DTA096A - Lab 2 Report

Visual Representation and Information Theory

Name: Andrea Faccioli
Date: 03/10/2025

1 Experimental Setup

The experiments have been carried out using Python within a Jupyter Notebook environment. Core libraries included NumPy for numerical operations, OpenCV and scikit-image for image processing. All scripts and functions used are provided in Appendix A.2.

2 Information Variation Within and Across Images

2.1 Spatial and Inter-Image Entropy

The entropy values show how much information or randomness each image contains, and they are directly linked to the compressibility of an image. In the following tasks three images have been analyzed, each one characterized by a different level of details. The smooth moon image has the lowest entropy (H(X) = 4.88), reflecting low variability and high redundancy. The camera and grass images have higher entropy (7.23 and 7.28), consistent with their richer textures and edges that increase intensity variation. These results confirm that smoother regions contain less information and are therefore more compressible, while textured or edge-rich regions contain more information and are harder to compress efficiently. Consistently, the moon image shows a narrow histogram concentrated around mid-gray levels, while the other two are spread out over the entire range.

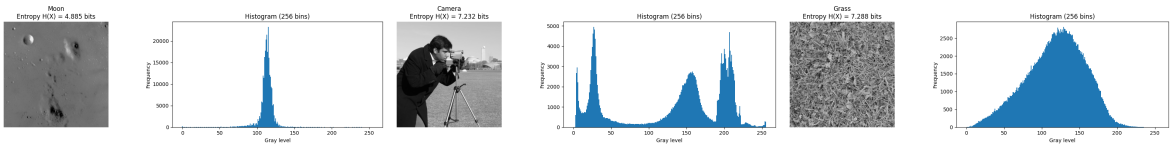

Figure 1: 256-bin histogram and global entropy H(X) of 3 gray scale images

The histograms of the three crops highlight the strong variability within a single image. The smooth crop shows a single sharp peak and low entropy, meaning high redundancy and easy compressibility. The edge-rich crop has two peaks and moderate entropy, reflecting contrast between regions. The textured crop has a spread-out histogram and high entropy, indicating high variability and low compressibility.

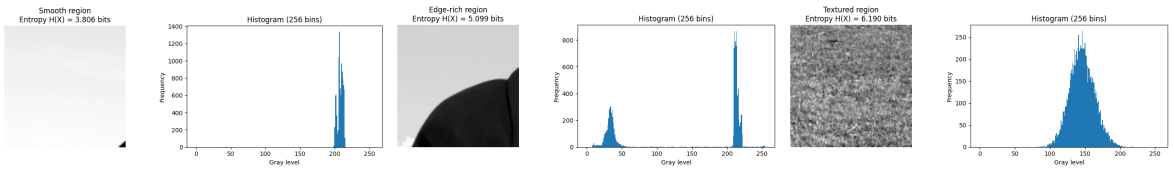

Figure 2: 256-bin histogram and entropy H(X) of 3 crops of the same gray scale image (Camera)

The block-wise local entropy map highlights how information content varies across the image. High-entropy regions appear as bright areas in the heatmap and correspond to visually complex structures such as edges, textures, and detailed patterns (face, camera support). In contrast, smooth or homogeneous regions (sky, black coat) show low entropy, appearing darker due to their uniform intensity.

3 Prediction: Type and Length

3.1 Predictors and Residual vs. Conditional Entropy

The three predictive schemes (left neighbor, average of left and top, and JPEG-LS median predictor) were applied to three test images of varying texture: *moon* (smooth), *camera* (moderately detailed), and *grass*

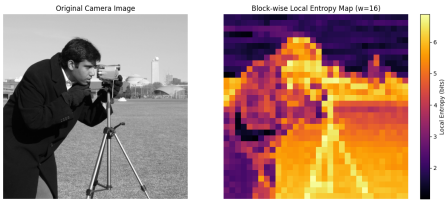

Figure 3: Block-wise local entropy map of a gray scale image (Camera)

(highly textured). For each method, the residuals $R = X - \hat{X}$ were computed and analyzed. The residual histograms showed that smoother images produced sharper, narrower peaks centered around zero, indicating that the predictors captured most of the pixel variation. In contrast, the textured image (*grass*) exhibited wider histograms, reflecting higher prediction errors due to stronger local intensity variations.

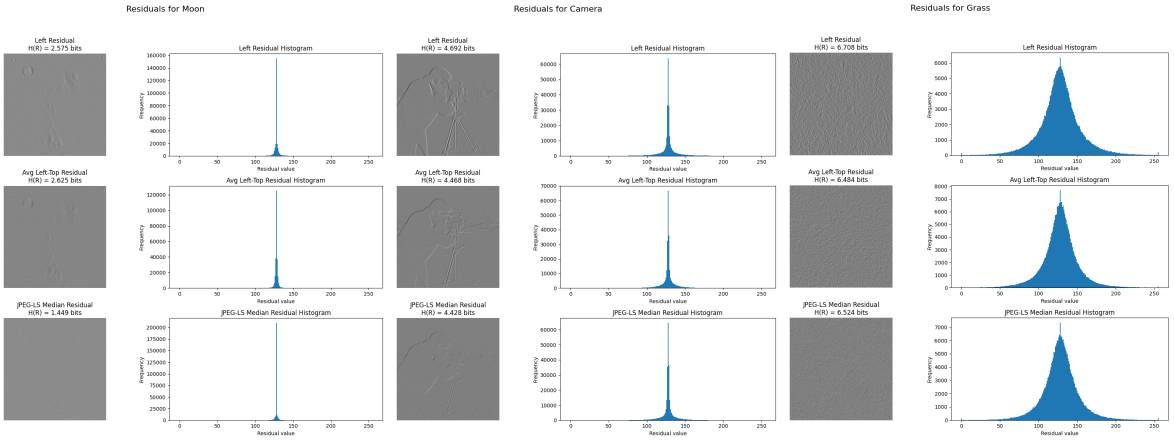

Figure 4: Residual image and residual histogram of 3 gray scale images

The conditional entropies $H(Y \mid X)$ computed from the joint histogram of horizontal neighbours $(X_{i,j-1}, X_{i,j})$ are very close to the marginal entropies of the residuals $H(R)$ produced by the left predictor;

Table 1: Comparison of Conditional Entropy $H(Y|X)$ and Residual Entropy $H(R)$ for the Left Predictor

Image	$H(Y\|X)$ **[bits]**	$H(R)$ **[bits]**
Moon	2.411	2.575
Camera	4.014	4.692
Grass	6.482	6.708

In theory, these two quantities should be similar, because both reflect how well the left neighbor predicts the current pixel. The small differences we observe ($H(R)$ being slightly higher) come from the fact that $H(R)$ treats the residuals as an independent signal without explicitly conditioning on the neighbor, while $H(Y \mid X)$ accounts for this dependency exactly.

4 Transform Coding: DCT vs. KLT

4.1 Basis Visualization and Energy Compaction

The 2D DCT basis functions represent spatial frequency patterns: low-frequency bases correspond to smooth intensity variations and gradients, while higher-frequency ones capture fine details and texture. In contrast, the KLT (Karhunen–Loève Transform) basis is data-adaptive and derived from the image's covariance structure, so its leading components reflect the dominant local patterns present in the image, such as edges, orientations, or textures.

A comparison of the bases is reported in Figure 5: the first KLT components closely resemble the low-frequency DCT bases, while producing also localized and oriented patterns to better match the image content. Thus, DCT provides a fixed, general-purpose frequency representation, whereas KLT offers optimal, image-specific one.

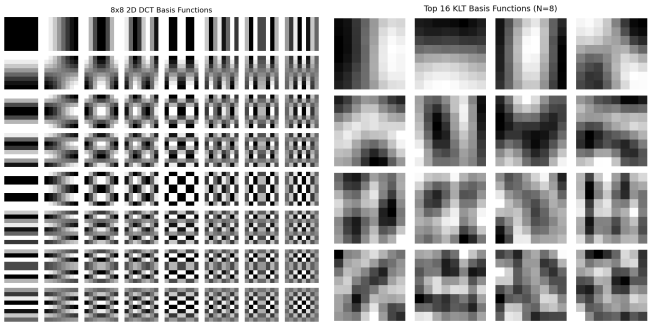

Figure 5: 2D DCT and Top 16 KLT Basis Functions

Results of applying DCT and KLT to *camera* image have been reported in Table 2. The results show that both DCT and KLT achieve strong energy compaction, as most of the signal energy is concentrated in a small fraction of coefficients. However, the KLT consistently outperforms the DCT in terms of reconstruction quality for the same retention ratio, achieving notably higher PSNR values. This is expected since the KLT is data-adaptive and optimally decorrelates the image patches, capturing the most significant variations with fewer coefficients. The DCT, being a fixed transform, performs slightly less efficiently but still provides excellent energy compaction, as seen from the nearly identical energy ratios. Overall, the KLT offers superior compression efficiency, while the DCT remains a simpler and computationally efficient approximation.

Table 2: Energy Compaction Analysis of DCT and KLT Transforms applied to *camera*

Retention Ratio	**DCT PSNR (dB)**	**KLT PSNR (dB)**	**DCT Energy Ratio**	**KLT Energy Ratio**
0.1	27.34	30.13	0.995	0.997
0.2	29.35	33.05	0.997	0.999
0.3	31.53	35.85	0.998	0.999
0.4	32.55	38.23	0.998	1.000
0.5	34.44	41.23	0.999	1.000
0.6	35.57	44.22	0.999	1.000
0.7	37.36	47.83	0.999	1.000
0.8	39.68	53.62	1.000	1.000
0.9	43.28	61.84	1.000	1.000

Reconstructed images with DCT and KLT for the same coefficient retention (30%) have have been reported in Figure 6. The right image, reconstructed with KLT, presents sharper edges and fewer blocky artifacts, in line with theory expectations.

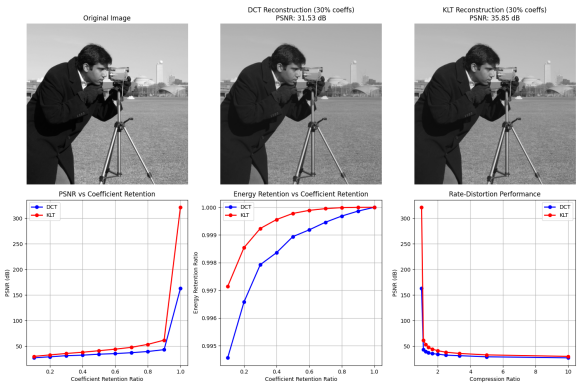

Figure 6: Reconstructed images comparison

In Figure 7 the MSE of reconstructed images with two different block sizes (4 and 16) have been plotted, highlighting the superior performance of KLT basis functions for both the dimensions.

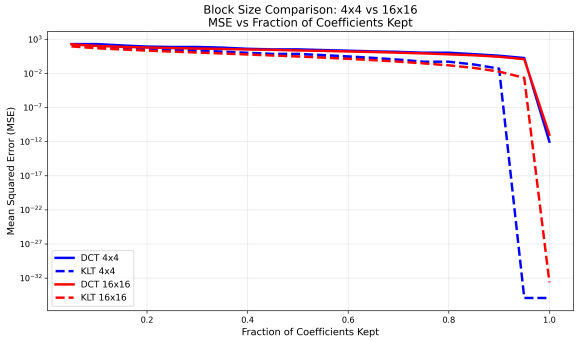

Figure 7: Comparison

5 Entropy Coding: Huffman vs. Arithmetic

5.1 Average Code Length vs. Entropy

In this section two different coding strategies will be analyzed and compared: Huffman coding and Arithmetic coding. The expirement requires to compute firstly the PMF (Probability Mass Function) of the residuals of an image: *camera* image has been used for this task, with left predictor defined in section 3.1.

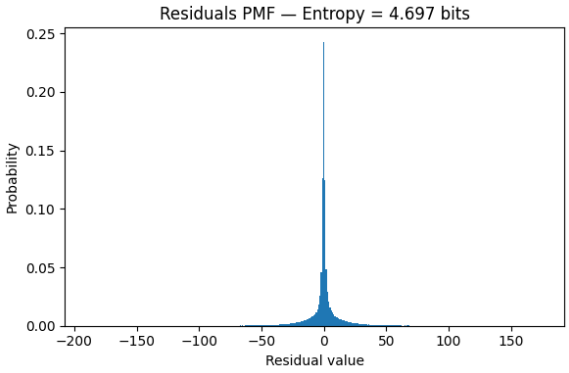

Figure 8: Residuals PMF

Then, the source entropy was computed as $H(R) = 4.697$ bits per symbol, representing the theoretical lower bound on the average code length achievable by any lossless coder. The Huffman coder achieved an average code length of $\bar{L}_{\text{Huffman}} = 4.715$ bits per symbol, while the Arithmetic coder reached $\bar{L}_{\text{Arithmetic}} = 4.697$ bits per symbol, effectively matching the entropy limit. The small redundancy observed for Huffman coding (0.018 bits/symbol) arises because Huffman codes must assign an integer number of bits per symbol, which prevents them from perfectly matching fractional probabilities. Arithmetic coding, on the other hand, encodes entire sequences as subintervals of the unit range, allowing it to represent fractional bit lengths precisely and thus achieve compression much closer to the theoretical entropy bound. This confirms that while Huffman coding is simpler, Arithmetic coding provides optimal performance in term of code length.

Table 3: Coding performance compared to the entropy bound for the residuals

Metric	**Bits per symbol**
Residual entropy $H(R)$	4.697
Average Huffman code length $\bar{L}_{\text{Huffman}}$	4.715
Average Arithmetic code length $\bar{L}_{\text{Arithmetic}}$	4.697
Huffman redundancy $\bar{L}_{\text{Huffman}} - H(R)$	0.018
Arithmetic redundancy $\bar{L}_{\text{Arithmetic}} - H(R)$	0.000

A Appendix 1

A.1 Images collection

The images processed in Lab 2 session here.

A.2 Source code

The complete code of Lab 2 session can be found here.