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.

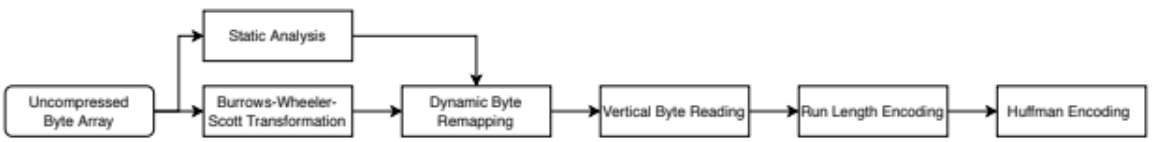
Step-3: Read the first element of input data1.

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_na
me)
image_name = "./data/gray2.png"
output_name = "./data/gray2_mod.png"
width=300
height=300
grayscale_image_process(image_name,width,height,output_na
me)
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:

	Image	Mode	Original Size	Compressed Size	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
0	./data/b&w1_mod.png	bw	2500	1068	42.720000	57.280000	21.638155
1	./data/b&w2_mod.png	bw	10000	2360	23.600000	76.400000	7.990360
2	./data/gray1_mod.png	grayscale	40000	134172	335.430000	-235.430000	106.009960
3	./data/gray2_mod.png	grayscale	90000	183616	204.017778	-104.017778	163.899183
4	./data/rgb1_mod.png	rgb	480000	728096	151.686667	-51.686667	1262.986660
5	./data/rgb2_mod.png	rgb	750000	1270360	169.381333	-69.381333	2613.003492

Paper 2:

Image	Mode	Original Size	Compressed Size	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
./data/b&w1_mod.png	bw	2500	1068	42.720000	57.280000	0.000000
./data/b&w2_mod.png	bw	10000	2360	23.600000	76.400000	15.625000
./data/gray1_mod.png	grayscale	40000	134172	335.430000	-235.430000	109.400749
./data/gray2_mod.png	grayscale	90000	183616	204.017778	-104.017778	124.977589
./data/rgb1_mod.png	rgb	480000	728096	151.686667	-51.686667	2919.356108
./data/rgb2_mod.png	rgb	750000	1270360	169.381333	-69.381333	4229.779243