CS 6351 DATA COMPRESSION

VECTOR QUANTIZATION

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE

By the end of this lecture, you will be able to:

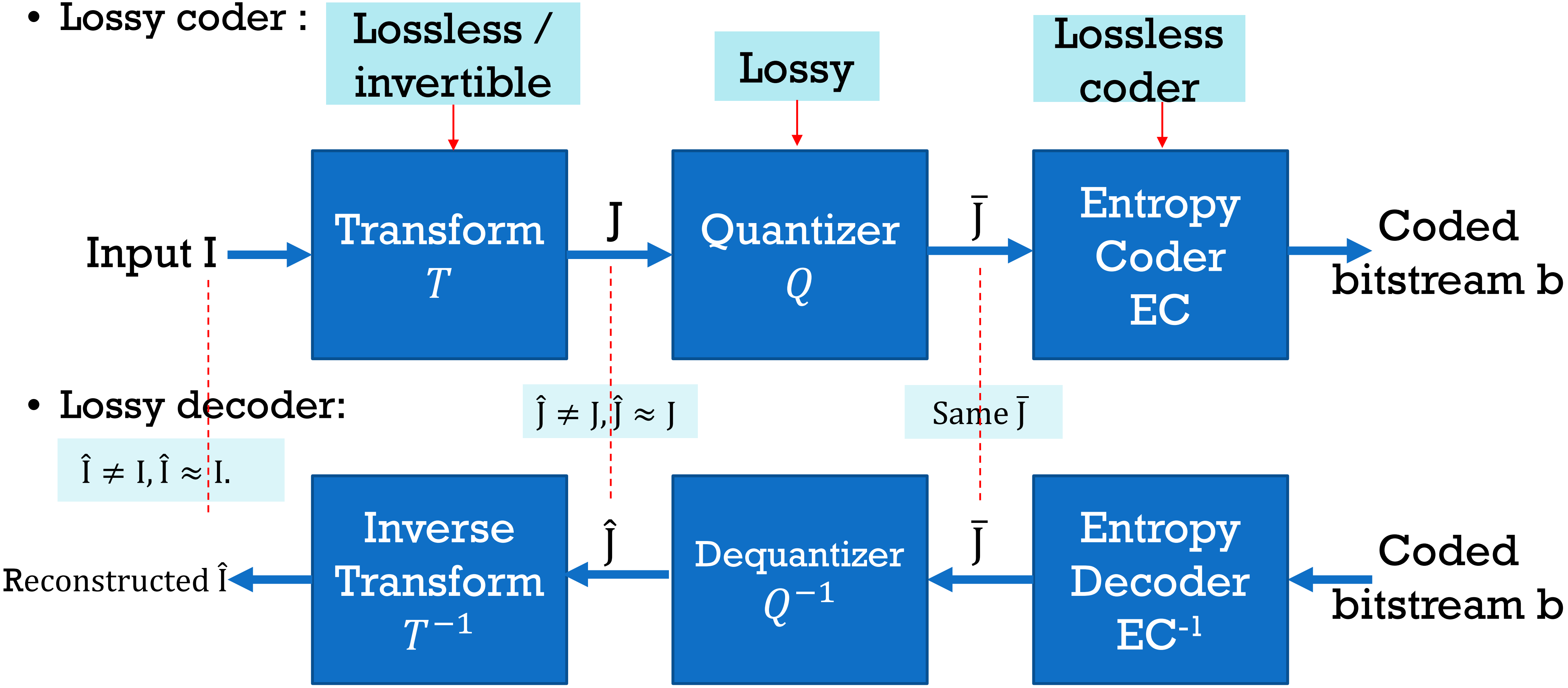
1. Describe vector quantization and its correlation-preserving property
2. Apply vector quantization for lossy compression of audio-visual signals
3. Optimize codevector size
4. Derive the Linde-Buzo-Gray (LBG) algorithm for constructing optimal VQ codebooks
5. Draw a parallel between LBG algorithm and k-means clustering
6. Design tree structures of codebooks for more efficient search of best matches
7. Derive more advanced versions of VQ, and argue about their advantages
8. Analyze the tradeoff between reconstruction quality, codebook size, and image variability, and conceptually argue why transforms are better than VQ

OUTLINE

- Why vector quantization
- Definition of vector quantizers
- Coding and decoding with VQ
- Optimization of codevector size
- Linde-Buzo-Gray algorithm for constructing optimal codebooks
- Connection between the LBG algorithm and k-means clustering
- Faster search in codebooks
- Advanced VQ
- Closing remarks about VQ

GENERAL SCHEME OF LOSSY COMPRESSION

- Lossy coder :
- Lossy decoder:



Can we replace (the transform & scalar quantizer modules) with a more sophisticated quantizer?

BACKGROUND AND MOTIVATION

- 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
  - Or use transforms
- We explored the use of transforms
- Now we want to explore using the alternative approach:
  - Vector quantization

VECTOR QUANTIZATION (VQ)

- VQ is a generalization of scalar quantization: It quantizes vectors (contiguous blocks) of data rather than individual elements of data
- VQ can be used as a standalone compression technique operating directly on the original data (e.g., 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
- VQ can also be used to quantize (and thus compress) groups of parameters in parameterized models (e.g., coefficients of polynomial models of sound signals)

THE MAIN TECHNIQUE OF VQ

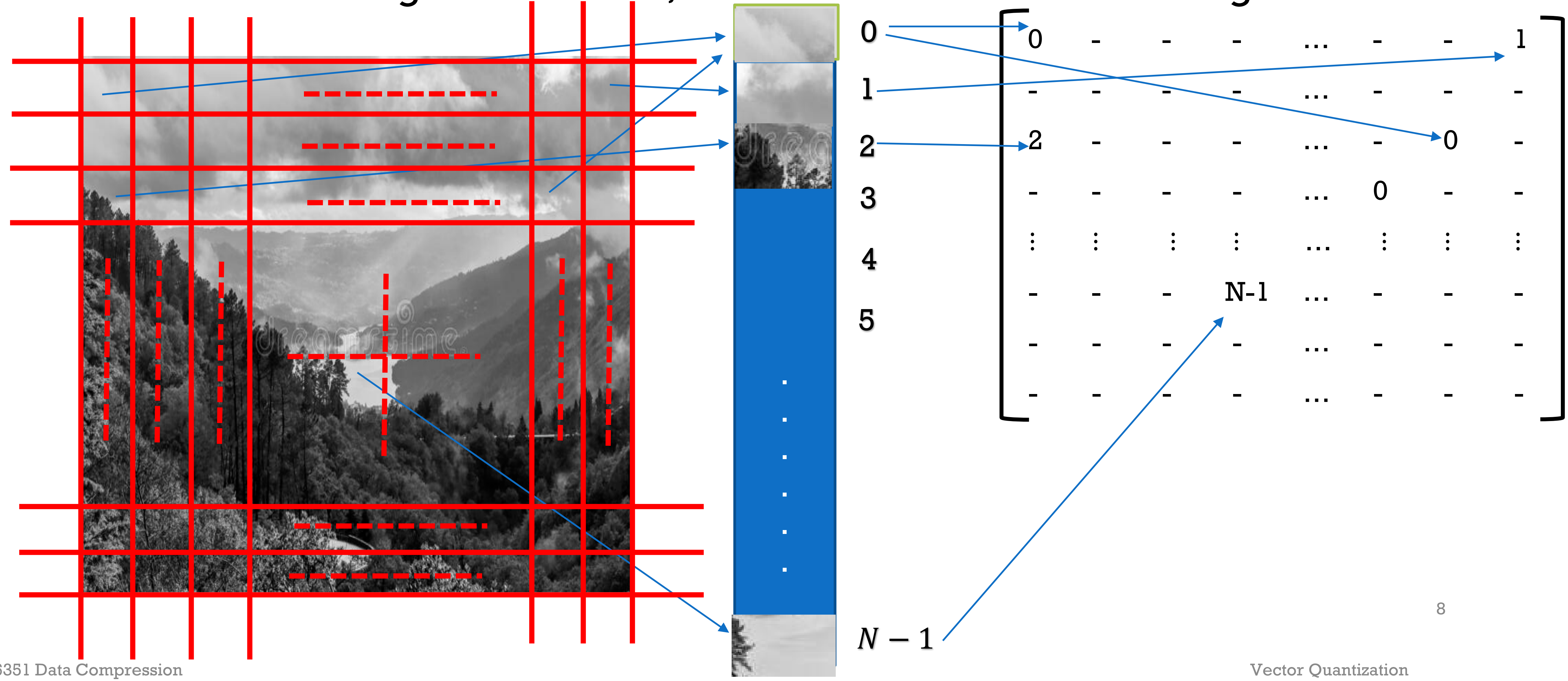
- **Build a dictionary** $CB[0{:}\,N-1]$, or ``audio/visual alphabet", called ***codebook***, of codevectors
  - Each codevector $CB[i]$ is a 1D/2D block of n samples or $p \times q$ pixels
- **Coding**
  1. Partition the input into non-overlapping blocks (vectors) of n pixels
  2. For each vector u, search the codebook CB for the best matching codevector $\hat{u}$, and code u by the index i of $\hat{u}$ in CB (i.e., $\hat{u} = CB[i]$)
  3. Losslessly compress the indices
- **Decoding** (A simple table lookup):
  1. Losslessly decompress the indices
  2. Replace each index i by codevector $CB[i]$

VQ CODING ILLUSTRATION

1. VQ applied to images

assuming we got a codebook $CB[0: N-1]$

2. Divide the image into blocks, and find+code their matching codevectors



CODEBOOK MATTERS (1/3)

- The codebook can be stored/transmitted or assumed/shared b/w coder & decoder
- The codebook can be generated on an image-by-image basis or class-by-class basis or application-by-application basis, with different pros and cons
- The **image-per-image** basis
  - For each separate image, a customized, optimized codebook is created and included in the coded bitstream
  - **Pros**: better fidelity, better quality of reconstruction
  - **Cons**:
    - higher bitrates, lower compression ratios
    - More time per image, to construct the optimal customized codebook

CODEBOOK MATTERS (2/3)

- The codebook can be stored/transmitted or assumed/shared b/w coder & decoder
- The codebook can be generated on an image-by-image basis or class-by-class basis or application-by-application basis, with different pros and cons
- The **application-by-application** basis
  - Only one codebook is created from a large, representative set of images/objects in the application domain (e.g., x-rays, animal pictures, human pictures, etc.)
  - The codebook is shared b/w coder and decoder, and not stored in the coded bitstreams
  - **Pros**:
    - lower bitrates, higher compression ratios
    - Less time per image because codebook is available (constructed ahead of time)
  - **Cons**: lower fidelity, lower quality reconstruction

CODEBOOK MATTERS (3/3)

- The codebook can be stored/transmitted or assumed/shared b/w coder & decoder
- The codebook can be generated on an image-by-image basis or class-by-class basis or application-by-application basis, with different pros and cons
- The **class-by-class** basis
  - One codebook per class, constructed ahead of time. #codebooks = #classes
  - Those codebooks are shared between coder and decoder
  - The coded bitstream has to include a code of the class of the coded image
  - Class-determination:
    - <u>Manual</u>: The coder can be informed by the user which class an input belongs to, or
    - <u>Automated</u>: A separate algorithm (classifier) has to classify the image
  - **Pros and cons**: half-way between the image-by-image and the application-by-application approaches

VQ ISSUES

- Codebook size (# of codevectors) N_c
- Codevector size n
- 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 a larger bitrate
  - But that is mitigated if the codebook is shared
- A small N_c has the opposite effects
- Typical values for N_c: $2^7, 2^8, 2^9, 2^{10}, 2^{11}$
- How about codevector size?
  - 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 (1/4)
-- CODEBOOK INCLUDED IN BITSTREAM --

- Optimal codevector size for minimum bitrate
  - Consider $N \times N$ images with r bits/pixel, for some given fixed N and r
  - Assume the number N_c of codevectors in the codebook is given and fixed
  - Let $n = p \times p$ be the codevector size to be optimized (so the variable to be optimized is p)
  - To simplify matters, assume that we use fixed-length encoding to code the indices of the matching codevectors, i.e., $\log N_c$ bits per index
  - The size S of a VQ-compressed image is:
    - S = (size of the image blocks' codevectors' indices codes) + (size of the codebook)
      - Size of the codevectors' indices codes: $\left(\frac{N}{p}\right)^2 \log N_c$ (Why?)
      - Size of the codebook: $p^2 r N_c$ (Why?)
    - Therefore, $S = \left(\frac{N}{p}\right)^2 \log N_c + p^2 r N_c$

CODEVECTOR SIZE OPTIMIZATION (2/4)
-- CODEBOOK INCLUDED IN BITSTREAM --

- Optimal codevector size $p \times p$ for minimum bitrate:
  - The size S of a VQ-compressed image is: $\boldsymbol{S} = \left(\frac{\boldsymbol{N}}{\boldsymbol{p}}\right)^{\mathbf{2}} \mathbf{log}\,\boldsymbol{N_c} + \boldsymbol{p^2 r N_c}$
  - The bitrate $R = \frac{S}{N^2} = \frac{\mathbf{log}\,\boldsymbol{N_c}}{p^2} + \frac{\boldsymbol{rN_c}}{N^2}\boldsymbol{p^2}$. Treat it as a function of the variable p
  - To minimize the bitrate R, compute its derivative and set it to 0: $\frac{dR}{dp} = 0$
    - Assume temporarily that p is a real variable (rather than just a positive integer)
  - $\frac{dR}{dp} = -2\frac{\mathbf{log}\,\boldsymbol{N_c}}{p^3} + 2\frac{\boldsymbol{rN_c}}{N^2}p$
  - $\frac{dR}{dp} = 0 \Rightarrow -2\frac{\mathbf{log}\,\boldsymbol{N_c}}{p^3} + 2\frac{\boldsymbol{rN_c}}{N^2}p = 0 \Rightarrow \frac{\mathbf{log}\,\boldsymbol{N_c}}{p^3} = \frac{\boldsymbol{rN_c}}{N^2}p \Rightarrow$

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

CODEVECTOR SIZE OPTIMIZATION (3/4)
-- CODEBOOK INCLUDED IN BITSTREAM --

- Optimal codevector size $p \times p$ for minimum bitrate

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

- Concrete values of optimal p for $N = 512$ and different values of N_c:

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
Closest power-of-2 value of p	8	8	4	4	4	4

CODEVECTOR SIZE OPTIMIZATION (4/4)
-- CODEBOOK INCLUDED IN BITSTREAM --

- Concrete values of optimal p for $N = 512$ and different values of N_c:

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
Closest power-of-2 value of p	8	8	8	4	4	4

- Therefore, optimal 2D codevector (powers-of-2) sizes are 4×4 and 8×8
- Interestingly, statistical studies on natural images have shown that there is little or no correlation between pixels more than 8 positions apart
- Therefore, 4×4 and 8×8 codewords are excellent choices from both the bitrate standpoint and the correlation-exploitation standpoint

CONSTRUCTION OF CODEBOOKS (1/5)
-- THE LINDE-BUZO-GRAY ALGORITHM--

- Given one image, or a collection of images, how do we construct an optimal codebook of a given size N_c and a given block/codevector size of $p \times p$?
- Preliminaries:
  - Divide the image(s) into $p \times p$ blocks (there will be many of them, relative to the codebook size N_c). Let's call the set of those blocks "*dataset*"
  - Select/construct an N_c-block codebook such that every block in the dataset has on average a very good MSE-match in the codebook
    - A "very good MSE-match" means it has a minimum MSE on average, or near minimum
  - That means that the codebook is a good cross-representation of the dataset

CONSTRUCTION OF CODEBOOKS (2/5)
-- THE LINDE-BUZO-GRAY ALGORITHM--

- The codebook is a good cross-representation of the dataset
- How can we find such a small (N_c) number of representative blocks?
- **Strategy**: **clustering**
- Clustering is widely used in machine learning (in what is called unsupervised learning)
- There are many clustering algorithms, and more are being created
- One of the most widely studied and widely used clustering algorithms is ***k-means*** clustering
- Linde, Buzo and Gray have invented a codebook generation algorithm (called the **LBG algorithm**) that is equivalent to k-means clustering

CONSTRUCTION OF CODEBOOKS (3/5)
-- THE LINDE-BUZO-GRAY ALGORITHM--

- Linde, Buzo and Gray have invented a codebook generation algorithm (called the LBG algorithm) that is equivalent to k-means clustering
- The LBG algorithm (as the k-means algorithm) is an iterative algorithm, presented next

CONSTRUCTION OF CODEBOOKS (4/5)

-- THE MAIN IDEA OF THE LBG ALGORITHM--

Main idea:

1. Start with an initial codebook of N_c vectors: $V_1, V_2, \dots, V_{N_c}$;
2. Form N_c clusters from a set of training vectors (the dataset):
   - Put each training vector v in Cluster i if codeword V_i is the closest match to v;
3. Repeatedly restructure the classes by doing the following two steps:
   a. compute the new centroids of the clusters: $V_i = mean(\text{Cluster } i), \text{for } i = 1,2, \dots, N_c$
   b. Recluster by putting each training vector v in the class of v's closest new centroid;
4. Stop when the total distortion (differences between the training vectors and their centroids) ceases to change much, or when the centroids stop changing much;
5. Take the most recent centroids as the codebook.

CONSTRUCTION OF CODEBOOKS (5/5)
-- THE LBG ALGORITHM IN DETAIL --

1. Start with a set of training vectors and an initial codebook $\hat{U}_1^{(1)}, \hat{U}_2^{(1)}, \ldots, \hat{U}_{N_c}^{(1)}$;
2. Initialize: the iteration index k:=1; distortion $D^{(0)} := \infty$: *converged* := **false**;
3. While (not converged) do
   a. **Reclustering**: For each training vector v, find the closest $\hat{U}_i^{(k)}$, i.e., $d(v, \hat{U}_i^{(k)}) = \min_{1 \le j \le N_c} d(v, \hat{U}_j^{(k)})$, and put v in Cluster i; //$d(v,w)$ is the Euclidean distance between vectors v and w
   b. Compute the new total distortion $D^{(k)}$: $D^{(k)} = \sum_{i=1}^{N_c} \sum_{v \in Cluster\ i} d(v,\ \hat{U}_i^{(k)})$
   c. If $\left|D^{(k)} - D^{(k-1)}\right| < t$, where t is a given preset tolerance that is very small, then

      converged := true; //convergence is reached;

      take the most recent $\hat{U}_1^{(k)}, \hat{U}_2^{(k)}, \ldots, \hat{U}_{N_c}^{(k)}$ as the codebook, and return;
   d. Else

      $k := k + 1$;

      New centroids: compute the new cluster centroids (vector means): $\hat{U}_i^{(k)} = \frac{\sum_{v \in Cluster\ i} v}{|Cluster\ i|}, i = 1,2, \ldots, N_c$;

INITIAL CODEBOOK (1/4)

- The LBG algorithm starts from an initial codebook
- What could that initial codebook be?
- There are three methods for constructing an initial codebook
  - The random method
  - Pairwise Nearest Neighbor Clustering
  - Splitting
- They are addressed next

INITIAL CODEBOOK (2/4)
-- THE RANDOM METHOD --

- The random method:
  - Choose randomly N_c vectors (or blocks) from the dataset

INITIAL CODEBOOK (3/4)
-- PAIRWISE NEAREST NEIGHBOR CLUSTERING --

- The Pairwise Nearest Neighbor Clustering method:
  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 those N_c clusters as the initial codebook

INITIAL CODEBOOK (4/4)
-- THE SPLITTING METHOD --

- The splitting method:
  1. Compute the centroid X_1 of the entire training set
  2. Perturb X_1 to get X_2, (e.g., X_2=.99*X_1); call {X_1 , X_2} the *current codebook* CCB*;*
  3. Apply LBG on the current codebook CCB to get an optimum codebook: CCB=LBG(dataset, CCB);
  4. Perturb each codevector in CCB to double the size of CCB
  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)
-- WHY AND HOW TO CONSTRUCT IT --

- We showed the codebook before as an unstructured array
- But since we need to search for a best match in the codebook whenever we code each block/vector in an input, a linear search is too slow
- Therefore, we need alternative structures of the codebook that allow for faster search (for a best match)
- One good structure is an m-ary tree
- Tree design and construction:
  1. Start with the codebook as the leaves: one leaf node per codevector
  2. Repeat until you construct the root
     - cluster all the nodes of the current level of the tree into m-node clusters
     - create a parent node for each cluster of m nodes, and set that new node to the centroid of its m children

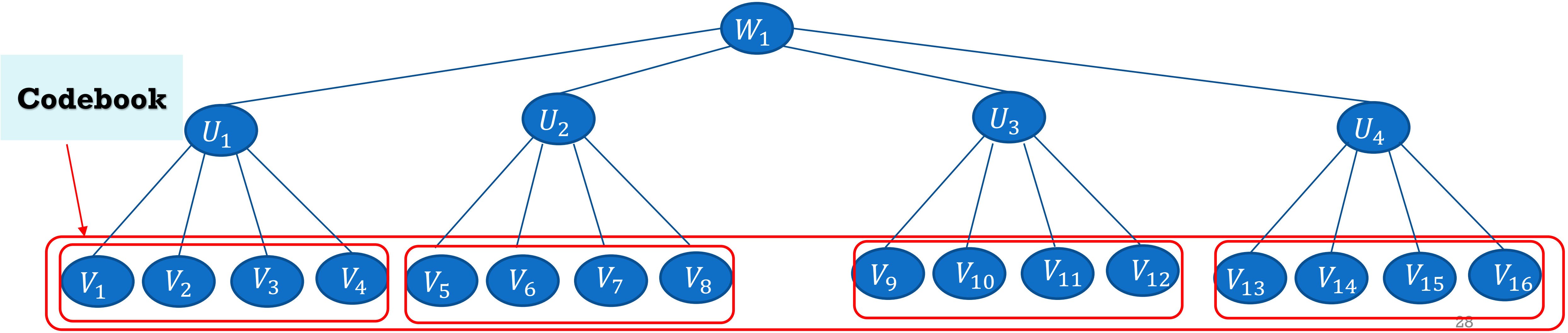
CODEBOOK STRUCTURE (m-ARY TREES)

-- HOW TO CONSTRUCT AN 4-ARY TREE --

1. Start with the codebook (of 16 vectors in this illustration)
2. Cluster the 16 codevectors into 4 clusters, each of 4 vectors:

$$C_1=\{V_1,V_2,V_3,V_4\},\ C_2=\{V_5,V_6,V_7,V_8\},\ C_3=\{V_9,V_{10},V_{11},V_{12}\},\ C_4=\{V_{13},V_{14},V_{15},V_{16}\}$$

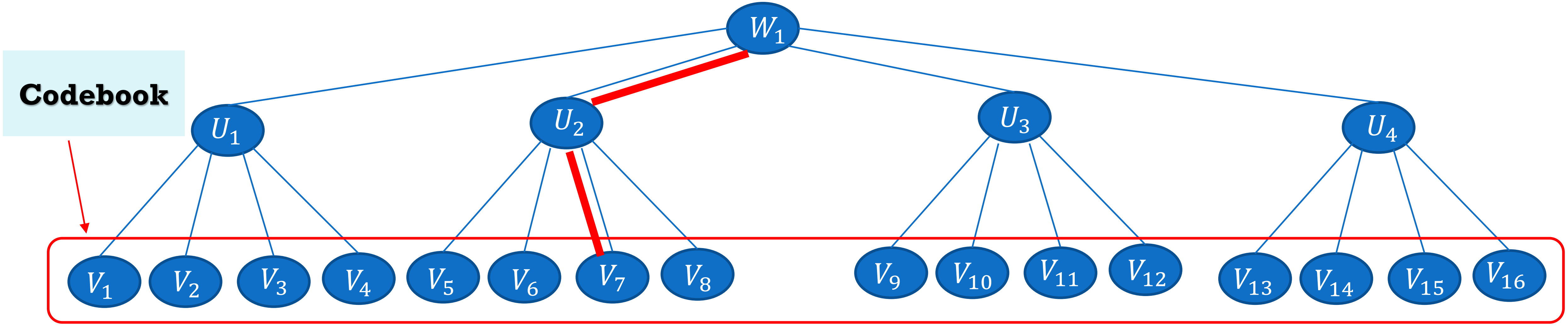
3. Create 4 new vectors: $\{U_1,U_2,U_3,U_4\}$ where $U_i = \text{mean}(C_i)$
4. Now $\{U_1,U_2,U_3,U_4\}$ is a single cluster, create its parent W_1=mean($\{U_1,U_2,U_3,U_4\}$)



CODEBOOK STRUCTURE (m-ARY TREES)

-- HOW TO SEARCH IN m-ARY TREES --

- 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



Find the best match of v among $\{U_1, U_2, U_3, U_4\}$: U_2

Find the best match of v among $\{V_5, V_6, V_7, V_8\}$: V_7

- The search time is logarithmic (rather than linear) in the codebook size
- The match found is not always the minimum-MSE match, but still a good match

REFINED TREES

- In addition to m-ray trees, two other kinds of trees have been developed
  - **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 (1/5)

- One can use vector quantization in more sophisticated/advanced ways than the way presented so far
- The following are four different advanced ways:
  - Prediction/Residual VQ (P/R VQ)
  - Mean/Residual VQ (M/R VQ)
  - Interpolation/residual VQ (I/R VQ)
  - Gain/Shape VQ (G/S VQ)
- They are addressed briefly next

ADVANCED VQ (2/5)
-- PREDICTION/RESIDUAL VQ --

- Prediction/Residual VQ (P/R VQ): For each vector v to be coded do
  1. Predict vector v, i.e., calculate a prediction/estimate $\hat{v}$ of v
  2. compute the residual vector $e = v - \hat{v}$;
  3. VQ-Code the residual vector e
- Advantages:
  - The original vectors v's exhibit so large a variety that a large codebook is needed to be representative; otherwise, a smaller codebook will lead to poor reconstruction
  - The residual vectors, on the other hand, exhibit a much smaller range of variety;
    - therefore, a smaller codebook is enough for those residual vectors
  - The better the prediction model, the smaller (and less varied) the residual vectors are, leading to better reconstruction quality and/or smaller codebooks
  - Shared residual-codebooks are better than shared original codebooks, again because residuals exhibit much less variety across images and across applications
    - => shared residual codebooks yield low bitrates without the quality penalty

ADVANCED VQ (3/5)
-- MEAN/RESIDUAL VQ --

- Mean/Residual VQ (M/R VQ): For each vector v to be coded do
  1. Compute the mean of v and subtract it from v: $e = v - mean(v)$
  2. VQ-code the residual vector e
  3. 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
- Therefore, this approach has some of the advantages of the previous approach (Prediction/Residual VQ)
- **Exercise**: Show that the M/R VQ is a special case of P/R VQ
- **Exercise**: Which is better, M/R VQ or P/R VQ, and why?

ADVANCED VQ (4/5)
-- INTERPOLATION/RESIDUAL VQ --

- Interpolation/residual VQ (I/R VQ)
  1. subsample the image by choosing every l^{th} pixel
  2. code the subsampled image using scalar quantization
  3. Upsample the image using bilinear interpolation
  4. VQ-code the residual (i.e., original-upsampled) image
- Remark: Residuals have fewer variations, leading to smaller codebooks
- Therefore, this approach has some of the advantages of the previous approaches (P/R and M/R VQ)

ADVANCED VQ (5/5)
-- GAIN/SHAPE VQ --

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

- $v = (v_1, v_2, \dots, v_n)$
- $m = mean(v) = \frac{1}{n}\sum_{i=1}^{n} v_i$
- $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - m)^2}$
- Normalized(v)= $\frac{v}{\sigma}$

- One can also apply M/R and G/S VQ, making all the vectors 0-mean 1-variance vectors (i.e., mean-normalized and gain-normalized)
- That would yield even less varied residuals, leading to even smaller codebooks

[ADVANCED] VQ VS. TRANSFORM+SCALAR QUANTIZATION

- While VQ preserves correlations, studies and experiments have shown that the resulting compression ratios for decent reconstruction quality is quite modest (4-8)
- That is inadequate, and much smaller than DCT-based/JPEG compression.
  - Can you reason why? What property(ies) of transforms are lacking in VQ?
- Therefore, VQ (including advanced VQ) is rarely used, and only in specific situations
  - For example, in the audio portion of MPEG, where window-based parameterized modeling of audio signals is employed, VQ is used to code the parameters
- Nevertheless, VQ development was instructive

NEXT LECTURE

- Filtering
- Subband coding