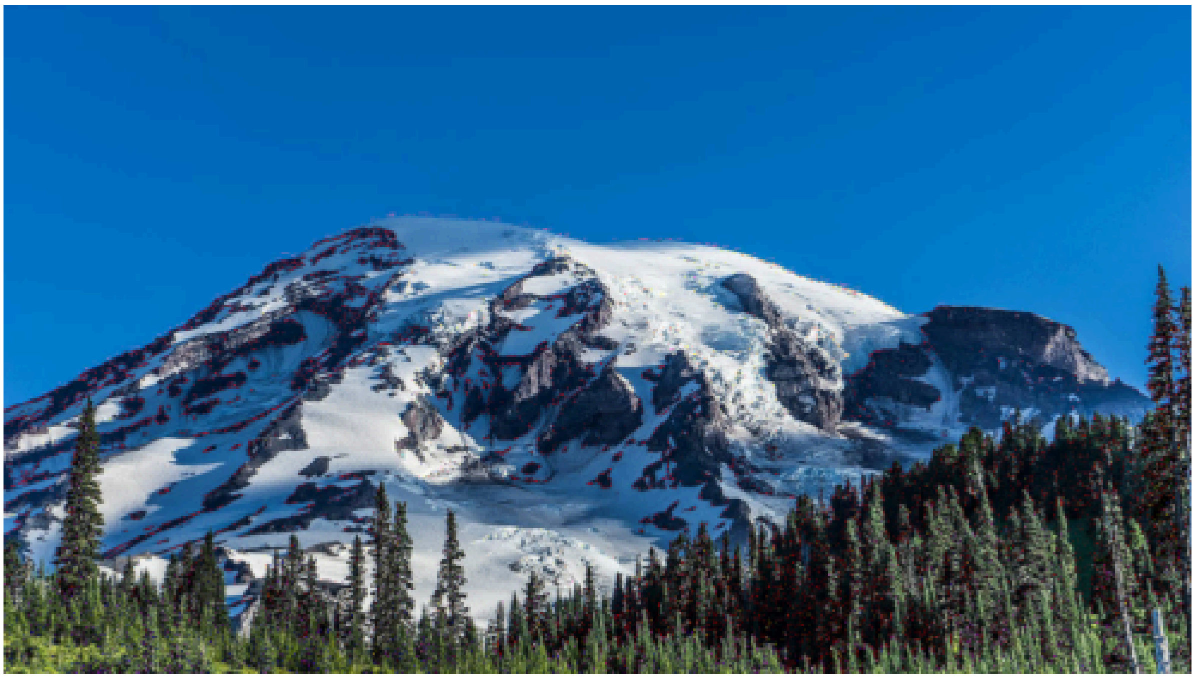


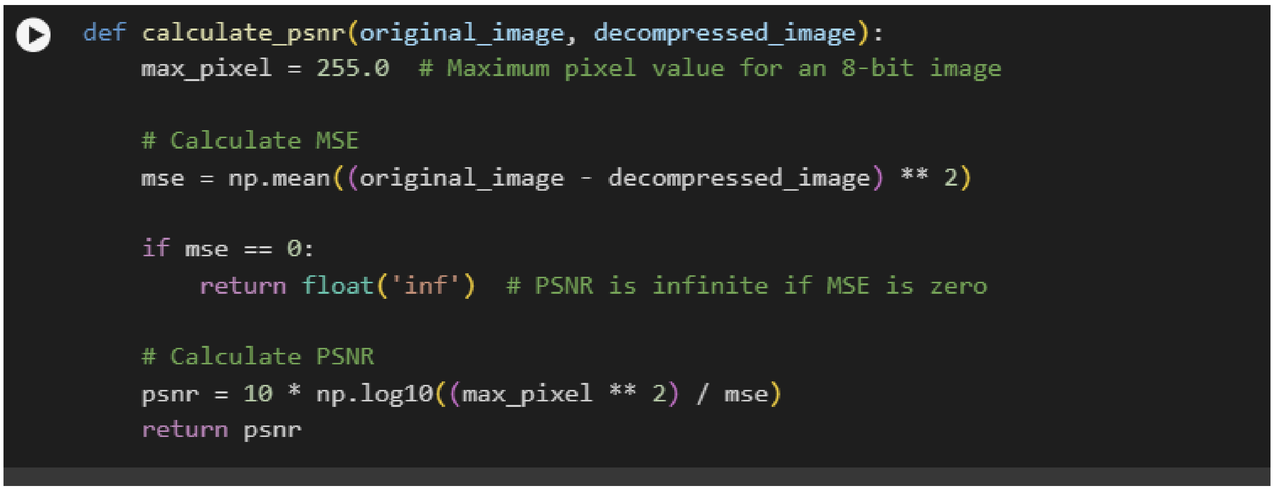
Image 11: Decompressed image using m = 4

8- Computing PSNR

The quality of the decompressed image is assessed using the Peak Signal-to-Noise Ratio (PSNR), calculated as:

$$PSNR = 10\log_{10}\frac{peak^2}{MSE}$$

Where peak is the max value of pixel data type and MSE is Mean square error



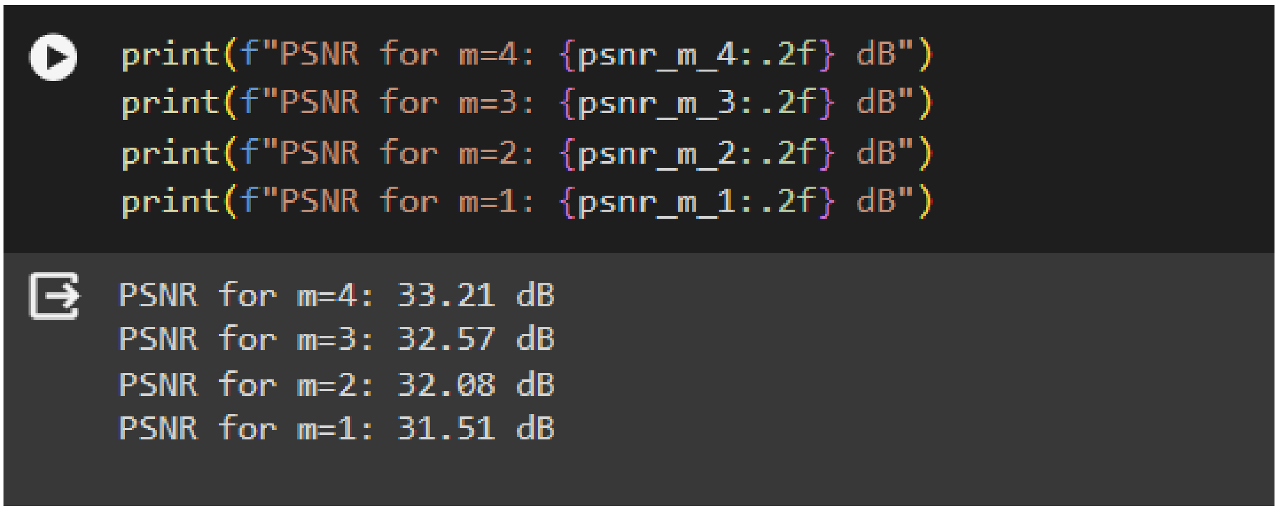
```python
def calculate_psnr(original_image, decompressed_image):
    max_pixel = 255.0  # Maximum pixel value for an 8-bit image

    # Calculate MSE
    mse = np.mean((original_image - decompressed_image) ** 2)

    if mse == 0:
        return float('inf')  # PSNR is infinite if MSE is zero

    # Calculate PSNR
    psnr = 10 * np.log10((max_pixel ** 2) / mse)
    return psnr
```

Image 12: Code For Calculating PSNR.



```python
print(f"PSNR for m=4: {psnr_m_4:.2f} dB")
print(f"PSNR for m=3: {psnr_m_3:.2f} dB")
print(f"PSNR for m=2: {psnr_m_2:.2f} dB")
print(f"PSNR for m=1: {psnr_m_1:.2f} dB")
```

```
PSNR for m=4: 33.21 dB
PSNR for m=3: 32.57 dB
PSNR for m=2: 32.08 dB
PSNR for m=1: 31.51 dB
```

Image 13: Code For Printing PSNR.

9- curve displaying the PSNR against m

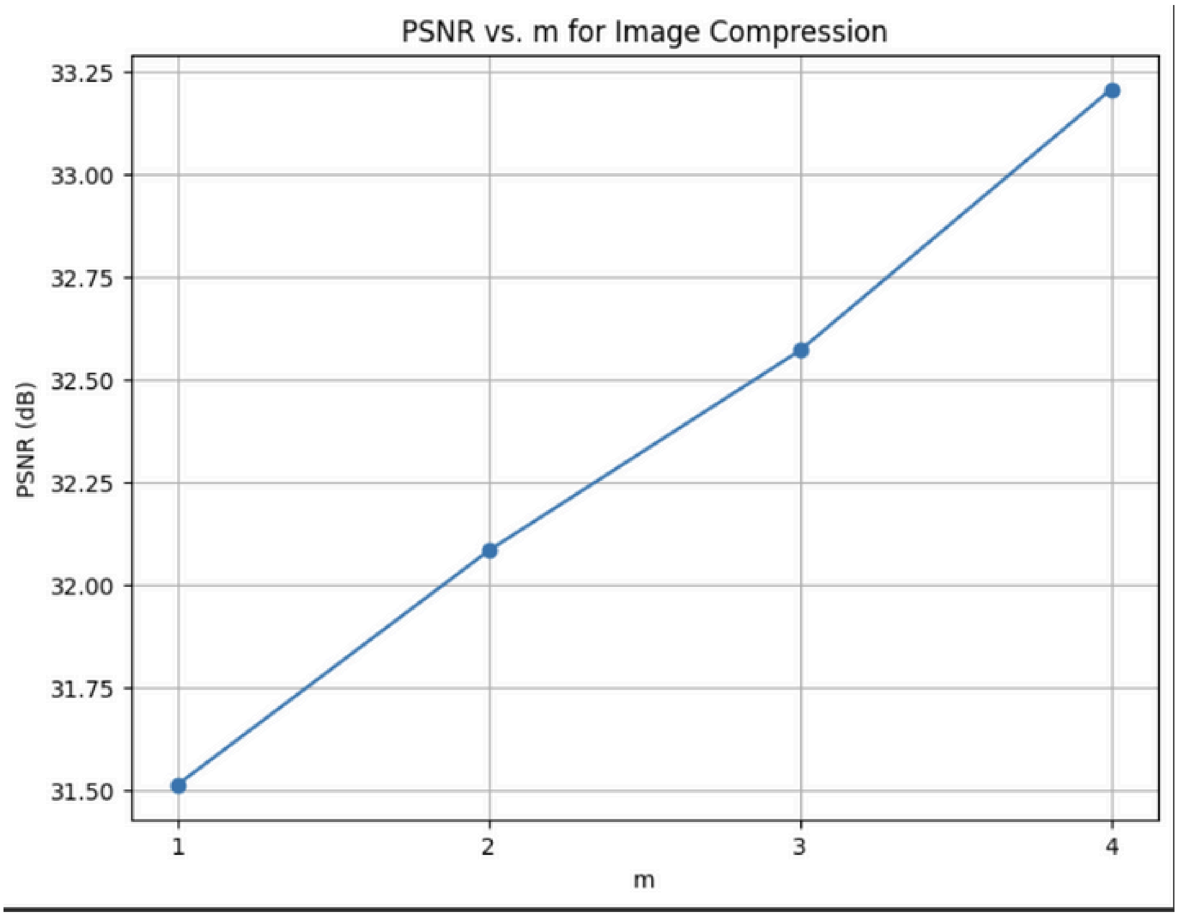


Image 14: curve displaying the PSNR against m

Increasing the value of m indeed allows us to retain more effective frequencies, which can enhance the quality of the decompressed image. This increase in effective frequencies typically leads to a higher PSNR (Peak Signal-to-Noise Ratio) as well. Therefore, there is a direct proportionality between the value of mm and the PSNR: as m increases, the PSNR also tends to increase, indicating better image quality.

10- Code

```
###############################Imports########################################
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy.fft import dct
from scipy.fft import idct
import os
import cv2
import sys

output_dir = 'compressed_images'
if not os.path.exists(output_dir):
  os.makedirs(output_dir)

#Reads image and convert to rgb
with Image.open("/content/image1.png").convert('RGB') as img:
  img.save("/content/image1.bmp", format='BMP')
with Image.open("/content/image1.bmp") as img_bmp:
  img_array = np.array(img_bmp)
img_array.shape

output_dir = 'compressed_images'
if not os.path.exists(output_dir):
  os.makedirs(output_dir)

colors = ["Reds","Greens","Blues"]
for i in range(3):
  plt.figure(figsize=(14, 6))
  plt.imshow(img_array[:,:,i],cmap=colors[i])
  plt.axis("off")
  plt.colorbar()
  plt.savefig(f"a_{colors[i][:-1].lower()}.png")
  plt.show()
```

```
def dct_compress(img_array, m):
  # Calculate the size of the compressed image array
  compressed_height = m * ((img_array.shape[0] + 7) // 8)
  compressed_width = m * ((img_array.shape[1] + 7) // 8)
  compressed_image = np.zeros((compressed_height, compressed_width, 3))

  # Process each color component in blocks of 8x8 pixels
  for i in range(0, img_array.shape[0], 8):
    for j in range(0, img_array.shape[1], 8):
      for channel in range(3):
        block = img_array[i:i+8, j:j+8, channel]
        # Apply 2D DCT
        dct_block = dct(dct(block, axis=0, norm='ortho'), axis=1, norm='ortho')
        # Retain only the top-left square of the DCT coefficients
        compressed_block = dct_block[:m, :m]
        compressed_image[m*(i//8):m*((i//8)+1),m*(j//8):m*((j//8)+1),channel] = compressed_block
  return compressed_image
```

```
def dct_decompress(compressed_image, m):
 # Initialize array for the decompressed image
 decompressed_height = ((compressed_image.shape[0] + m - 1) // m) * 8
 decompressed_width = ((compressed_image.shape[1] + m - 1) // m) * 8
 decompressed_image_array = np.zeros((decompressed_height, decompressed_width, 3))

 # Iterate over each block and each channel to decompress
 for i in range(0, compressed_image.shape[0],m):
  for j in range(0, compressed_image.shape[1],m):
   for channel in range(3):
    # Retrieve the compressed block
    compressed_block = compressed_image[i:i+m, j:j+m, channel]

    # Create a block of zeros with the same shape as the original block
    dct_block = np.zeros((8, 8))

    # Fill the top-left square of the block with the retained DCT coefficients
    dct_block[:m, :m] = compressed_block

    # Apply inverse 2D DCT
    idct_block = idct(idct(dct_block.T, norm='ortho').T, norm='ortho')

    # Assign the decompressed block to the corresponding part of the decompressed image
    decompressed_image_array[8*(i//m):8*((i//m)+1),8*(j//m):8*((j//m)+1), channel] = idct_block

 return decompressed_image_array.astype(np.uint8)
```

```
def display_image(image_array,m):
 plt.imshow(image_array)
 plt.axis('off') # Turn off axis numbers
 plt.savefig(f"decompressed_image_{m}.png")
 plt.show()

def calculate_psnr(original_image, decompressed_image):
 max_pixel = 255.0 # Maximum pixel value for an 8-bit image

 # Calculate MSE
 mse = np.mean((original_image - decompressed_image) ** 2)

 if mse == 0:
  return float('inf') # PSNR is infinite if MSE is zero

 # Calculate PSNR
 psnr = 10 * np.log10((max_pixel ** 2) / mse)
 return psnr
```

```
#m->4
m = 4
compressed_image_m_4 = dct_compress(img_array , m)
compressed_image_m_4
original_size = img_array.size
compressed_size = compressed_image_m_4.size
print(f"Original size: {original_size} ")
print(f"Compressed size: {compressed_size} ")
compressed_image_m_4_gray = cv2.cvtColor(compressed_image_m_4.astype(np.uint8), cv2.COLOR_BGR2GRAY)
cv2.imwrite(os.path.join(output_dir, f'compressed_image_m{m}.bmp'), compressed_image_m_4_gray)
decompressed_image_array_m_4 = dct_decompress(compressed_image_m_4,m)
decompressed_image_array_m_4
decompressed_image_array_m_4.size
display_image(decompressed_image_array_m_4,m)
psnr_m_4 = calculate_psnr(img_array,decompressed_image_array_m_4)
print(f"PSNR for m=4: {psnr_m_4:.2f} dB")
```

```
#m->3
m = 3
compressed_image_m_3 = dct_compress(img_array , m)
compressed_image_m_3
original_size = img_array.size
compressed_size = compressed_image_m_3.size
print(f"Original size: {original_size} ")
print(f"Compressed size: {compressed_size} ")
compressed_image_m_3_gray = cv2.cvtColor(compressed_image_m_3.astype(np.uint8), cv2.COLOR_BGR2GRAY)
cv2.imwrite(os.path.join(output_dir, f'compressed_image_m{m}.bmp'), compressed_image_m_3_gray)
decompressed_image_array_m_3 = dct_decompress(compressed_image_m_3,m)
decompressed_image_array_m_3
decompressed_image_array_m_3.size
display_image(decompressed_image_array_m_3,m)
psnr_m_3 = calculate_psnr(img_array,decompressed_image_array_m_3)
print(f"PSNR for m=3: {psnr_m_3:.2f} dB")
```

```
#m->2
m = 2
compressed_image_m_2 = dct_compress(img_array , m)
compressed_image_m_2
original_size = img_array.size
compressed_size = compressed_image_m_2.size
print(f"Original size: {original_size} ")
print(f"Compressed size: {compressed_size} ")
compressed_image_m_2_gray = cv2.cvtColor(compressed_image_m_2.astype(np.uint8), cv2.COLOR_BGR2GRAY)
cv2.imwrite(os.path.join(output_dir, f'compressed_image_m{m}.bmp'), compressed_image_m_2_gray)
decompressed_image_array_m_2 = dct_decompress(compressed_image_m_2,m)
decompressed_image_array_m_2
decompressed_image_array_m_2.size
display_image(decompressed_image_array_m_2,m)
psnr_m_2 = calculate_psnr(img_array,decompressed_image_array_m_2)
print(f"PSNR for m=2: {psnr_m_2:.2f} dB")
```

```
#m->1
m = 1
compressed_image_m_1 = dct_compress(img_array , m)
compressed_image_m_1
original_size = img_array.size
compressed_size = compressed_image_m_1.size
print(f"Original size: {original_size} ")
print(f"Compressed size: {compressed_size} ")
compressed_image_m_1_gray = cv2.cvtColor(compressed_image_m_1.astype(np.uint8), cv2.COLOR_BGR2GRAY)
cv2.imwrite(os.path.join(output_dir, f'compressed_image_m{m}.bmp'), compressed_image_m_1_gray)
decompressed_image_array_m_1 = dct_decompress(compressed_image_m_1,m)
decompressed_image_array_m_1
decompressed_image_array_m_1.size
display_image(decompressed_image_array_m_1,m)
psnr_m_1 = calculate_psnr(img_array,decompressed_image_array_m_1)
print(f"PSNR for m=1: {psnr_m_1:.2f} dB")
```

```
#curve
m_values = [1, 2, 3, 4]
psnr_values = [psnr_m_1, psnr_m_2, psnr_m_3, psnr_m_4]
# Plot the graph
plt.figure(figsize=(8, 6))
plt.plot(m_values, psnr_values, marker='o', linestyle='-')
plt.xlabel('m')
plt.ylabel('PSNR (dB)')
plt.title('PSNR vs. m for Image Compression')
plt.grid(True)
plt.xticks(m_values)
plt.show()
```

Running on Google Colab