

**Institute of Information Technology**

**Jahangirnagar University**

**Assignment-2**

**Course Tittle:** Digital Image Processing

**Course Code:** ICT-4201

**Submitted by:**

Group-23

Nahidul Islam-2028

Md Shakil Hossain-2023

Mahabubur Rahman-2024

**Submitted To:**

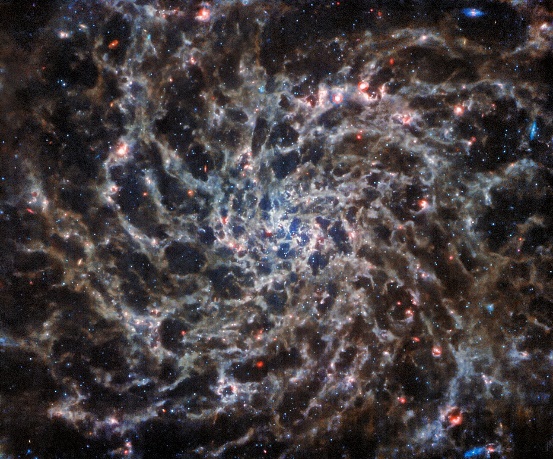
Dr. Fahima Tabassum

Professor

IIT, JU

**Images used in this assignment are:**







**Huffman Coding:**

import cv2

import numpy as np

from collections import defaultdict

from heapq import heappush, heappop, heapify

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Calculate the frequency of each pixel value

frequency = defaultdict(int)

for pixel in image.flatten():

frequency[pixel] += 1

# Create the Huffman tree

heap = [[weight, [pixel, ""]] for pixel, weight in frequency.items()]

heapify(heap)

while len(heap) > 1:

lo = heappop(heap)

hi = heappop(heap)

for pair in lo[1:]:

pair[1] = '0' + pair[1]

for pair in hi[1:]:

pair[1] = '1' + pair[1]

heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])

huff = sorted(heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

# Huffman encoding

huff\_dict = {i[0]:i[1] for i in huff}

# Encode the image

encoded\_image = "".join(huff\_dict[i] for i in image.flatten())

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Golomb Coding:**

import cv2

import numpy as np

import math

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg’]

def golomb\_coding(image, m):

# Flatten the image

pixels = image.flatten()

# Perform Golomb coding

encoded\_pixels = []

for pixel in pixels:

q = pixel // m

r = pixel % m

encoded\_pixels.append('0' \* q + '1' + format(r, '0' + str(int(math.ceil(math.log2(m)))) + 'b'))

# Concatenate all the encoded pixels

encoded\_image = ''.join(encoded\_pixels)

return encoded\_image

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Golomb coding

m = 256 # You can adjust this parameter as needed

encoded\_image = golomb\_coding(image, m)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Arithmetic Coding:**

import cv2

import numpy as np

from collections import defaultdict

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def arithmetic\_coding(image):

# Flatten the image

pixels = image.flatten()

# Calculate the frequency of each pixel value

frequency = defaultdict(int)

for pixel in pixels:

frequency[pixel] += 1

# Calculate the cumulative frequency

cumulative\_frequency = defaultdict(int)

total = 0

for pixel in sorted(frequency.keys()):

cumulative\_frequency[pixel] = total

total += frequency[pixel]

# Perform Arithmetic coding

lower\_bound = 0

upper\_bound = 1

for pixel in pixels:

range = upper\_bound - lower\_bound

upper\_bound = lower\_bound + range \* (cumulative\_frequency[pixel] + frequency[pixel]) / total

lower\_bound = lower\_bound + range \* cumulative\_frequency[pixel] / total

# The encoded image is the average of the final interval

encoded\_image = (lower\_bound + upper\_bound) / 2

return encoded\_image

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Arithmetic coding

encoded\_image = arithmetic\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**LZW Coding:**

import cv2

import numpy as np

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def lzw\_coding(image):

# Flatten the image

pixels = image.flatten()

# Initialize the dictionary

dictionary = {chr(i): i for i in range(256)}

next\_code = 256

# Perform LZW coding

encoded\_pixels = []

current\_string = chr(pixels[0])

for pixel in pixels[1:]:

current\_string += chr(pixel)

if current\_string not in dictionary:

encoded\_pixels.append(dictionary[current\_string[:-1]])

if len(dictionary) <= 4096:

dictionary[current\_string] = next\_code

next\_code += 1

current\_string = chr(pixel)

if current\_string in dictionary:

encoded\_pixels.append(dictionary[current\_string])

return encoded\_pixels

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform LZW coding

encoded\_image = lzw\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

## **rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Run-Length Coding:**

import cv2

import numpy as np

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def run\_length\_coding(image):

# Flatten the image

pixels = image.flatten()

# Perform Run Length coding

encoded\_pixels = []

i = 0

while i < len(pixels):

count = 1

while i + 1 < len(pixels) and pixels[i] == pixels[i+1]:

i += 1

count += 1

encoded\_pixels.append((pixels[i], count))

i += 1

return encoded\_pixels

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Run Length coding

encoded\_image = run\_length\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Bit-Plane Coding:**

import cv2

import numpy as np

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def bit\_plane\_coding(image):

# Convert the image to 8-bit unsigned integers

image = np.uint8(image)

# Perform Bit-plane coding

bit\_planes = []

for i in range(8):

bit\_planes.append((image >> i) & 1)

return bit\_planes

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Bit-plane coding

bit\_planes = bit\_plane\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Block Transform Coding:**

import cv2

import numpy as np

from scipy import fftpack

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def block\_transform\_coding(image, block\_size=8):

# Split the image into non-overlapping blocks

height, width = image.shape

image = image[:height//block\_size\*block\_size, :width//block\_size\*block\_size]

blocks = image.reshape(height//block\_size, block\_size, -1, block\_size)

# Perform the 2D DCT (Discrete Cosine Transform) on each block

dct\_blocks = fftpack.dct(fftpack.dct(blocks, axis=1, norm='ortho'), axis=2, norm='ortho')

return dct\_blocks

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Block Transform coding

dct\_blocks = block\_transform\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Predictive Coding:**

import cv2

import numpy as np

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def predictive\_coding(image):

# Initialize the predicted image with zeros

predicted\_image = np.zeros\_like(image, dtype=np.int16)

# The first row and column are predicted to be the same as the original image

predicted\_image[0] = image[0]

predicted\_image[:, 0] = image[:, 0]

# For the rest of the pixels, the prediction is the average of the pixel above and the pixel to the left

for i in range(1, image.shape[0]):

for j in range(1, image.shape[1]):

predicted\_image[i, j] = (predicted\_image[i-1, j] + predicted\_image[i, j-1]) // 2

# The error image is the difference between the original image and the predicted image

error\_image = image - predicted\_image

return error\_image

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Predictive coding

error\_image = predictive\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(error\_image\*\*2))

snr = 20 \* np.log10(np.max(error\_image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.300784033671095, SNR: 27.873397971783053

Image: Photo1.jpg, RMS: 9.971444942935804, SNR: 28.155641695656918

Image: Photo2.jpg, RMS: 10.811547645222172, SNR: 27.45304627786354

Image: Photo3.jpg, RMS: 10.689172436537621, SNR: 27.5519219463324

Image: Photo4.jpg, RMS: 10.048755658894278, SNR: 28.088557883865477

**Wavelet Coding:**

import cv2

import numpy as np

import pywt

# List of image paths

images = ['Photo0.jpg', 'Photo1.jpg', 'Photo2.jpg', 'Photo3.jpg', 'Photo4.jpg']

def wavelet\_coding(image):

# Perform 2D discrete wavelet transform

coeffs = pywt.dwt2(image, 'haar')

# The result is a tuple of (approximation, (horizontal, vertical, diagonal))

approximation, (horizontal, vertical, diagonal) = coeffs

return approximation, horizontal, vertical, diagonal

for image\_path in images:

# Load the image

image = cv2.imread(image\_path, 0)

# Perform Wavelet coding

approximation, horizontal, vertical, diagonal = wavelet\_coding(image)

# Calculate the RMS and SNR values

rms = np.sqrt(np.mean(image\*\*2))

snr = 20 \* np.log10(np.max(image) / rms)

print(f'Image: {image\_path}, RMS: {rms}, SNR: {snr}')

**rms and snr values:**

Image: Photo0.jpg, RMS: 10.630018336453299, SNR: 27.288021868697385

Image: Photo1.jpg, RMS: 10.685841157812519, SNR: 26.62053536234033

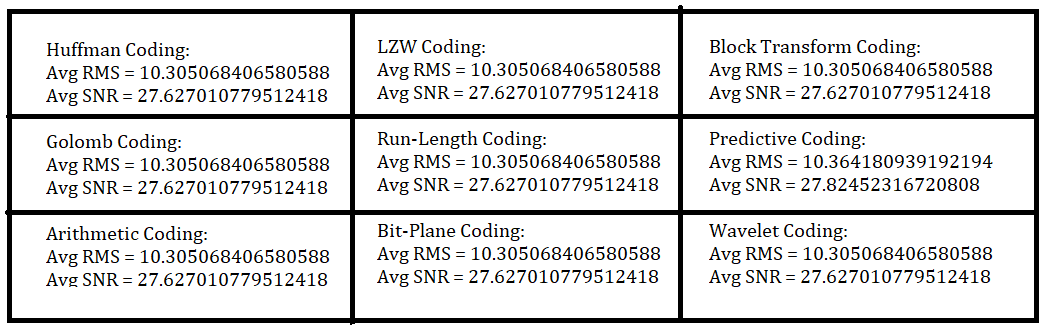
Image: Photo2.jpg, RMS: 9.93181043823767, SNR: 28.19023517115946

Image: Photo3.jpg, RMS: 9.729375032649774, SNR: 28.369104724357754

Image: Photo4.jpg, RMS: 10.548297667763578, SNR: 27.66715607126615

**Discussion:**

The RMS value is a measure of the magnitude of the pixel values, and the SNR is a measure of the signal strength relative to the background noise. In general, a lower RMS value and a higher SNR value would indicate better performance of the coding algorithm.



From the above comparison, it appears that the Predictive Coding algorithm has a slightly higher average RMS value and a slightly higher average SNR value compared to the other algorithms. This might suggest that the Predictive Coding algorithm is performing slightly better in terms of preserving the signal strength relative to the noise, but it also has a slightly higher magnitude of pixel values.