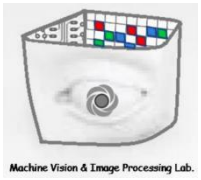


Digital Image Processing

Computer Homework 5

Due date: 1404/03/31

JPEG Compression Techniques for Image Processing

1. Instructions

a) Submit the assignment as a **Jupyter Notebook** (.ipynb) containing:
   i. Executed results of the code (no empty cells).
   ii. Explanations in Markdown cells for each step of the assignment.

b) Important Notes:
   i. Attach all necessary files (including sample images) to ensure the notebook runs directly.
   ii. Keep each part of the code in separate cells for clarity.
   iii. Add proper documentation and comments in both Markdown and code cells for a better score.
   iv. Both Python and MATLAB can be submitted, but Python submissions will be awarded 10% more points.

2. Preliminaries

JPEG (Joint Photographic Experts Group) is a widely used lossy compression standard for digital images. It reduces file size by exploiting:

- Human Visual Perception: Discards high-frequency details less noticeable to the eye.
- Transform Coding: Uses the Discrete Cosine Transform (DCT) [more info at **Appendix. A**] to convert spatial pixel data into frequency components.
- Quantization: Approximates DCT coefficients to prioritize important visual information.
- Entropy Coding: Applies lossless compression (Huffman/RLE) to further shrink data.

The process involves 8×8 block processing, color space conversion (RGB → YCbCr), and chroma subsampling to achieve high compression ratios while maintaining perceptual quality. Below is a mathematical breakdown of each step.

1. Color Space Conversion & Downsampling

➢ RGB → YCbCr Conversion:

$$Y = 0.299R + 0.587G + 0.114B$$
$$Cb = -0.1687R - 0.3313G + 0.5B + 128$$
$$Cr = 0.5R - 0.4187G - 0.0813B + 128$$

➢ Downsampling: Chroma components (Cb, Cr) are subsampled (e.g., 4:2:0) due to reduced human sensitivity to color details.

2. Discrete Cosine Transform (DCT)

➢ 8×8 Block Processing:
Each channel is divided into 8×8 blocks.

➢ 2D DCT (Transforms spatial data to frequency domain):

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

where $N = 8$, $C(k) = \begin{cases} \frac{1}{\sqrt{2}} & k = 0 \\ 1 & \text{otherwise} \end{cases}$.

- **Output**:
  - $F(0,0)$: **DC coefficient** (average intensity).
  - $F(u,v)|_{(u,v)\neq(0,0)}$: **AC coefficients** (high-frequency details).

3. Quantization (Lossy Step)

➢ Element-wise Division by quantization matrix

$$F_Q(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

➢ Quantization Matrix Design:

Larger divisors for high frequencies (e.g., bottom-right of Q) to discard imperceptible details.

Example luminance matrix:

$$Q_Y = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

4. Entropy Coding (Lossless Compression)

√ Zigzag Scanning: Converts 8×8 matrix → 1D sequence (prioritizes low-frequency coefficients).

√ Run-Length Encoding (RLE): Compresses sequences of zeros in AC coefficients.

√ Huffman Coding: Assigns variable-length codes to frequent symbols.

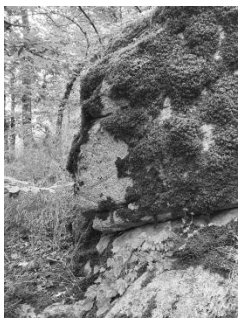
5. Decompression (Inverse Process)

1. **Inverse Entropy Coding**: Rebuild quantized coefficients.
2. **Dequantization**: $\hat{F}(u,v) = F_Q(u,v) \times Q(u,v)$.
3. **Inverse DCT (IDCT)**:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)\hat{F}(u,v) \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

4. **YCbCr → RGB Conversion** and chroma upsampling.

3. Tasks & Implementation



Task 1: Apply JPEG Compression to an Greyscale Image

➢ Step 1: Load Image

1. Load an image (Kaleybar.png)

➢ Step 2: Block Splitting

Block Splitting is a preprocessing step in JPEG compression where the image is divided into smaller blocks, typically 8×8 or 16×16 pixels. **But why is this necessary?** Consider a 256×256 image: splitting it into 1024 blocks of 8×8 pixels each significantly reduces computational complexity. The 2D-DCT has a complexity of $O(N^4)$ (or $O((N \log_2 N)^2)$ with the Fast Cosine Transform). Comparing the computations: Full image: 256^4, Fast DCT on full image: $(256 \times \log_2 256)^2$, Fast DCT on 1024 blocks: $1024 \times (8 \times \log_2 8)^2$

Clearly, processing smaller blocks is far more efficient.

1. Extract the image's height and width.
2. Split the image into 8×8 blocks (or another chosen size).
3. Store each block in an empty list for further processing (DCT, quantization, etc.).
4. The "block" parameter defines the partitioning size (e.g., 8×8).

```
block = 8
sliced = []

height = len(image)
width = len(image[0])
print(f"The image height is {height}, the image width is {width} pixels")
```

```
The image height is 600, the image width is 800 pixels
```

`

➢ Step 3: Shifting and Dividing Into Parts and

Before applying the Discrete Cosine Transform (DCT) to an 8×8 image block, we first shift pixel values from the standard unsigned range [0, 255] (for 8-bit images) to a signed range centered at zero [-128, 127].
Why Shift Values?

- The DCT works more efficiently with symmetrical data around zero.
- Shifting reduces the dynamic range of coefficients, improving compression.
- For an 8-bit image, we subtract 128 (the midpoint of [0, 255]) from each pixel.

Write a code snippet to do this job.

➢ Step 4: Discrete Cosine Transform

Write a code snippet to calculate the DCT of blocks and append all blocks into DCT output list using cv2.dct
Again, there is a computationally costly algorithm here. Write a code snippet (function) to quantize each pixel in blocks. Notice that most of the higher-frequency elements of the sub-block (i.e., those with an x or y spatial frequency greater than 4) are diminished into zero values.

➢ Step 5: Quantization

In the written code, one should be able to rearrange the Quantization matrix by changing selectQMatrix input, variable Q. Matrix options are: "Q10", "Q50" and "Q90".

```
selectedQMatrix = selectQMatrix("Q10")
```

For matrix selection you can use the QMatrix.py in this format:

```
from QMatrix import selectQMatrix
```

➢ Step 6: Inverse Discrete Cosine Transform

Using cv2.idct() function and write a code snippet to compute Inverse Discrete Cosine transform of Blocks. Do not forget to add 128 to the IDCT values. This will bring back the intensities in the range of [0 255].

➢ Step 7: Complete the Puzzle

Put the Reconstructed blocks together. Consider a list as invList. It is one dimensional list that contains 8x8 numpy arrays inside it. In order to recover the image:

- invList elements should be parsed at width/block steps. For example: If the image width is 1200 pixels, parsing steps should be 1200/8=150.
- Use np.hstack if you are going to recover the image's rows first, else use np.vstack for recover columns first.
- Append all recovered columns into another list.
- Apply np.vstack if you have been used np.hstack, reverse order with entry II.

➢ Step 8: Compare the Results Visually and Quantitatively

In this step, you will analyze the effects of different each quantization matrices on the compressed image by evaluating both visual quality and quantitative metrics.

1. Visual Comparison
   i. Display the reconstructed images after applying each quantization matrix.
   ii. Observe and describe how the visual quality changes (e.g., sharpness, artifacts, blurriness).

2. Quantitative Evaluation
   i. Calculate the file size of each compressed image after saving it in JPEG format.
   ii. Compute the Root Mean Squared Error (RMSE) between the original and the reconstructed images to measure distortion.
   iii. Summarize how the choice of quantization matrix affects the balance between compression ratio and image quality.

➢ Step 9: Entropy Coding **Zigzag + RLE** (Optional Extra Point)

In this step, you will apply a simple entropy coding technique to compress an 8×8 block of quantized DCT coefficients. This compression step is lossless and occurs after quantization but before IDCT. It can significantly reduce the size of the data with no loss of quality.

1. Zigzag Ordering

   Convert the 2D quantized block into a 1D array by scanning in a zigzag pattern, which orders coefficients from low to high frequency.

$$\begin{bmatrix} 10 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix} \Rightarrow$$

```
[10, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0]
```

2. Run-Length Encoding (RLE)

   Compress the zigzag-ordered array by encoding sequences of zeros as (0, count) pairs, while keeping non-zero values as-is..

   For example, [15, 0, 0, 0, -3, 0, 2] becomes: [(15), (0,3), (-3), (0,1), (2)]

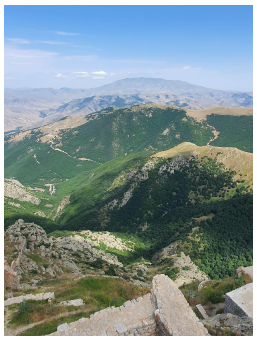
3. Save the Encoded Data and Compare the Size

Save the RLE-compressed data for all blocks into a .pkl file using Python's pickle module. This file simulates the compressed binary stream used in JPEG files. Compare this file size with original image size.

4. Reconstruct the Image After Saving .pkl

After saving the compressed data in .pkl format, students should load this file, decode the run-length encoded data to recover the zigzag-ordered coefficients, then apply the inverse zigzag scan to restore the original 2D quantized blocks. These blocks can then be inverse-quantized and passed through IDCT to reconstruct the image.

Task 2: Apply JPEG Compression to an RGB Image



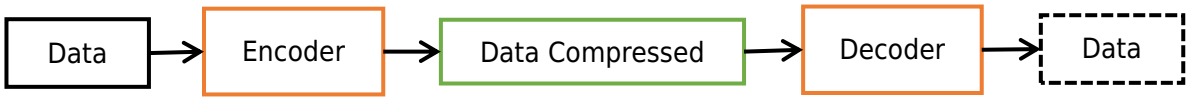
JPEG compression converts an RGB image into the YCbCr color space to separate luminance (Y) from chrominance (Cb and Cr). Because the human visual system is more sensitive to luminance changes, the chrominance channels can be compressed more aggressively without significant perceived quality loss. This improves compression efficiency and final image quality compared to compressing RGB channels directly.

- Step 1: Convert the input RGB image to YCbCr color space.
- Step 2: Apply block-wise DCT, quantization independently on the Y, Cb, and Cr channels.
- Step 3: Reconstruct each channel after decoding and inverse transforms.
- Step 4: Convert the reconstructed YCbCr image back to RGB and merge channels to obtain the final compressed image.
- Step 5: Compare the compression results with those obtained by direct RGB channel compression (Optional Extra Point)

Task 3: Image Compression Using Deep Learning (Extra Point)

In this assignment, you will compress and reconstruct images using an autoencoder, a type of neural network that learns to encode the input image into a smaller representation (compressed form) and then decode it back to the original image. Autoencoders are useful in image compression because they learn the most important features of an image and remove unnecessary details. You will train your model so that the output (reconstructed image) is as close as possible to the input image.

The schema for a compression-decompression task is presented in following Figure:



➢ Step 1: Provide Dependencies

```
!pip install tensorflow-compression
```

```
!git clone https://github.com/tensorflow/compression tfc
```

```
import tfc.models.tfci
```

➢ Step 2: Check Existing Models

```
!python tfc/models/tfci.py models
```

➢ Step 3: Select The Model

Here we choose **Factorized Prior Autoencoder**

➢ Step 4: Run The Compress Process

```
!python tfc/models/tfci.py compress bmshj2018-factorized-msssim-6 Jamalabad-Caravanserai.jpg
```

- bmshj2018-factorized-msssim : model name;
- number 6 at the end of the name indicates the quality level (1: lowest, 8: highest);
- 1.png : input file name (image).

➢ Step 5: Quantitative Evaluation

iv. Calculate the file size of compressed image after saving it in JPEG format.

v. Compute the Structural Similarity (SSIM) between the original and the reconstructed images to

➢ Step 6: Decompression Compressed Image

```
!python tfc/models/tfci.py decompress Jamalabad-Caravanserai.tfci
```

➢ Step 7: Test other Models

Best of Luck

For further information, do not hesitate to contact me at a.najafi@email.kntu.ac.ir.

Where Is Iran’s Azerbaijan Region?

Iranian Azerbaijan lies in the northwest of the country, encompassing the provinces of East Azerbaijan and West Azerbaijan. This region is a cultural and historical treasure, deeply rooted in the history of Persia, and known for its diverse landscapes, resilient people, and contributions to literature, architecture, and freedom movements. Rich in both natural beauty and heroic tales, it offers many destinations that inspire pride and curiosity.

Below are some of the must-see historical landmarks that showcase the legacy of Azerbaijani culture:

◆ **Babak Castle** (قلعه بابک) – Kaleybar, East Azerbaijan:

The mountaintop fortress of Babak Khorramdin, a legendary Iranian freedom fighter, sits over 2,300 meters above sea level. Hikers reach it through forests and rocky paths. It symbolizes bravery and resistance against oppression and is a powerful reminder of national pride.

◆ **Blue Mosque of Tabriz** (مسجد کبود) – Tabriz

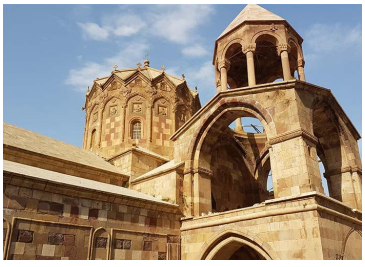
Built in 1465, this mosque is known for its magnificent blue tiles and elegant calligraphy. Though partially damaged by earthquakes, it still stands as a jewel of Islamic architecture and Safavid-era art.

◆ **Kandovan Village** (روستای کندوان) – Near Osku

Often compared to Cappadocia in Turkey, this ancient village is carved into volcanic rock. Families still live inside these cave-like structures, some over 700 years old. It's a living example of sustainable architecture and a unique lifestyle.

◆ **Girl Bridge** (پل دختر میانه) – Miyaneh

This historical bridge is located in Miyaneh, East Azerbaijan, and dates back to the Safavid era. Built over the Qezel Ozan River, it served as a crucial passage on ancient trade routes. It's an architectural symbol of the region’s historical connectivity and engineering skills.



◆ **Saint Stepanos Monastery** (کلیسای سنت استپانوس) – Jolfa

One of Iran’s most beautiful Armenian churches, built in the 9th century and surrounded by mountains. It reflects the deep coexistence of Persian and Armenian Christian heritage.

4. Appendix

A. **D**iscrete **C**osine **T**ransform (DCT): DCT expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. Therefore, an image can be represented by a linear combination of Discrete Cosine Transform basis images as illustrated in the following figure which shows 64 DCT basis.