

**A Lecture Series on
DATA COMPRESSION**

Lossy Compression — Quantization

Abdou Youssef (Speaker)

Anastase Nakassis (MDVTG Manager)

Motivation

- Need for low bitrate
 - Less than 0.2 bit per pixel for video
- Inadequacy of lossless compression
 - Achievable compression ratio hardly above 2
 - Achievable bitrate hardly below 4 bits per pixel
- Presence of visual redundancy that can be greatly exploited

General Scheme of Lossy Compression

- Approach
 1. Transform: Convert the data to the frequency domain
 2. Quantize: Under-represent the high frequencies
 3. Losslessly compress the quantized data
- Properties of transforms
 - Decorrelation of data
 - Separation of data into
 - * vision-sensitive data (low-frequency data)
 - * vision-insensitive data (high-frequency data)
- Various transforms achieve both properties
 - Fourier Transform
 - Discrete Cosine Transform (DCT)
 - Other Fourier-like transforms: Haar, Walsh, Hadamard
 - Wavelet transforms

- Properties of quantization
 - Progressive under-representation of higher-frequency data
 - Conversion of visual redundancy to symbol-level redundancy that leads to high compression ratios
 - Minimum and controlled distortion: more errors in less sensitive regions

- Examples of quantizers
 - Uniform scalar quantization
 - Non-uniform scalar quantization
 - Vector quantization (VQ)

Illustration of the Effects of Transforms and the Need for Quantization

Scalar Quantization

- Definition of Quantization/Dequantization:
 - A k -level quantizer is typically characterized by $k + 1$ *decision levels* d_0, d_1, \dots, d_k , and by k *reconstruction levels* r_0, r_1, \dots, r_{k-1} .
 - The d_i 's divide the range of data under quantization into k consecutive intervals $[d_0, d_1)$ $[d_1, d_2)$ \dots $[d_{k-1}, d_k)$.
 - Each r_i is in $[d_i, d_{i+1})$, and can be viewed as the “centroid” of its interval.
 - Quantizing a number a means locating the interval $[d_i, d_{i+1})$ that contains a , and replacing a by index i .
 - Dequantization (in reconstruction) is the process of replacing each index i by the value r_i . This approximates every original number that was in interval $[d_i, d_{i+1})$ by the centroid r_i .
- A few quantizers assume that $d_0 = -\infty$ and $d_k = \infty$. But in the majority of quantizers d_0 and d_k are finite numbers representing the minimum and maximum value of the data being quantized.

Types of Scalar Quantizers

- Uniform Quantizers
 - All the decision intervals are of equal size $\Delta = \frac{d_k - d_0}{k}$
 - * $d_i = d_0 + i\Delta$
 - The reconstruction levels r_i 's are the centers of the intervals
 - * $r_i = \frac{d_i + d_{i+1}}{2}$
- Non-Uniform Quantizers
 - Either the decision intervals are not of equal size
 - Or the reconstruction levels are not the centers of their intervals
- Semi-Uniform Quantizers
 - Equal intervals
 - The reconstruction levels are not necessarily the interval centers

Illustration of Quantizers

Optimal Non-Uniform Quantizers (Max-Lloyd Quantizers)

- Assume the data under quantization follows a probability distribution $p(x)$

- The mean-square error (MSE) is

$$E = \sum_{l=0}^{k-1} \int_{d_l}^{d_{l+1}} (x - r_l)^2 p(x) dx$$

- To minimize the MSE, compute the partial derivatives of E with respect to each d_i and each r_i , and set them to 0
- Notation: The partial derivative of E with respect to a variable u is denoted E_u

Optimal Non-Uniform Quantizers (Cont.)

- $E_{r_i} = 2 \int_{d_i}^{d_{i+1}} (r_i - x)p(x)dx = 2[r_i \int_{d_i}^{d_{i+1}} p(x)dx - \int_{d_i}^{d_{i+1}} xp(x)dx]$
- $E_{r_i} = 0$ implies

$$r_i = \frac{\int_{d_i}^{d_{i+1}} xp(x)dx}{\int_{d_i}^{d_{i+1}} p(x)dx}, \text{ for all } i = 0, 1, \dots, k - 1$$

- This means that the optimal r_i is the probabilistic centroid of its interval
- $E_{d_i} = -(r_i - d_i)^2 p(d_i) + (r_{i-1} - d_i)^2 p(d_i)$
- $E_{d_i} = 0$ implies that $(r_{i-1} - d_i)^2 = (r_i - d_i)^2 p(d_i)$, that is, $d_i - r_{i-1} = r_i - d_i$. Hence,

$$d_i = \frac{r_{i-1} + r_i}{2}, \text{ for all } i = 1, 2, \dots, k - 1$$

- Remark: If $p(x)$ is not known theoretically, treat r_i as the statistical centroid of its interval, and d_i still as $\frac{r_{i-1} + r_i}{2}$

Max-Lloyd Quantizers (Cont.)

- The k centroid equations for the r_i 's, and the $k - 1$ decision level equations for the d_i 's form $2k - 1$ equations of $2k - 1$ unknowns
- Unfortunately, these equations are non-linear, and thus hard to solve directly.
- The Max-Lloyd technique is an iterative algorithm for deriving very accurate approximations of d_i 's and r_i 's
- Max-Lloyd Algorithm (for designing an optimal k -level quantizer)

1. Start with arbitrary initial estimate for the d_i 's

2. **Repeat**

- For all i , compute $r_i = \frac{\int_{d_i}^{d_{i+1}} xp(x)dx}{\int_{d_i}^{d_{i+1}} p(x)dx}$

- For all i , compute $d_i = \frac{r_{i-1} + r_i}{2}$

Until (the error between successive estimates of the d_i 's are less than a set tolerance)

Optimal Semi-Uniform Quantizers

- The decision levels d_i 's are known constants ($d_i = d_0 + i\Delta$)
- The reconstruction levels r_i are the probabilistic (or statistical) centroids of their intervals $[d_i, d_{i+1})$.

Uniform vs. Non-Uniform vs. Semi-Uniform Quantizers

	Advantages	Disadvantages
Uniform Quantizers	<ul style="list-style-type: none"> • Very simple • Fast to de-/quantize • Stores the d_i's & r_i's • Leads to good symbol redundancy 	<ul style="list-style-type: none"> • The quality of reconstructed data is not optimal
Non-Uniform Quantizers	<ul style="list-style-type: none"> • The quality of reconstructed data is optimal 	<ul style="list-style-type: none"> • Costly to compute the d_i's & r_i's • Costly to de-/quantize • Stores the d_i's & r_i's
Semi-Uniform Quantizers	<ul style="list-style-type: none"> • Trivial d_i's and fast to compute r_i's • Fast to de-/quantize • No storage of the d_i's • good symbol redundancy • The quality is very good 	<ul style="list-style-type: none"> • The quality of reconstructed data is less than optimal • Still requires the storage of the r_i's

Vector Quantization

- Scalar quantization is insensitive to inter-pixel correlations
- Scalar quantization not only fails to exploit correlations, it also destroys them, thus hurting the image quality
- Therefore, quantizing correlated data requires alternative quantization techniques that exploit and largely preserve correlations
- Vector quantization (VQ) is such a technique
- VQ is a generalization of scalar quantization: It quantizes vectors (contiguous blocks) of pixels rather than individual pixels
- VQ can be used as a standalone compression technique operating directly on the original data (images or sounds)
- VQ can also be used as the quantization stage of a general lossy compression scheme, especially where the transform stage does not decorrelate completely, such as in certain applications of wavelet transforms

The Main Technique of VQ

- Build a dictionary or “visual alphabet”, called codebook, of codevectors. Each codevector is a (1D or 2D) block of n pixels
- Coding
 1. Partition the input into blocks (vectors) of n pixels
 2. For each vector v , search the codebook for the best matching codevector \hat{v} , and code v by the index of \hat{v} in the codebook
 3. Losslessly compress the indices
- Decoding (A simple table lookup)
 1. Losslessly decompress the indices
 2. Replace each index i by codevector i of the codebook
- The codebook has to be stored/transmitted
- The codebook can be generated on an image-per-image basis or a class-per-class basis

VQ Issues

- Codebook size (# of codevectors) N_c — how big or small?
- Codevector size n — how big or small?
- Codebook construction — what codevectors to include?
- Codebook structure — for faster best-match searches
- Global or local codebooks — class- or image-oriented VQ?

Sizes of Codebooks and Codevectors (Tradeoffs)

- A large codebook size N_c allows for representing more features, leading to better reconstruction quality
- But a large N_c causes more storage and/or transmission
- A small N_c has the opposite effects
- Typical values for N_c : $2^7, 2^8, 2^9, 2^{10}, 2^{11}$
- A larger codevector size n exploits inter-pixel correlations better
- But n should not be larger than the extent of spatial correlations

Codevector Size Optimization

- Optimal codevector size n for minimum bitrate and constant N_c
 - Consider $N \times N$ images with r bits per pixel, and let $n = p \times p$ be the block size
 - Size S of the compressed image: $\frac{N^2}{p^2} \log N_c + p^2 r N_c$ bits
 - The bitrate $R = \frac{S}{N^2} = \frac{\log N_c}{p^2} + \frac{r N_c}{N^2} p^2$
 - For minimum bitrate R , the derivative $\frac{dR}{dp} = 0$
 - Since $\frac{dR}{dp} = -2 \frac{\log N_c}{p^3} + 2 \frac{r N_c}{N^2} p$, we have

$$p = \left[\frac{N^2 \log N_c}{r N_c} \right]^{\frac{1}{4}}$$

- Concrete figures of optimal n for $N = 512$

N_c	2^6	2^7	2^8	2^9	2^{10}	2^{11}
p	7.4	6.5	5.6	4.9	4.2	3.6

Codevector Size Optimization (Cont.)

- Therefore, optimal 2D codevector sizes are 4×4 and 8×8 for powers-of-2 sizes
- Interestingly, statistical studies on natural images have shown that there is little correlation between pixels more than 8 positions apart, and in fact, most of the correlations are among pixels that are within 4 positions away.
- Therefore, 4×4 and 8×8 codewords are excellent choices from both the bitrate standpoint and the correlation-exploitation standpoint

Construction of Codebooks (The Linde-Buzo-Gray Algorithm)

- Main idea

1. Start with an initial codebook of N_c vectors;
2. Form N_c classes from a set of training vectors: put each training vector v in Class i if the i -th initial codeword is the closest match to v ;

Note: The training set is the set of all the blocks of the image being compressed. For global codebooks, the training set is a the set of all the blocks of a representative subset of images selected from the class of images of the application.

3. Repeatedly restructure the classes by computing the new centroids of the recent classes, and then putting each training v vector in the class of v 's closest new centroid;
4. Stop when the total distortions (differences between the training vectors and their centroids) ceases to change much;
5. Take the most recent centroids as the codebook.

The Linde-Buzo-Gray (LBG) Algorithm (Cont.)

- The algorithm in detail

1. Start with a set of training vectors and an initial code book $\hat{X}_1^{(1)}, \hat{X}_2^{(1)}, \dots, \hat{X}_{N_c}^{(1)}$. Initialize the iteration index l to 1 and the initial distortion $D^{(0)}$ to ∞ .

2. For each training vector v , find the closest $\hat{X}_i^{(l)}$:

$$d(v, \hat{X}_i^{(l)}) = \min_{1 \leq k \leq N_c} d(v, \hat{X}_k^{(l)}),$$

where $d(v, w)$ is the (Euclidian or MSE) distance between the vectors v and w .

Put v in class i .

3. Compute the new total distortion

$$D^{(l)} = \sum_{i=1}^{N_c} \sum_{v \in \text{Class } i} d(v, \hat{X}_i^{(l)}).$$

If $|\frac{D^{(l-1)} - D^{(l)}}{D^{(l-1)}}| \leq \text{TOLERANCE}$, the convergence is reached; stop and take the most recent $\hat{X}_1^{(l)}, \hat{X}_2^{(l)}, \dots, \hat{X}_{N_c}^{(l)}$ to be the codebook.

Else, goto step 4.

4. Compute the new class centroids (vector means):

$$l := l + 1; \text{ and for all } i, \hat{X}_i^{(l)} := \frac{\sum_{v \in \text{Class } i} v}{\text{size of Class } i}.$$

Goto 2.

Initial Codebook

- Three methods for constructing an initial codebook
 - The random method
 - Pairwise Nearest Neighbor Clustering
 - Splitting
- The random method:
 - Choose randomly N_c vectors from the training set
- Pairwise Nearest Neighbor Clustering:
 1. Form each training vector into a cluster
 2. Repeat the following until the number of clusters becomes N_c : merge the 2 clusters whose centroids are the closest to one another, and recompute their new centroid
 3. Take the centroids of the N_c clusters as the initial codebook

- Splitting
 1. Compute the centroid X_1 of the training set
 2. Perturb X_1 to get X_2 , (e.g., $X_2 = .99 * X_1$)
 3. Apply LBG on the current initial codebook to get an optimum codebook
 4. Perturb each codevector to double the size of the codebook
 5. Repeat step 3 and 4 until the number of codevectors reaches N_c
 6. In the end, the N_c codevectors are the whole desired codebook

Codebook Structure

(m -ary Trees)

- Tree design and construction
 1. Start with codebook as the leaves
 2. Repeat until you construct the root
 - cluster all the nodes of the current level into m -node clusters
 - create a parent node for each cluster of m nodes
- Searching for a best match of a vector v in the tree
 - Search down the tree, always following the branch that incurs the least MSE
 - The search time is logarithmic (rather than linear) in the codebook size
- Refined Trees
 - Tapered trees: The number of children per node increases as one moves down the tree
 - Pruned Trees: Eliminate the codevectors that contribute little to distortion reduction

Advanced VQ

- Prediction/Residual VQ
 - Predict vectors
 - compute the residual vectors,
 - VQ-Code the residual vectors
- Mean/Residual VQ (M/R VQ)
 - Compute the mean of each vector and subtract it from the vector
 - VQ-code the residual vectors
 - Code the means using DPCM and scalar quantization
 - Remark: Once the means are subtracted from the vectors, many vectors become very similar, thus requiring fewer codevectors to represent them
- Interpolation/residual VQ (I/R VQ)
 - subsample the image by choosing every l -th pixel
 - code the subsampled image using scalar quantization
 - Upsample the image using bilinear interpolation
 - VQ-code the residual (original-upsampled) image
 - remark: residuals have fewer variations, leading to smaller codebooks

- Gain/Shape VQ (G/S VQ)
 - Normalize all vectors to have unit gain (unit variance)
 - Code the gains using scalar quantization
 - VQ-code the normalized vectors