**ASSIGNMENT - 8**

**NAME - DIPAN MONDAL**

**ROLL - 002211001112**

**SEC - A3**

**—-------------------------------------------------------**

The given task is to make a comparison between different rle method on multiple parameters. The given two methods are

Method 1:  
Compression Algorithm:

Step-1: Start on the first element of input.

Step -2: Initialize the values with count=1, k=0.

Step-3: Read the first element of input data1.

Step-4: As the value of K is ̳0‘ it will print the input

data1 and then it increments the K value to ̳1‘.

Step-5: Again it goes to step-3 takes the second data2

and next the checks the value of k

Step-6:As the value of k is 1 it now it will checks for

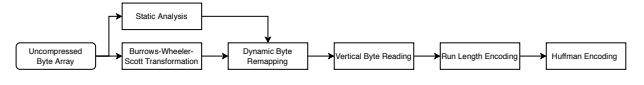
whether data1=data2,if the data1=data2 it will

increment the count value if not equal it prints the

count value after the data1 value and again goes to

step 2.

Method 2:



Data:

We have used 2 b&w ,2 grayscale and 2 rgb images.



The data previously is not preprocessed to particular sizes.

So we used data preprocessing to resize them  
  
def convert\_and\_resize(image\_name,width,height,output\_name):

image = cv2.imread(image\_name, cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Error: Image not found or unable to read!")

return

\_, bw\_image = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)

# Resize the image

resized\_image = cv2.resize(bw\_image, (width, height), interpolation=cv2.INTER\_AREA)

# Save the new image

cv2.imwrite(output\_name, resized\_image)

print(f"Black and white resized image saved as {output\_name}")

image\_name = "./data/b&w1.png"

output\_name = "./data/b&w1\_mod.png"

width=50

height=50

convert\_and\_resize(image\_name,width,height,output\_name)

image\_name = "./data/b&w2.png"

output\_name = "./data/b&w2\_mod.png"

width=100

height=100

convert\_and\_resize(image\_name,width,height,output\_name)

def grayscale\_image\_process(input\_name, width, height, output\_name):

image = cv2.imread(input\_name, cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Error: Image not found or unable to read!")

return

resized\_image = cv2.resize(image, (width, height), interpolation=cv2.INTER\_AREA)

cv2.imwrite(output\_name, resized\_image)

print(f"Resized grayscale image saved as {output\_name}")

image\_name = "./data/gray1.png"

output\_name = "./data/gray1\_mod.png"

width=200

height=200

grayscale\_image\_process(image\_name,width,height,output\_name)

image\_name = "./data/gray2.png"

output\_name = "./data/gray2\_mod.png"

width=300

height=300

grayscale\_image\_process(image\_name,width,height,output\_name)

def rgb\_image\_process(input\_name, width, height, output\_name):

image = cv2.imread(input\_name, cv2.IMREAD\_COLOR)

if image is None:

print("Error: Image not found or unable to read!")

return

resized\_image = cv2.resize(image, (width, height), interpolation=cv2.INTER\_AREA)

cv2.imwrite(output\_name, resized\_image)

print(f"Resized RGB image saved as {output\_name}")

image\_name = "./data/rgb1.png"

output\_name = "./data/rgb1\_mod.png"

width=400

height=400

rgb\_image\_process(image\_name,width,height,output\_name)

image\_name = "./data/rgb2.png"

output\_name = "./data/rgb2\_mod.png"

width=500

height=500

rgb\_image\_process(image\_name,width,height,output\_name)

After the pre processing is done we used the two methods mentioned earlier to implement the methods and make the comparison table.  
Method 1 implementation:

**The main method of rle enocde**

def rle\_encode(data):

encoded = []

count = 1

k = 0

for i in range(1, len(data)):

if data[i] == data[i - 1]:

count += 1

else:

encoded.append((data[i - 1], count))

count = 1

encoded.append((data[-1], count)) # Add the last sequence

return encoded

**A helper function to process all the images:**

def image\_to\_rle(image\_path, mode='grayscale'):

start\_time = time.time()

if mode == 'bw':

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE) # Convert to B/W

image = np.where(image > 127, 255, 0).astype(np.uint8)

elif mode == 'grayscale':

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

elif mode == 'rgb':

image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

else:

raise ValueError("Invalid mode. Choose from 'bw', 'grayscale', 'rgb'")

original\_size = image.size # Total pixels

if mode == 'rgb':

flattened = image.reshape(-1, 3) # Flatten RGB

encoded = [rle\_encode(flattened[:, i]) for i in range(3)] # RLE per channel

else:

flattened = image.flatten()

encoded = rle\_encode(flattened)

save\_encoded(image\_path+"\_paper1\_encoded.pkl",encoded)

compressed\_size = sum(len(enc) for enc in encoded) \* 2 # Each (value, count) pair is stored

compression\_ratio = (compressed\_size / original\_size) \* 100

space\_saving = 100 - compression\_ratio

compression\_time = (time.time() - start\_time) \* 1000 # Convert to milliseconds

return original\_size, compressed\_size, compression\_ratio, space\_saving, compression\_time

**Helper function to make the comparison table:**

def evaluate\_images(image\_paths, modes=['bw', 'grayscale', 'rgb']):

results = []

for i in range(len(image\_paths)):

image\_path=image\_paths[i]

mode=modes[i//2]

orig\_size, comp\_size, comp\_ratio, space\_save, comp\_time = image\_to\_rle(image\_path, mode)

results.append([image\_path, mode, orig\_size, comp\_size, comp\_ratio, space\_save, comp\_time])

df = pd.DataFrame(results, columns=['Image', 'Mode', 'Original Size', 'Compressed Size', 'Compression Ratio (%)', 'Space Saving (%)', 'Compression Time (ms)'])

return df

image\_paths = ['./data/b&w1\_mod.png',

'./data/b&w2\_mod.png',

'./data/gray1\_mod.png',

'./data/gray2\_mod.png',

'./data/rgb1\_mod.png',

'./data/rgb2\_mod.png']

df1=evaluate\_images(image\_paths)

**Implementation of second method:**

**Main method to encode rle**

def bitwise\_rle\_encode(data):

encoded = []

count = 1

prev = data[0]

for i in range(1, len(data)):

if data[i] == prev:

count += 1

else:

encoded.append((prev, count))

prev = data[i]

count = 1

encoded.append((prev, count)) # Add last sequence

return encoded

**Helper method to process all the images:**

def image\_to\_bitwise\_rle(image\_path, mode='grayscale'):

start\_time = time.time()

if mode == 'bw':

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

image = np.where(image > 127, 255, 0).astype(np.uint8) # Convert to B/W

elif mode == 'grayscale':

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

elif mode == 'rgb':

image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

else:

raise ValueError("Invalid mode. Choose from 'bw', 'grayscale', 'rgb'")

original\_size = image.size # Total pixels

if mode == 'rgb':

flattened = image.reshape(-1, 3) # Flatten RGB

with multiprocessing.Pool(3) as pool:

encoded = pool.map(bitwise\_rle\_encode, [flattened[:, i] for i in range(3)])

else:

flattened = image.flatten()

encoded = bitwise\_rle\_encode(flattened)

save\_encoded(image\_path+"\_paper2\_encodes.pkl",encoded)

compressed\_size = sum(len(enc) for enc in encoded) \* 2 # Each (value, count) pair

compression\_ratio = (compressed\_size / original\_size) \* 100

space\_saving = 100 - compression\_ratio

compression\_time = (time.time() - start\_time) \* 1000 # Convert to milliseconds

return original\_size, compressed\_size, compression\_ratio, space\_saving, compression\_time

**Helper method to save the results:**

def evaluate\_bitwise\_images(image\_paths, modes=['bw', 'grayscale', 'rgb']):

results = []

for i in range(len(image\_paths)):

image\_path=image\_paths[i]

mode=modes[i//2]

orig\_size, comp\_size, comp\_ratio, space\_save, comp\_time = image\_to\_bitwise\_rle(image\_path, mode)

results.append([image\_path, mode, orig\_size, comp\_size, comp\_ratio, space\_save, comp\_time])

df = pd.DataFrame(results, columns=['Image', 'Mode', 'Original Size', 'Compressed Size', 'Compression Ratio (%)', 'Space Saving (%)', 'Compression Time (ms)'])

return df

if \_\_name\_\_ == "\_\_main\_\_":

image\_paths = ['./data/b&w1\_mod.png',

'./data/b&w2\_mod.png',

'./data/gray1\_mod.png',

'./data/gray2\_mod.png',

'./data/rgb1\_mod.png',

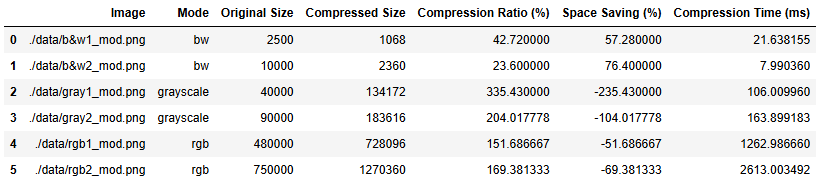
'./data/rgb2\_mod.png']

df\_results = evaluate\_bitwise\_images(image\_paths)

print(df\_results)

**Results:**

Paper 1:



Paper 2:

