Introduction to Assignment 4:
JPEG Image Compression and the Discrete Cosine Transform

Niclas Börlin
with thanks to Lars Karlsson

Matrix Computations and Applications

A high-level view of image compression

- Let \( M \) be an image.
- Express \( M \) as a linear combination

\[ M = \sum_{ij} \alpha_{ij} M_{ij} \]

of basis vectors (images) \( M_{ij} \) with scalar coefficients \( \alpha_{ij} \).
- Agree on the basis vectors \( M_{ij} \) (no explicit storage required).
- Approximate the coefficients \( \alpha_{ij} \).
- Store the approximate coefficients compactly (compression).

What basis vectors \( M_{ij} \) should we choose?

Pixel basis (grayscale)

- Straightforward basis: The pixels!
- Define

\[ M_{ij} = e_i e_j^T, \]

where \( e_i \) is the \( i \)-th column of the identity matrix \( I \).
- \( M_{ij} \) is zero everywhere except in its \((ij)\) entry.
- Advantages:
  - Simple basis vectors.
  - Straightforward to obtain coefficients.
- Disadvantages:
  - Must approximate every coefficient or loose entire pixels.
  - Poor compression ratio.
Discrete Cosine Transform (DCT) basis

Discrete Cosine Transform for a 1D signal
Component-wise form

- Input: vector \( x \in \mathbb{R}^n \).
- Output: vector \( y \in \mathbb{R}^n \).
- Transformation:

\[
y_u = C(u) \sqrt{\frac{2}{n}} \sum_{i=1}^{n} x_i \cos \left( \frac{(2i-1)(u-1)\pi}{2n} \right),
\]

where

\[
C(u) = \begin{cases} 
1/\sqrt{2} & \text{if } u = 1, \\
1 & \text{otherwise}. 
\end{cases}
\]

Discrete Cosine Transform for a 1D signal
Matrix form

- Input: vector \( x \in \mathbb{R}^n \).
- Output: vector \( y \in \mathbb{R}^n \).
- Transformation:

\[
y = Zx,
\]

where

\[
Z \in \mathbb{R}^{n \times n} \text{ and } z_{ij} = C(i) \sqrt{\frac{2}{n}} \cos \left( \frac{(2j-1)(i-1)\pi}{2n} \right),
\]

- In MATLAB: \( Z = \text{dctmtx}(n) \)
- \( Z \) is orthogonal, i.e. \( Z^{-1} = Z^T \)

The DCT is a change of basis!
Discrete Cosine Transform for a 2D signal

Forward DCT:
- Input: matrix $X \in \mathbb{R}^{n \times n}$.
- Output: matrix $Y \in \mathbb{R}^{n \times n}$.
- Transformation: $Y = ZXZ^T$.

Inverse DCT:
- Input: matrix $Y \in \mathbb{R}^{n \times n}$.
- Output: matrix $X \in \mathbb{R}^{n \times n}$.
- Transformation: $X = Z^TYZ$.

Color spaces

- RGB:
  $$p_{RGB} = \begin{pmatrix} r \\ g \\ b \end{pmatrix},$$
  where $r$, $g$, and $b$ are the red, green, and blue components of the pixel $p$, respectively.
- YCbCr:
  $$p_{YCbCr} = \begin{pmatrix} y \\ c_b \\ c_r \end{pmatrix},$$
  where $y$ is the luminance, and $c_b$ and $c_r$ are the chroma (blue) and chroma (red) components, respectively.
- (Analog) RGB and (analog) YCbCr are related by
  $$p_{YCbCr} = \begin{pmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{pmatrix} p_{RGB}.$$
- Another change of basis!

Color images

- The human eye is highly sensitive to small changes in luminance (light intensity).
- The human eye is less sensitive to small changes in color.
- Better compression can be achieved by compressing the luminance and color information separately.
- Therefore, we abandon the RGB color space in favor of the YCbCr color space.

JPEG Image Compression Pipeline

1. Read the input image.
2. Transform the image to the YCbCr color space.
3. For each component:
   3.1 Partition into $8 \times 8$ tiles.
   3.2 For each tile:
      3.2.1 Shift the range from unsigned to signed.
      3.2.2 Transform using the Forward DCT.
      3.2.3 Quantize (reduce range and round to integer). Lossy!
      3.2.4 Compress coefficients (lossless).
4. Store compressed image.
JPEG Image Decompression Pipeline

1. Read compressed image.
2. For each component:
   2.1 Partition into $8 \times 8$ tiles.
   2.2 For each tile:
      2.2.1 Decompress the coefficients (lossless).
      2.2.2 Dequantize (increase range and convert to floating point).
      2.2.3 Transform using the Inverse DCT.
      2.2.4 Shift the range from signed to unsigned.
3. Transform to the RGB color space.