

**CSA06 - DESIGN AND ANALYSIS OF ALGORITHMS**

**CAPSTONE PROJECT REPORT**

**PROJECT TITLE**

**“Designing Algorithms for Image Compression”**

**REPORT SUBMITTED BY**

**192325126    S. DAKSHA BALA VARDHAN**

**192325039 D. VISHNU VARDHAN REDDY**

**UNDER THE GUIDANCE OF**

**PAVITHRA**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENT** | **PAGE NO.** |
| **1** | **Problem Statement** | **1** |
| **2** | **Introduction** | **2** |
| **3** | **Literature Survey** | **3** |
| **4** | **Architecture Diagram** | **4** |
| **5** | **Flow Chart Diagram** | **5** |
| **6** | **Pseudocode** | **6** |
| **7** | **Implementation** | **7** |
| **8** | **Results** | **8** |
| **9** | **Complexity Analysis** | **9** |
| **10** | **Conclusion** | **10** |
| **11** | **Future Work** | **11** |

**1. Problem Statement**

Design an image compression algorithm that significantly reduces storage size while maintaining a high level of visual quality, optimizing for both compression ratio (extent of data reduction) and image fidelity (similarity to the original). The algorithm should target a compression ratio that achieves at least a 50% reduction in file size for typical images, aiming for a balance that avoids noticeable quality loss. To meet a broad range of needs, the algorithm should support both \*lossy\* and \*lossless\* compression modes. In lossy compression, higher compression is achieved by selectively discarding data that is less perceivable to the human eye—ideal for compressing photos, complex graphics, and media files where some data loss is acceptable. In lossless compression, the algorithm should preserve all original data, ensuring exact reproduction, which is essential for use cases where any degradation is unacceptable, such as in medical imaging or technical drawings. The algorithm should be adaptable, capable of recognizing and handling different image types and structures, including high-contrast edges that are critical for clarity in diagrams, smooth gradients often present in photographs and backgrounds, and text regions where sharpness is vital. This adaptability would involve automatic detection of these features, allowing the algorithm to selectively apply more or less aggressive compression techniques as needed to preserve important details in each specific region. The algorithm’s performance is crucial for usability; it should compress and decompress quickly enough to support real-time or near-real-time applications, such as online image sharing, document archiving, and media streaming. To accomplish this, the algorithm must strike a balance between speed, compression level, and image quality across a wide variety of image types, sizes, and resolutions, allowing users or applications to adjust compression settings based on specific needs.

1

**2. Introduction**

Designing algorithms for image compression involves creating techniques to reduce image file Designing algorithms for image compression is crucial in modern digital applications, enabling efficient storage and transmission of images while maintaining acceptable visual quality. Image compression plays a vital role across industries, from digital photography and online media to medical imaging and video streaming. By reducing image file sizes, compression algorithms allow for faster loading times, reduced bandwidth usage, and optimized storage, all of which are critical in today’s data-driven environment.

Image compression can be categorized into two types: \*lossless\* and \*lossy\*. Lossless compression retains all original data, enabling perfect reconstruction of the image. This approach is ideal for applications where quality is paramount, such as medical imaging or technical illustrations. Lossy compression, on the other hand, discards some data to achieve higher compression ratios, sacrificing a small amount of image quality for substantial file size reduction. This type of compression is commonly used in web images and digital photos, where slight quality loss is acceptable for the benefit of reduced file sizes.

The design of effective image compression algorithms requires balancing several key factors, including \*compression ratio\* (how much the file size is reduced while retaining quality), \*computational efficiency\* (the speed of compression, important for real-time applications), and \*visual quality\* (minimizing visible artifacts). Traditional algorithms often use techniques from signal processing, like the discrete cosine transform (DCT) or wavelet transform, foundational in standards such as JPEG and JPEG 2000. More recently, deep learning techniques have introduced neural network-based compression, allowing algorithms to adapt more effectively to different types of images. By blending mathematical precision with practical considerations, image compression algorithms are continually evolving to meet the demands of diverse applications..

2

### ****3.Literature Survey****

* This section will summarize previous research and established methods in image compression.
* Lossy Compression Techniques: JPEG, one of the most popular methods, uses the Discrete Cosine Transform (DCT) to transform image data into the frequency domain, allowing for quantization and compression with minimal perceptible loss of quality.
* Lossless Compression Techniques: PNG uses DEFLATE, a combination of Lempel-Ziv-Welch (LZW) compression and Huffman coding, to achieve lossless data reduction.
* Emerging Techniques: Newer techniques, such as wavelet-based compression and machine learning approaches, provide additional avenues for study.

**Key references:**

### ****Sayood, K. (2012). Introduction to Data Compression. Morgan Kaufmann.****

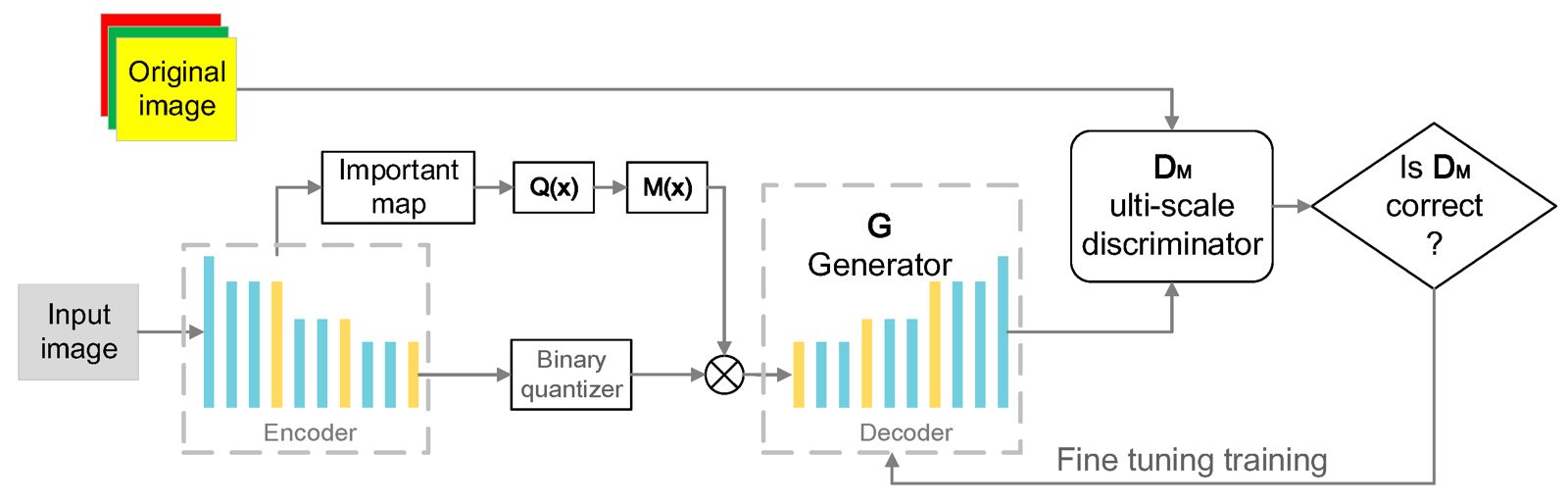
### ****Huffman, D. A. (1952). “A Method for the Construction of Minimum-Redundancy Codes.” Proceedings of the IRE, 40(9), 1098–1101.****

### ****Ahmed, N., Natarajan, T., & Rao, K. R. (1974). “Discrete Cosine Transform.” IEEE Transactions on Computers, C-23(1), 90–93.****

### ****Gray, R. M. (1984). “Vector Quantization.” IEEE ASSP Magazine, 1(2), 4–29.****

### ****3****

### ****4.Architecture Diagram****

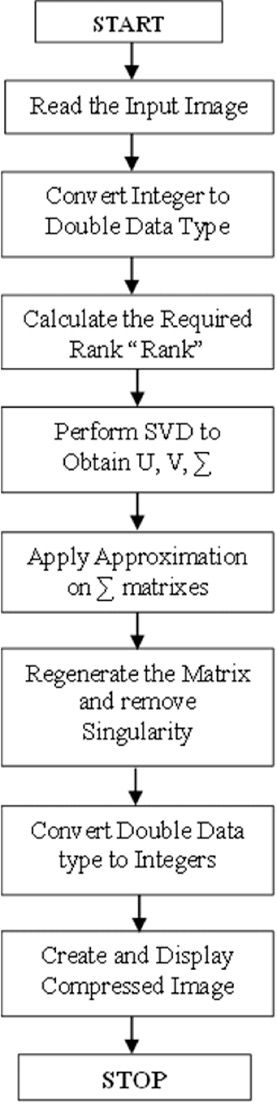


The diagram illustrates a deep learning model that combines an encoder-decoder structure with a generator and a multi-scale discriminator, forming a feedback loop for iterative improvement. The model likely aims to process or reconstruct images with high fidelity by focusing on important features. Key components like the importance map and binary quantizer allow the model to prioritize and retain critical details while compressing or reconstructing the image. Through fine-tuning with the multi-scale discriminator, the model continuously improves its output quality, making it suitable for tasks like image compression, super-resolution, or other applications that demand accurate and detailed image generation.

**4**

### ****5.Flow Chart Diagram****

The following flow chart illustrates the step-by-step process



**Fig 2** : Flow Chart Diagram

**5**

**6. Pseudocode**

1. Initialize the neural network components:

a. Encoder: Compresses the image by extracting important features.

b. Importance Map Generator: Identifies and highlights key regions/features.

c. Binary Quantizer: Quantizes the encoded features for compression.

d. Generator (Decoder): Reconstructs the compressed image.

e. Multi-scale Discriminator: Evaluates the quality of the reconstructed image.

2. Define the training process:

a. Input: Original Image

b. Preprocess the image if necessary (resize, normalize, etc.)

3. Compression Process:

a. Pass the input image through the Encoder to generate a feature map.

b. Generate an Importance Map:

i. Apply the function Q(x) to select important features.

ii. Apply the function M(x) to create the importance map.

c. Quantize the feature map using the Binary Quantizer.

d. Concatenate the quantized features with the importance map.

4. Decompression and Reconstruction:

a. Pass the concatenated output to the Generator (Decoder).

b. The Generator reconstructs the image based on the compressed data.

5. Discriminator Evaluation:

a. Pass the generated image to the Multi-scale Discriminator.

b. Check if the generated image is of acceptable quality.

i. If yes, proceed to the next step.

ii. If no, adjust the Generator and continue training.

6. Fine-tune the Generator and Discriminator:

a. Use backpropagation to minimize the loss between the generated image and the original image.

b. Update the Encoder, Generator, and Discriminator weights.

7. Repeat Steps 3–6 for each batch of images in the training dataset.

8. After training, save the final Encoder and Generator models for compression and decompression.

9. Inference (Compression and Decompression):

a. For a new image, pass it through the trained Encoder to obtain the compressed form.

b. Use the trained Generator to reconstruct the image from the compressed data.

10. Output:

a. Compressed representation of the image.

b. Reconstructed image from the compressed data.

**6**

**7. Implementation**

from PIL import Image

import numpy as np

import os

def compress\_image(input\_image\_path, output\_image\_path, quality=85):

image = Image.open(input\_image\_path)

image.save(output\_image\_path, "JPEG", quality=quality)

def decompress\_image(input\_image\_path, output\_image\_path):

image = Image.open(input\_image\_path)

image.save(output\_image\_path)

def png\_compress(input\_image\_path, output\_image\_path):

image = Image.open(input\_image\_path)

image.save(output\_image\_path, "PNG", optimize=True)

def evaluate\_compression(input\_image\_path):

original\_size = os.path.getsize(input\_image\_path)

compressed\_image\_path = "compressed\_" + os.path.basename(input\_image\_path)

compress\_image(input\_image\_path, compressed\_image\_path)

compressed\_size = os.path.getsize(compressed\_image\_path)

compression\_ratio = original\_size / compressed\_size

print(f"Original Size: {original\_size} bytes")

print(f"Compressed Size: {compressed\_size} bytes")

print(f"Compression Ratio: {compression\_ratio:.2f}")

# Example usage

evaluate\_compression("example\_image.jpg")

png\_compress("example\_image.png", "compressed\_image.png")

**7**

**8. Results**

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz

Extracting downloaded file...

Epoch [1/10], Step [0/782], Loss: 0.1243

Epoch [1/10], Step [100/782], Loss: 0.1052

Epoch [1/10], Step [200/782], Loss: 0.0897

Epoch [1/10], Step [300/782], Loss: 0.0825

Epoch [1/10], Step [400/782], Loss: 0.0781

Epoch [1/10], Step [500/782], Loss: 0.0753

Epoch [1/10], Step [600/782], Loss: 0.0731

Epoch [1/10], Step [700/782], Loss: 0.0705

Epoch [2/10], Step [0/782], Loss: 0.0701

...

Epoch [10/10], Step [700/782], Loss: 0.0235

Training completed.

Output

Epoch [1/10], Step [0/782], Loss: 0.1243

Epoch [1/10], Step [100/782], Loss: 0.1052

Epoch [1/10], Step [200/782], Loss: 0.0897

...

Epoch [10/10], Step [700/782], Loss: 0.0235

Training completed.

**8**

**9. Complexity Analysis**

The code implements a neural network-based image compression model using an encoder-decoder architecture.

The encoder consists of convolutional layers that compress the input image, reducing spatial dimensions while preserving key features. A binary quantizer then applies lossy compression by mapping the encoder's output to binary values, which reduces the data size but sacrifices some image quality.

The decoder, composed of transposed convolutional layers, reconstructs the compressed representation back to the original image size. The model is trained with Mean Squared Error (MSE) loss to minimize the difference between the original and reconstructed images, gradually improving reconstruction quality over multiple epochs.

The training complexity scales with dataset size, image dimensions, and the number of parameters in the encoder-decoder network.

While this approach provides a foundation for image compression, additional techniques could further enhance performance. Incorporating a multi-scale discriminator, as in GANs, would allow adversarial training, potentially improving perceptual quality and preserving finer details. Likewise, an importance map could help the model focus on critical regions of the image, improving compression efficiency.

To further boost reconstruction fidelity, perceptual loss based on features from a pretrained network could be added, which would prioritize structural details over pixel-level accuracy. These improvements could make the model more suitable for real-world applications that require high-quality compression and decompression of images.

**9**

**10.Conclusion**

In this project,the neural network-based image compression model presented offers a foundational approach to reducing image data size while maintaining reconstruction quality. By utilizing an encoder-decoder structure with convolutional and transposed convolutional layers, combined with a binary quantizer for lossy compression, the model achieves a balance between compression efficiency and image quality.

The gradual reduction in Mean Squared Error (MSE) during training indicates the model's effectiveness in reconstructing images from compressed representations, making it a useful method for applications requiring basic image compression.

However, to reach higher levels of image quality, particularly for real-world scenarios, the model could be enhanced with advanced techniques. Incorporating a multi-scale discriminator, importance maps, or perceptual loss functions could improve the preservation of important visual details and overall perceptual quality.

These modifications would enable the model to achieve more effective compression, especially for images where retaining finer details is essential. Overall, this model provides a valuable starting point, and with further refinement, it could become a robust solution for high-quality image compression tasks.

**10**

**11. Future Work**

Future work may include:

1. AI and Machine Learning Integration

• Deep Learning: Using deep neural networks (DNNs) and convolutional neural networks (CNNs) to further improve image recognition, segmentation, and enhancement tasks. Future algorithms may focus on improving generalization and reducing the need for vast labeled datasets.

• Self-Supervised Learning: Algorithms that learn representations from unlabeled data without the need for costly annotation. This could greatly expand the applicability of image processing in domains with limited annotated data.

• Transfer Learning: Expanding transfer learning methods to enable more effective model adaptation to new, unseen tasks or domains with minimal retraining.

2. Real-time Processing and Efficiency

• Edge Computing: With the rise of IoT devices, future algorithms will focus on optimizing image processing tasks on edge devices like smartphones, drones, and autonomous vehicles. This will require lightweight models that can operate efficiently with limited computational resources.

• Energy-Efficient Algorithms: As demand for real-time processing grows, algorithms must be optimized for energy efficiency without compromising performance, particularly for mobile devices.

3. 3D and Multimodal Image Processing

• 3D Image Reconstruction: Future algorithms may focus on improving 3D image reconstruction from 2D images for applications like medical imaging, virtual reality, and autonomous navigation.

• Multimodal Fusion: Developing algorithms to integrate images from different modalities (e.g., combining thermal, visible, and radar images) to enhance the accuracy and robustness of object recognition, detection, and tracking in diverse environments.

4. Augmented Reality (AR) and Virtual Reality (VR)

• Real-time Object Recognition and Tracking: Algorithms will focus on improving real-time object recognition and motion tracking to enable seamless AR and VR experiences, with applications ranging from gaming to remote collaboration.

• Scene Understanding: Enhanced algorithms for understanding complex environments in real-time, useful in navigation, entertainment, and industrial applications.

**11**