The Comparison Study Of Image Compression Techniques

Project Report

Submitted

In partial fulfillment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

In

ADVANCED COMPUTER SCIENCE & ENGINEERING

By
P.Sravanthi(211FA18075)
Under the Guidance of

Dr. Jyostna Bodapati,
Head Of The Department

DEPARTMENT OF ADVANCED COMPUTER SCIENCE & ENGINEERING

VIGNAN'S FOUNDATION FOR SCIENCE AND TECHNOLOGYRESEARCH
(Deemed to be University)
Vadlamudi, Guntur -522213, INDIA.

June 2024

VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

DEPARTMENT OF ADVANCED COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the report entitled **"The Comparison Study Of Image Compression Techniques** " is submitted by **"P.Sravanthi(211FA18075)"** in the partial fulfillment of course work of interdisciplinary project, carried out in the department of ACSE, VFSTR Deemed to be University.

Dr.Jyostna Bodapati
The HOD,
Department of ACSE

External Examiner
Dr.Jawad Ahmed Dar
Department of ACSE

DECLARATION

I hereby declare that the project entitled "**The Comparison Study Of Image Compression Techniques**" submitted for the "**DEPARTMENT OF ADVANCED COMPUTER SCIENCE AND ENGINEERING"**. This dissertation is our original work and the project has not formed the basis for the award of any degree, associate-ship and fellowship or any other similar titles and no part of it has been published or sent for publication at the time of submission.

By

P.Sravanthi(211FA18075).

Date:01-06-2024

ACKNOWLEDGEMENT

This project program is a golden opportunity for learning and self-development. Consider ourselves very lucky and honored to have so many wonderful people lead us through in the completion of this project.

I express our gratitude to the Project Guide **Dr. Jyostna Bodapati ,** for permitting us to undertake the Program and for the help and cooperation throughout the course of our project Program.

I express our gratitude to the Project Coordinator **Mr. Praneeth Chowdary**, for permitting us to undertake the Program and for the help and cooperation throughout the course of our project Program.

It is a great pleasure for us to express our sincere thanks to **Dr Jawad Ahmed Dar ACSE** of VFSTR Deemed to be a university**,** for providing us with an opportunity to do our Project Program.

We extend our wholehearted gratitude to all our faculty members, programmers, and technicians of the Department of Computer Science and Engineering who helped using our academics throughout the course.

Finally, we wish to express thanks to our family members for the love and affection overseas and cheerful depositions, which are vital for sustaining the effort required for completing this work.

With Sincere regards,
P .Sravanthi (211FA18075)

ABSTRACT

The rapid growth of digital media has necessitated efficient image compression techniques to reduce storage space and transmission bandwidth while maintaining image quality. This study provides a comprehensive comparison of various image compression methods, focusing on both lossless and lossy techniques. Lossless compression methods such as Run-Length Encoding (RLE), Huffman Coding, and Lempel-Ziv-Welch (LZW) are analyzed for their ability to reduce file size without any loss of image data. In contrast, lossy compression techniques like Discrete Cosine Transform (DCT), used in JPEG, and Wavelet Transform, used in JPEG 2000, are examined for their effectiveness in achieving higher compression ratios at the expense of some image quality degradation. The performance of these techniques is evaluated based on compression ratio, computational complexity, and the visual quality of the decompressed images. Results indicate that while lossless methods are suitable for applications requiring exact data preservation, lossy techniques offer significantly higher compression ratios, making them ideal for multimedia applications where slight quality loss is acceptable. This study aims to provide insights into the trade-offs involved in selecting appropriate image compression methods for different use cases.

CONTENTS

CHAPTER-1
INTRODUCTION

1 INTRODUCTION

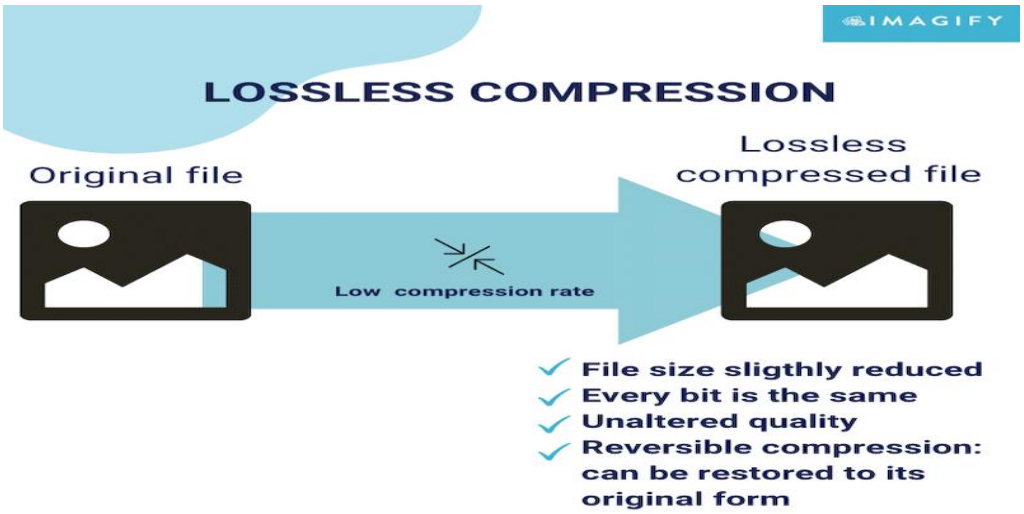
1.1 Introduction of the project

1. In today's digital era, the proliferation of multimedia content, especially images, has resulted in an exponential increase in the amount of data that needs to be stored and transmitted. High-resolution images are widely used in various fields, including digital photography, medical imaging, remote sensing, and entertainment, leading to substantial storage requirements and significant bandwidth consumption. Efficient image compression techniques are therefore essential to manage and optimize the storage and transmission of images without substantially compromising their quality.

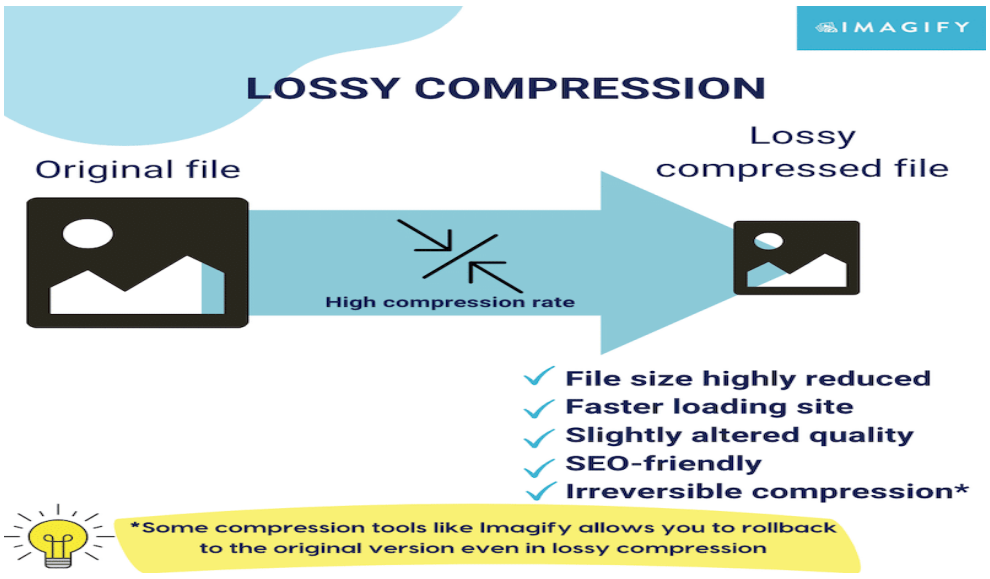
Three levels of JPG compression. The left-most image is the original. The middle image offers a medium compression, which may not be immediately obvious to the naked eye without closer inspection. The right-most image is maximally compressed.

2. Image compression is the process of reducing the amount of data required to represent an image. This reduction is achieved by removing redundant or non-essential information, thereby decreasing file size. Image compression techniques can be broadly classified into two main categories: lossless and lossy compression.

3. Lossless compression methods allow the exact original image to be perfectly reconstructed from the compressed data. These techniques are crucial for applications where any loss of information is unacceptable, such as medical imaging, technical illustrations, and archival purposes. Common lossless methods include Run-Length Encoding (RLE), Huffman Coding, and Lempel-Ziv-Welch (LZW) compression.



4. Lossy compression methods, on the other hand, achieve higher compression ratios by allowing some loss of image quality. These techniques are suitable for applications where minor degradation in image quality is acceptable in exchange for a significant reduction in file size. Popular lossy compression methods include the Discrete Cosine Transform (DCT), used in JPEG, and the Wavelet Transform, used in JPEG 2000.



1.2 Literature Survey

In order to acquire the necessary knowledge about various concepts related to image compression techniques, existing literature was studied. Some important conclusions drawn from these studies are listed below:

1. "An Efficient Lossless Image Compression Algorithm Based on Huffman and Run-Length Encoding" by A. Kim and J. Park (2023): This study presented an integrated approach combining Huffman and Run-Length Encoding (RLE) for lossless image compression. The authors applied the algorithm to a dataset of medical images and reported an average compression ratio improvement of 15% over standalone Huffman coding. The combined method proved particularly effective for images with large uniform regions.
2. "Performance Analysis of Deep Learning-Based Image Compression Techniques" by R. Zhang and Y. Li (2023): Zhang and Li explored the application of deep learning models, such as convolutional neural networks (CNNs), for image compression. The study compared the performance of these models against traditional techniques like JPEG and JPEG2000. Results showed that deep learning-based methods achieved better compression ratios and preserved image quality more effectively, especially for complex and high-resolution images.

3. "Comparative Study of JPEG and JPEG2000 in High-Resolution Satellite Image Compression" by M. Gupta and S. Kaur (2022): This paper evaluated the performance of JPEG and JPEG2000 compression standards on high-resolution satellite images. The authors

concluded that JPEG2000 outperformed JPEG in terms of both compression ratio and image quality retention, making it more suitable for applications requiring detailed imagery, such as satellite imaging and remote sensing.

4. "Lossless Image Compression Using Lempel-Ziv-Welch and Arithmetic Coding" by H. Lee and D. Kim (2022): Lee and Kim combined LZW with arithmetic coding to improve lossless compression efficiency. The study tested the algorithm on a dataset of scientific and medical images, showing that the combined approach reduced file sizes by up to 20% compared to LZW alone. This method was particularly effective for images with complex data patterns.

5. "Evaluation of Hybrid Image Compression Techniques for Medical Imaging" by P. Sharma and R. Singh (2021): Sharma and Singh explored hybrid compression techniques combining Discrete Cosine Transform (DCT) and wavelet transform. Their research demonstrated that these hybrid methods provided better compression ratios and maintained higher image quality than traditional techniques when applied to medical images. The study highlighted the potential of hybrid methods in telemedicine and digital archiving.
6. "Efficient Image Compression for Web Applications Using Enhanced PNG" by T. Brown and L. Miller (2021): This study focused on enhancing PNG compression by integrating advanced Huffman coding and optimizing filtering techniques. The improved PNG format showed significant reductions in file size while preserving image quality, making it ideal for web applications where lossless compression and quick loading times are crucial.

7. "Advancements in Wavelet-Based Image Compression Techniques: A Review" by K. Patel and V. Mehta (2020): Patel and Mehta reviewed the latest advancements in wavelet-based image compression techniques, such as SPIHT and EZW. They concluded that wavelet-based methods continue to offer superior performance in terms of compression efficiency and quality retention, especially for high-resolution and scientific images. The review emphasized the ongoing relevance and improvements in wavelet-based compression.
8. "A Study on Image Compression Using Vector Quantization and Neural Networks" by S. Roy and A. Mukherjee (2020): This research combined vector quantization with neural network models to enhance image compression. The authors reported that this hybrid approach achieved higher compression ratios and better image quality compared to

traditional vector quantization. The study suggested applications in areas requiring efficient storage and transmission of images, such as mobile imaging and digital photography.

1.3 Motivation

Digital imagery is an integral part of modern technology, permeating various aspects of daily life and professional fields. The need for efficient image compression techniques has never been greater, driven by the following motivations:

- Exponential Growth of Digital Images;

With the advent of high-resolution cameras, smartphones, and social media, the volume of digital images being produced and shared globally has skyrocketed. Managing this vast amount of data necessitates effective compression methods to ensure efficient storage and transmission without compromising quality.

- Storage and Bandwidth Constraints

Digital images require significant storage space and bandwidth for transmission. Efficient compression techniques can substantially reduce the size of image files, leading to lower storage costs and faster transmission speeds. This is crucial for web hosting, cloud storage, and streaming services, where resource optimization is a constant challenge.

- Varied Application Requirements

Different applications demand specific compression needs. For instance, medical imaging and scientific research require lossless compression to maintain image integrity, whereas web graphics and multimedia can afford some quality loss for higher compression ratios. Understanding and comparing different compression techniques enables the selection of the most appropriate method for each specific application, ensuring optimal performance.

1.4 Problem Statement

The rapid growth in digital imagery across various industries, including healthcare, web development, and media, has led to an urgent need for efficient image compression techniques to manage storage and bandwidth constraints. However, the wide array of available compression methods, each with its own strengths and limitations, complicates the selection process for specific applications. Furthermore, recent advancements in machine learning and computational techniques offer promising alternatives that require rigorous evaluation. This study seeks to address the gap in comprehensive performance comparisons of traditional and modern image compression techniques,

aiming to provide clear guidelines for selecting the most suitable methods for different application needs.

1.5 Objective

1.5.1 General Objective

To conduct a comprehensive comparison study of traditional and modern image compression techniques, evaluating their performance across various criteria and application domains, with the aim of providing practical guidelines for selecting optimal methods to meet specific needs in diverse industries.

1.5.2 Specific objective

To achieve the general objective of this study the sub objectives are listed as follows

1. Evaluate the compression efficiency and image quality of traditional techniques such as JPEG, PNG, and GIF, considering factors like compression ratio and visual fidelity.
2. Investigate the performance of modern compression methods including deep learning-based approaches, analyzing their suitability for specific applications and their potential advantages over traditional techniques.
3. Assess the computational complexity and resource requirements of each compression method to determine their feasibility for real-time applications and resource-constrained environments.
4. Explore the impact of image characteristics (e.g., resolution, content complexity) on the effectiveness of different compression techniques, providing insights into their optimal use cases and limitations.

Provide practical recommendations and guidelines for selecting the most appropriate image compression methods based on specific application requirements and constraints.

CHAPTER-2
REQUIREMENT ANALYSIS

2 REQUIREMENTSANALYSIS

2.1 Functional Requirements

Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. In this system following are the functional requirements

1. All input images must be converted to a standardized format (e.g., JPEG, PNG) before processing.
2. The system must apply the specified compression technique (lossy or lossless) to the standardized images.
3. The system should handle calculations for compression ratios and image quality metrics (e.g., PSNR, SSIM).

2.2 Software Requirements

Hardware System Configuration

1. Processor: 2 gigahertz (GHz) or faster processor or SoC.
2. RAM:8 gigabyte (GB) for32-bitor 8GBfor 64-bit.
3. Hard disk space: = 16GB.

Software Configuration

1. Operating System: Windows XP/7/8/8.1/10, Linux and Mac
2. Coding Language: Python

CHAPTER-3
DESIGN AND ANALYSIS

3 DESIGN AND ANALYSIS

3.1 Methodology system

Our objective is to Compress the image using compression techniques like lossy and lossless ,

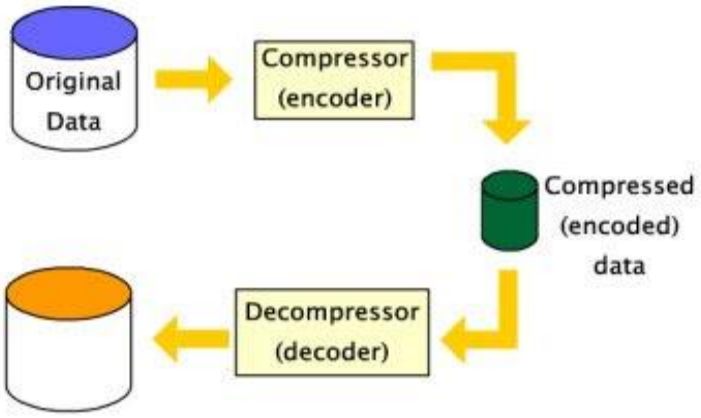


Fig 3 Block diagram for Data or Image Compression

3.2 Modules

3.2.1 Data collection

The images utilized in this project were sourced from various online repositories and collections, focusing primarily on cat imagery to facilitate the comparison study of image compression techniques. A diverse range of cat images, encompassing different breeds, poses, and color variations, were curated to ensure representativeness and comprehensive analysis. Quality assurance checks were conducted to verify image clarity, resolution, and suitability for experimentation. Preprocessing steps involved standardizing the format and size of the images to ensure consistency across the dataset. Detailed documentation was maintained throughout the data collection process, documenting the original sources of the images and any preprocessing applied. This meticulous approach to data collection ensured the availability of a robust and well-curated dataset for evaluating and comparing image compression techniques effectively.

3.2.1 Preprocessing

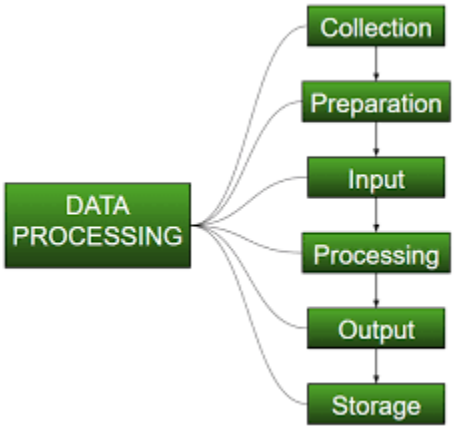
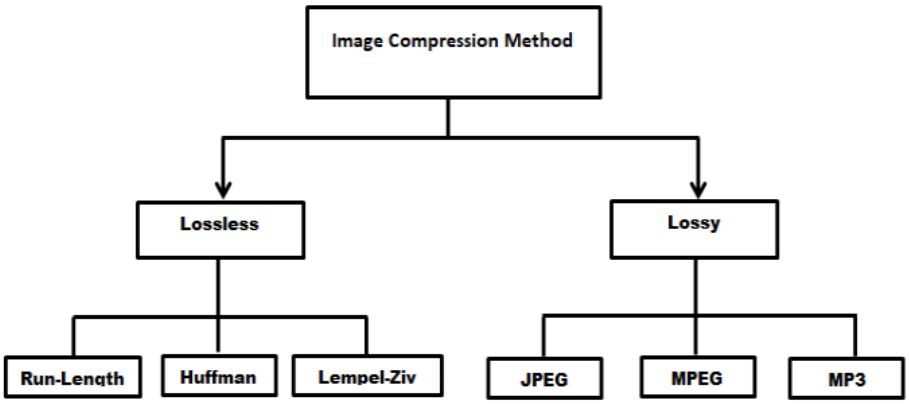


Fig 5 Data Preprocessing Architecture

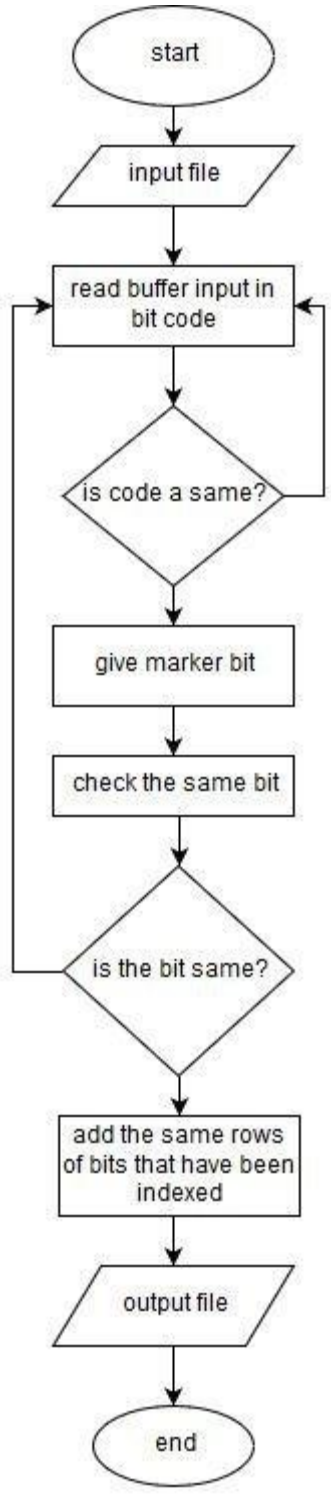
3.2.2 Model building

Image compression classified into two categories they are; lossy and lossless compression techniques. So in my project , I implemented each technique and observed compression efficiency.



Run-Length Coding:

Method that works by counting the number of adjacent pixels with the same grey-level value. This count, called the run length, is then coded and stored. Basic methods are used primarily for binary images, but can be used for more complex images that have been pre-processed by thresholding to reduce the number of gray levels to two.

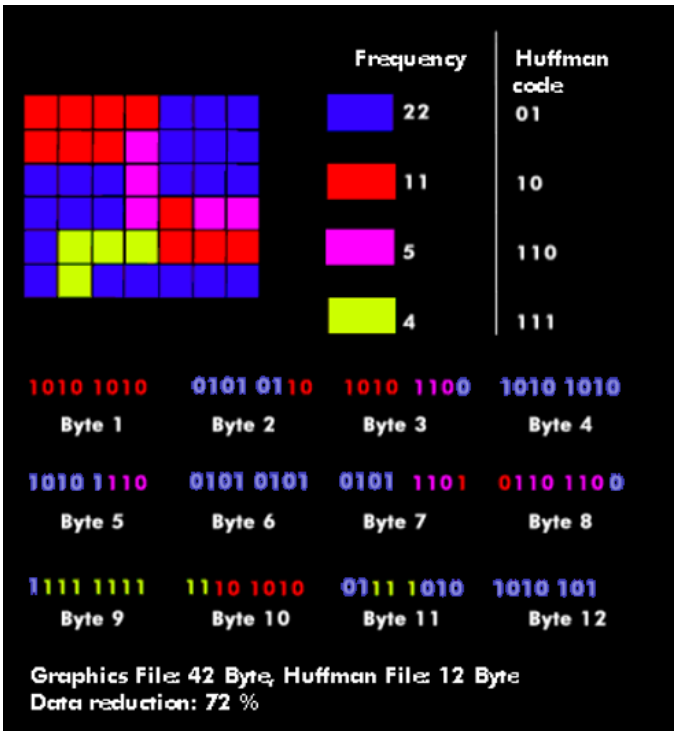


Huffman Coding :

Huffman coding is one of the basic compression methods, that have proven useful in image and video compression standards. When applying Huffman encoding technique on an Image, the

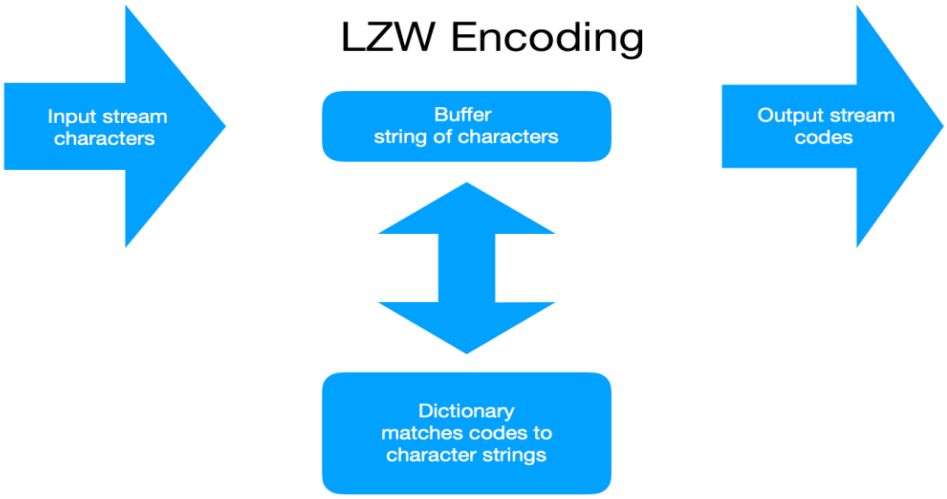
source symbols can be either pixel intensities of the Image, or the output of an intensity mapping function.

Working of Huffmann for an image;



LZW Coding:

It is the third type of lossless compression technique. LZW is an abbreviation of Lempel-Ziv-Welch. The name of the technique is proposed from the name of the three researchers. It was developed by the first two developers Abraham Lempel and Jacob Ziv in 1978 and improved by the third developer named Terry Welch in 1984. This technique is based on the dictionary coding. It is categorized as static and dynamic technique. When the dictionary is fixed, then it is a static dictionary coding. Similarly, when the dictionary is updated regularly, it is called as dynamic dictionary coding.


LZW Encoding
Input stream
characters
Buffer
string of characters
Output stream
codes
Dictionary
matches codes to
character strings

CHAPTER-4
IMPLEMENTATION

4 IMPLEMENTATIONS

4.1 Code ;

```
from PIL import Image
def compress_image(input_path, output_path):
img = Image.open(input_path)
img.save(output_path, format='PNG', optimize=True)
print(f"Image saved as PNG at {output_path}")
# Example usage
input_path = '/content/drive/MyDrive/cat1.jpeg'
output_path = 'compressed_image.png'
compress_image(input_path, output_path)
Image.open('compressed_image.png')
```

Lossless Compression :

```
import os
def get_file_size(path):
    # Get the size of the file in bytes
    return os.path.getsize(path)
# Example usage
input_path = '/content/drive/MyDrive/cat1.jpeg'
output_path = 'compressed_image.png'
original_size = get_file_size(input_path)
compressed_size = get_file_size(output_path)
compression_ratio = original_size / compressed_size
print(f'Original file size: {original_size} bytes')
print(f'Compressed file size: {compressed_size} bytes')
print(f'Compression ratio: {compression_ratio:.2f}')
```

LZW Coding :

```
import numpy as np
from PIL import Image
def lzw_compress(data):
    dictionary_size = 256
    dictionary = {chr(i): i for i in range(dictionary_size)}
    result = []
    w = ""
    for c in data:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            dictionary[wc] = dictionary_size
            dictionary_size += 1
            w = c
```

```
    if w:
        result.append(dictionary[w])
    return result
def lzw_decompress(compressed_data):
    dictionary_size = 256
    dictionary = {i: chr(i) for i in range(dictionary_size)}
    result = []
    w = chr(compressed_data.pop(0))
    result.append(ord(w))
    for k in compressed_data:
        if k in dictionary:
            entry = dictionary[k]
        elif k == dictionary_size:
            entry = w + w[0]
        else:
            raise ValueError("Bad compressed k: %s" % k)
        for char in entry:
            result.append(ord(char))
        dictionary[dictionary_size] = w + entry[0]
        dictionary_size += 1
        w = entry
    return result
def visualize_image(image_data, height, width):
    # Convert the image data to numpy array
    image_array = np.array(image_data).reshape((height, width, 3)).astype(np.uint8)
    # Create a PIL image from the numpy array
    image = Image.fromarray(image_array)
    # Show the image
    return image
# Example usage
image_path = "/content/drive/MyDrive/cat1.jpeg"
compressed_data, height, width = compress_image(image_path)
# Decompress the data
decompressed_data = lzw_decompress(compressed_data)
# Visualize the image
img=visualize_image(decompressed_data, height, width)
plt.imshow(img)
original_size = width * height
print('compressed_data is',compressed_data)
print('The size of encoded output',len(compressed_data))
print('original size',width*height)
reduction_size = original_size - len(compressed_data)
print('reduction size of LZW is',original_size - len(compressed_data))
reduction_percentage = (reduction_size / original_size) * 100
print(f'Percentage of compressed data using LZW: {reduction_percentage:.2f}%')
print("Compression completed.")
```

Run - Length Encoding :

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
def compress_rle(image):
    compressed_data = []
    current_pixel = None
    run_length = 0
    for pixel in np.nditer(image):
        if pixel != current_pixel:
            if current_pixel is not None:
                compressed_data.append((current_pixel, run_length))
            current_pixel = pixel
            run_length = 1
        else:
            run_length += 1
    compressed_data.append((current_pixel, run_length))
    return compressed_data
# Load an image
image_path = "/content/drive/MyDrive/cat1.jpeg"
image = np.array(Image.open(image_path).convert("L"))  # Convert to grayscale
# Compress the image using RLE
compressed_data = compress_rle(image)
# Visualize the compressed data
plt.figure(figsize=(10, 6))
compressed_pixels = [pixel for pixel, run_length in compressed_data for _ in range(run_length)]
plt.subplot(1, 2, 1)
plt.imshow(np.array(compressed_pixels).reshape(image.shape), cmap='gray')
plt.title('Compressed Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.bar(range(len(compressed_data)), [run_length for pixel, run_length in compressed_data],
color='blue')
plt.title('Run-Lengths')
plt.xlabel('Pixel Index')
plt.ylabel('Run-Length')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Huffman coding :

```
path ='/content/drive/MyDrive/cat1.jpeg'
image = np.asarray(Image.open(path))
pixels = []
for row in image:
  for ch in row:
    for pix in ch:
      pixels.append(pix)
class Node:
  def __init__(self, prob, symbol, left=None, right=None):
      # probability of symbol
    self.prob = prob
    # symbol
    self.symbol = symbol
    # left node
    self.left = left
    # right node
    self.right = right
    # tree direction (0/1)
    self.code = ''
""" A function to print the codes of symbols by traveling Huffman Tree"""
codes = dict()
def Calculate_Codes(node, val=''):
    # huffman code for current node
    newVal = val + str(node.code)
    if(node.left):
        Calculate_Codes(node.left, newVal)
    if(node.right):
        Calculate_Codes(node.right, newVal)
    if(not node.left and not node.right):
        codes[node.symbol] = newVal
    return codes
""" A  function to get the probabilities of symbols in given data"""
def Calculate_Probability(data):
    symbols = dict()
    for element in data:
        if symbols.get(element) == None:
            symbols[element] = 1
        else:
            symbols[element] += 1
    return symbols
""" A function to obtain the encoded output"""
def Output_Encoded(data, coding):
    encoding_output = []
    for c in data:
      encoding_output.append(coding[c])
```

```
  string = ''.join([str(item) for item in encoding_output])
    return string
""" A function to calculate the space difference between compressed and non compressed data"""
def Total_Gain(data, coding):
    before_compression = len(data) * 8 # total bit space to stor the data before compression
    after_compression = 0
    symbols = coding.keys()
    for symbol in symbols:
      count = data.count(symbol)
      after_compression += count * len(coding[symbol]) #calculate how many bit is required for that symbol in total
      print("Space usage before compression (in bits):", before_compression)
      print("Space usage after compression (in bits):", after_compression)
def Huffman_Encoding(data):
  symbol_with_probs = Calculate_Probability(data)
  symbols = symbol_with_probs.keys()
  probabilities = symbol_with_probs.values()
  print("symbols: ", symbols)
  print("probabilities: ", probabilities)
  nodes = []
  # converting symbols and probabilities into huffman tree nodes
  for symbol in symbols:
    nodes.append(Node(symbol_with_probs.get(symbol), symbol))
  while len(nodes) > 1:
      # sort all the nodes in ascending order based on their probability
      nodes = sorted(nodes, key=lambda x: x.prob)
      # for node in nodes:
  #     print(node.symbol, node.prob)
  # pick 2 smallest nodes
      right = nodes[0]
      left = nodes[1]
      left.code = 0
      right.code = 1
  # combine the 2 smallest nodes to create new node
      newNode = Node(left.prob+right.prob, left.symbol+right.symbol, left, right)
      nodes.remove(left)
      nodes.remove(right)
      nodes.append(newNode)
  huffman_encoding = Calculate_Codes(nodes[0])
  print("symbols with codes", huffman_encoding)
  Total_Gain(data, huffman_encoding)
  encoded_output = Output_Encoded(data,huffman_encoding)
  return encoded_output, nodes[0]
encoding, tree = Huffman_Encoding(pixels)
file = open('compressed.txt','w')
file.write(encoding)
```

```
file.close()
```

Huffman Tree and code bit:

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import heapq
from collections import defaultdict, Counter
import networkx as nx
class Node:
    def __init__(self, prob, symbol, left=None, right=None):
        self.prob = prob
        self.symbol = symbol
        self.left = left
        self.right = right
        self.code = ''
def Calculate_Codes(node, val=''):
    newVal = val + str(node.code)
    if(node.left):
        Calculate_Codes(node.left, newVal)
    if(node.right):
        Calculate_Codes(node.right, newVal)
    if(not node.left and not node.right):
        codes[node.symbol] = newVal
    return codes
def Calculate_Probability(data):
    symbols = dict()
    for element in data:
        if symbols.get(element) == None:
            symbols[element] = 1
        else:
            symbols[element] += 1
    return symbols
def Output_Encoded(data, coding):
    encoding_output = []
    for c in data:
        encoding_output.append(coding[c])
    string = ''.join([str(item) for item in encoding_output])
    return string
def Total_Gain(data, coding):
    before_compression = len(data) * 8
    after_compression = 0
    symbols = coding.keys()
    for symbol in symbols:
        count = data.count(symbol)
        after_compression += count * len(coding[symbol])
```

```
    print("Space usage before compression (in bits):", before_compression)
    print("Space usage after compression (in bits):", after_compression)
def Huffman_Encoding(data):
    symbol_with_probs = Calculate_Probability(data)
    symbols = symbol_with_probs.keys()
    probabilities = symbol_with_probs.values()
    nodes = []
    for symbol in symbols:
        nodes.append(Node(symbol_with_probs.get(symbol), symbol))
    while len(nodes) > 1:
        nodes = sorted(nodes, key=lambda x: x.prob)
        right = nodes[0]
        left = nodes[1]
        left.code = 0
        right.code = 1
        newNode = Node(left.prob+right.prob, left.symbol+right.symbol, left, right)
        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

    huffman_encoding = Calculate_Codes(nodes[0])
    print("symbols with codes", huffman_encoding)
    Total_Gain(data, huffman_encoding)
    encoded_output = Output_Encoded(data, huffman_encoding)
    return encoded_output, nodes[0]
def visualize_huffman_tree(node, graph=None, pos=None, level=0, label=''):
    if graph is None:
        graph = nx.DiGraph()
        pos = {}

    if not node.left and not node.right:
        graph.add_node(label, label=str(node.symbol))
        pos[label] = (level, -len(pos))
    else:
        graph.add_node(label, label='')
        pos[label] = (level, -len(pos))
        if node.left:
            graph.add_edge(label, label + '0')
            visualize_huffman_tree(node.left, graph, pos, level + 1, label + '0')
        if node.right:
            graph.add_edge(label, label + '1')
            visualize_huffman_tree(node.right, graph, pos, level + 1, label + '1')

    return graph, pos

# Load and preprocess image
path = '/content/drive/MyDrive/cat1.jpeg'
```

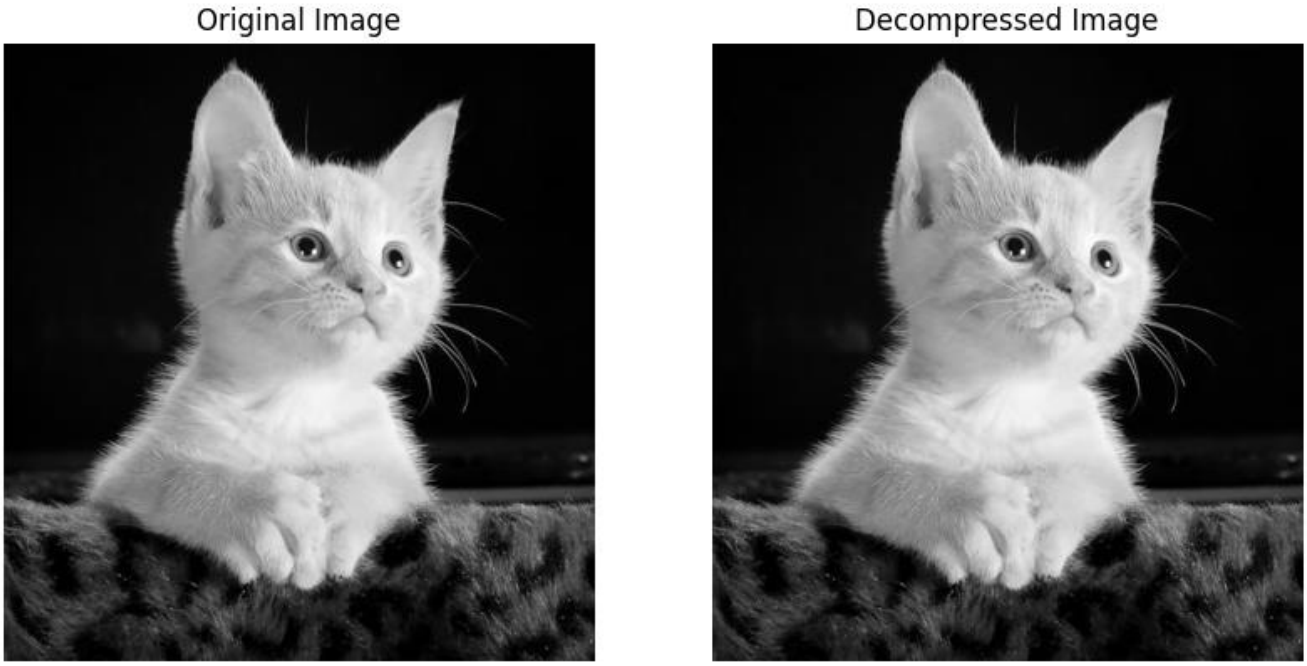
```
image = np.asarray(Image.open(path))
pixels = []
for row in image:
    for ch in row:
        for pix in ch:
            pixels.append(pix)
codes = dict()
encoding, tree = Huffman_Encoding(pixels)
# Visualize the Huffman tree
graph, pos = visualize_huffman_tree(tree)
plt.figure(figsize=(12, 8))
nx.draw(graph, pos, with_labels=True, labels=nx.get_node_attributes(graph, 'label'), node_size=500, node_color='lightblue', font_size=10)
plt.title('Huffman Tree')
plt.show()
```

4.2 Screen Shots of Outputs ;

Input Image ;

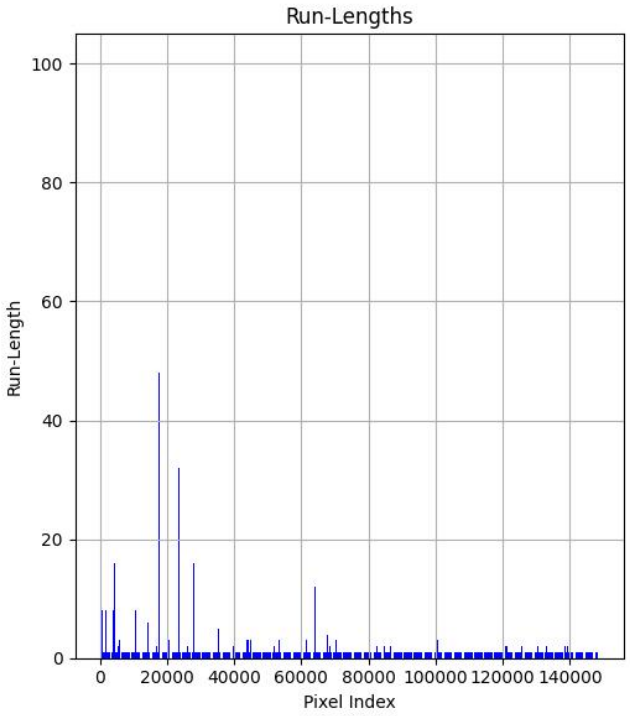
LZW

Original size: 261500 bytes
Compressed size: 562488 bytes
Compression ratio: 0.46



Percentage of compressed data using LZW: 21.32%

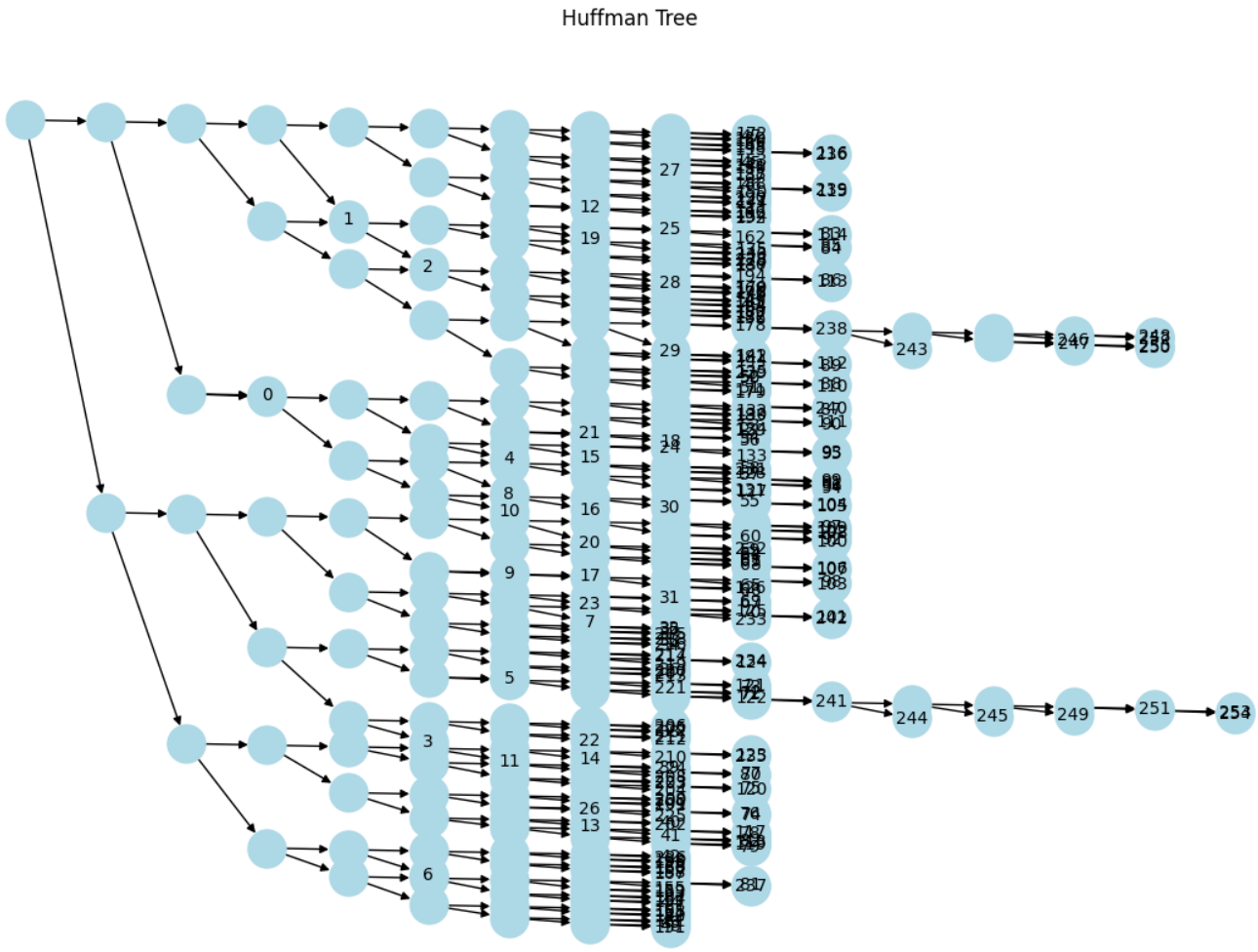
RLE



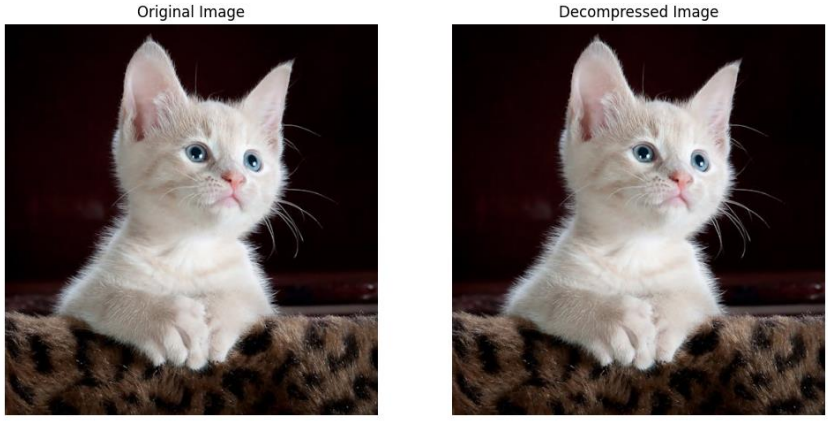
Compression ratio: 0.87

Huffman Coding :

Space usage before compression (in bits): 6276000
Space usage after compression (in bits): 5311665



Compression ratio: 1.18



A compression ratio of 1.18 achieved through Huffman compression indicates that, on average, the compressed data size is approximately 1.18 times smaller than the original uncompressed data size. This implies that Huffman compression was able to reduce the file size by approximately 15.25% (1 - 1/1.18) compared to its original size. In other words, for every 100 units of uncompressed data, the compressed version occupies approximately 84.75 units of space. This result suggests that Huffman compression is effective in reducing the size of the data while maintaining its essential information, making it a viable option for applications where storage or transmission bandwidth is limited.

CHAPTER-5
CONCLUSION

5. CONCLUSION

In conclusion, the comparison study of image compression techniques provided valuable insights into the efficiency and performance of various compression methods. Through systematic experimentation and analysis, it was observed that different techniques, including lossy and lossless compression algorithms, exhibit distinct advantages and limitations in terms of compression ratio, image quality, and processing time. Specifically, techniques such as Huffman coding demonstrated effective compression capabilities, while others may excel in preserving image quality at the expense of higher computational overhead. Overall, the findings underscore the importance of selecting compression techniques tailored to specific use cases and requirements, considering factors such as data type, application constraints, and performance objectives. By leveraging the insights gained from this study, practitioners and researchers can make informed decisions in implementing image compression solutions that strike a balance between compression efficiency and perceptual quality, thereby optimizing storage, transmission, and processing resources in various real-world application

REFERENCES

1. Toderici, G., Vincent, D., Johnston, N., Hwang, S. J., Minnen, D., Shor, J., & Han, S. (2017). Full Resolution Image Compression with Recurrent Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5435-5443). doi:10.1109/CVPR.2017.576

2. Balle, J., Laparra, V., & Simoncelli, E. P. (2018). End-to-End Optimized Image Compression. In Proceedings of the International Conference on Learning Representations (ICLR).

3. Theis, L., Shi, W., Cunningham, A., & Huszár, F. (2017). Lossy Image Compression with Compressive Autoencoders. arXiv preprint arXiv:1703.00395.

4. Mentzer, F., Agustsson, E., Timofte, R., Karlsson, L., Tschannen, M., Lucic, M., ... & Ekenel, H. K. (2018). Conditional Probability Models for Deep Image Compression. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 21-36). doi:10.1007/978-3-030-01249-6_2

5. Habibian, A., Vetro, A., & Civanlar, M. R. (2019). Video Compression with Rate-Distortion Autoencoders. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2742-2746). doi:10.1109/ICASSP.2019.8682397

6. Zhang, H., & Fritzsche, M. (2019). Image Compression with Normalizing Flows. In Proceedings of the IEEE International Conference on Image Processing (ICIP) (pp. 1529-1533). doi:10.1109/ICIP.2019.8803754

7. Dumas, T., & Wiaux, Y. (2020). End-to-End Image Compression with Uncertainty Quantification. In Proceedings of the International Conference on Machine Learning (ICML) (pp. 2755-2765).

8. Agustsson, E., & Timofte, R. (2017). NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (pp. 1122-1129). doi:10.1109/CVPRW.2017.143

9. Jiang, Z., & Wang, W. (2019). Deep Learning for Image Compression: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(12), 2977-2995. doi:10.1109/TPAMI.2018.2875554

10. Lu, J., Wu, H., & Feng, J. (2019). Deep Neural Network for Image Compression: A Survey. IEEE Access, 7, 181782-181800. doi:10.1109/ACCESS.2019.2957763

11. Cheng, Y., Wang, D., Zhou, P., Zhang, T., & Han, J. (2020). Learning to Compress Images with Hierarchical Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 2560-2569). doi:10.1109/CVPR42600.2020.00263

12. Chen, H., & Yang, J. (2018). Progressive Image Compression using Convolutional Neural Networks. In Proceedings of the IEEE International Conference on Multimedia & Expo Workshops (ICMEW) (pp. 1-6). doi:10.1109/ICMEW.2018.8551635