Lossy Image Compression Methods

- DCT Compression
  - JPEG
- Scalar quantization (SQ)
- Vector quantization (VQ)
- Wavelet Compression
  - SPIHT
  - GTW
  - EBCOT
  - JPEG 2000

Scalar Quantization

\[ D = \sum_{i=1}^{T} (x_i - c(\text{index}(x_i)))^2 \quad \text{(Distortion)} \]

\[ \text{MSE} = \frac{D}{T} \]

Scalar Quantization Strategies

- Build a codebook with a training set. Encode and decode with fixed codebook.
  - Most common use of quantization
- Build a codebook for each image. Transmit the codebook with the image.
  - Training can be slow.

Uniform Quantization Example

- 512 x 512 image with 8 bits per pixel.
- 8 codewords

<table>
<thead>
<tr>
<th>Index</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>79</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
</tr>
<tr>
<td>4</td>
<td>143</td>
</tr>
<tr>
<td>5</td>
<td>175</td>
</tr>
<tr>
<td>6</td>
<td>207</td>
</tr>
<tr>
<td>7</td>
<td>239</td>
</tr>
</tbody>
</table>
Uniform Quantization Example

Encoder

<table>
<thead>
<tr>
<th>input</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-31</td>
<td>000</td>
</tr>
<tr>
<td>32-63</td>
<td>001</td>
</tr>
<tr>
<td>64-95</td>
<td>010</td>
</tr>
<tr>
<td>96-127</td>
<td>011</td>
</tr>
<tr>
<td>128-159</td>
<td>100</td>
</tr>
<tr>
<td>160-191</td>
<td>101</td>
</tr>
<tr>
<td>192-223</td>
<td>110</td>
</tr>
<tr>
<td>224-255</td>
<td>111</td>
</tr>
</tbody>
</table>

Decoder

<table>
<thead>
<tr>
<th>code</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>111</td>
<td>20</td>
</tr>
</tbody>
</table>

Bit rate = 3 bits per pixel
Compression ratio = 8/3 = 2.67

Alternative Uniform Quantization
Calculation with Push to Zero

- Range = [min, max)
- Target is S symbols
- Choose Q = (max – min)/S

Encode x

\[ s = \left\lfloor \frac{x}{Q} + \frac{1}{2} \right\rfloor \]

Decode s

\[ x' = s \times Q \]

Improving Bit Rate

Frequency of pixel values

\[ q_j = \text{the probability that a pixel is coded to index } j \]

Potential average bit rate is entropy.

\[ H = \sum_j q_j \log_2 \left( \frac{1}{q_j} \right) \]

Example

- [0,100) with 5 symbols
- Q = 20

Encode

<table>
<thead>
<tr>
<th>input</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-31</td>
<td>0</td>
</tr>
<tr>
<td>32-63</td>
<td>1</td>
</tr>
<tr>
<td>64-95</td>
<td>10</td>
</tr>
<tr>
<td>96-127</td>
<td>11</td>
</tr>
<tr>
<td>128-159</td>
<td>101</td>
</tr>
<tr>
<td>160-191</td>
<td>110</td>
</tr>
<tr>
<td>192-223</td>
<td>111</td>
</tr>
</tbody>
</table>

Decode

<table>
<thead>
<tr>
<th>code</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>111</td>
<td>20</td>
</tr>
</tbody>
</table>

Example

- [0,90) with 5 symbols
- Q = 20

Encode

<table>
<thead>
<tr>
<th>input</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-31</td>
<td>0</td>
</tr>
<tr>
<td>32-63</td>
<td>1</td>
</tr>
<tr>
<td>64-95</td>
<td>10</td>
</tr>
<tr>
<td>96-127</td>
<td>11</td>
</tr>
<tr>
<td>128-159</td>
<td>101</td>
</tr>
<tr>
<td>160-191</td>
<td>110</td>
</tr>
<tr>
<td>192-223</td>
<td>111</td>
</tr>
</tbody>
</table>

Decode

<table>
<thead>
<tr>
<th>code</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>111</td>
<td>20</td>
</tr>
</tbody>
</table>

Example

- 512 x 512 image = 216,144 pixels

<table>
<thead>
<tr>
<th>index</th>
<th>input frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-31</td>
<td>25,000</td>
</tr>
<tr>
<td>32-63</td>
<td>100,000</td>
</tr>
<tr>
<td>64-95</td>
<td>90,000</td>
</tr>
<tr>
<td>96-127</td>
<td>10,000</td>
</tr>
<tr>
<td>128-159</td>
<td>10,000</td>
</tr>
<tr>
<td>160-191</td>
<td>10,000</td>
</tr>
<tr>
<td>192-223</td>
<td>10,000</td>
</tr>
<tr>
<td>224-255</td>
<td>9,144</td>
</tr>
</tbody>
</table>

Potential average bit rate

\[ H = \sum_{i=0}^{7} q_i \log_2 \left( \frac{1}{q_i} \right) \]

Potential average bit rate is entropy.

\[ H = \sum_{i=0}^{7} q_i \log_2 \left( \frac{1}{q_i} \right) \]

Arithmetic coding should work better.
Improving Distortion

• Choose the codeword as a weighted average

Let $p_x$ be the probability that a pixel has value $x$. Let $[L_j, R_j]$ be the input interval for index $j$. $c(j)$ is the codeword indexed $j$.

$$c(j) = \text{round} \left( \sum_{x \in [L_j, R_j]} x p_x \right)$$

Example

All pixels have the same index.

<table>
<thead>
<tr>
<th>pixel value</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

Old Codeword: $200$  
Old Distortion: $10000$

New Codeword: $11$  
New Distortion: $1000$

An Extreme Case

Only two codewords are ever used!!

Non-uniform Scalar Quantization

Lloyd Algorithm

• Lloyd (1957)
• Creates an optimized codebook of size $n$.
• Let $p_x$ be the probability of pixel value $x$.
  – Probabilities might come from a training set
• Given codewords $c(0), c(1), \ldots, c(n-1)$ and pixel $x$ let $\text{index}(x)$ be the index of the closest code word to $x$.
• Expected distortion is

$$D = \sum_x p_x (x - c(\text{index}(x)))^2$$

• Goal of the Lloyd algorithm is to find the codewords that minimize distortion.
• Lloyd finds a local minimum by an iteration process.

Lloyd Algorithm

Choose a small error tolerance $\epsilon > 0$.  
Choose start codewords $c(0), c(1), \ldots, c(n-1)$  
Compute $X(j) := \{x : x$ is a pixel value closest to $c(j)\}$  
Compute distortion $D$ for $c(0), c(1), \ldots, c(n-1)$  
Repeat
  Compute new codewords
  $$c'(j) = \text{round} \left( \sum_{x \in X(j)} x p_x \right)$$
  Compute $X'(j) = \{x : x$ is a pixel value closest to $c'(j)\}$
  Compute distortion $D'$ for $c'(0), c'(1), \ldots, c'(n-1)$
  if $|D - D'|/D < \epsilon$ then quit
  else $c := c'$, $X := X'$, $D := D'$
End[repeat]
Example
Initially \( c(0) = 2 \) and \( c(1) = 5 \)

<table>
<thead>
<tr>
<th>pixel value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
X(0) = [0.3, X(1) = [4, 7]
D(0) = 140 1^2 + 100 2^2 + 540 D(1) = 40 1^2 = 40
\]

\[
D = D(0) + D(1) = 580
\]

\[
c(0) = \text{round}(100 \times 0.1) = 100 1 \times 100 2 \times 40 3 \times 240 = 1
\]

\[
c(1) = \text{round}(40 3 \times 30 + 40 5 \times 10 + 6 \times 0 7 \times 100) = 5
\]

Example

<table>
<thead>
<tr>
<th>pixel value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
c(0) = 1; c(1) = 5
\]

\[
X(0) = [0.2, X(1) = [3, 7]
D(0) = 200 1^2 = 200
\]

\[
D(1) = 40 1^2 + 40 2^2 = 200
\]

\[
D = D(0) + D(1) = 400
\]

\[
c = c' X = X' D = D'
\]

Example

<table>
<thead>
<tr>
<th>pixel value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
c(0) = 1; c(1) = 4
\]

\[
X(0) = [0.2, X(1) = [3, 7]
D(0) = 200 1^2 = 200
\]

\[
D(1) = 60 1^2 + 10 2^2 = 100
\]

\[
D = D(0) + D(1) = 300
\]

\[
c = c' X = X' D = D'
\]

Example

<table>
<thead>
<tr>
<th>pixel value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
c(0) = 1; c(1) = 4
\]

\[
X(0) = [0.2, X(1) = [3, 7]
D(0) = 200 1^2 = 200
\]

\[
D(1) = 60 1^2 + 10 2^2 = 100
\]

\[
D = D(0) + D(1) = 300
\]

\[
\text{Exit with codeword } c(0) = 1\text{ and } c(1) = 4.
\]
Scalar Quantization Notes

- Useful for analog to digital conversion.
- Useful for estimating a large set of values with a small set of values.
- With entropy coding yields good lossy compression.
- Lloyd algorithm works very well in practice, but can take many iterations.
  - For n codewords should use about 20n size representative training set.
  - Imagine 1024 codewords.

Vector Quantization

vector: An \( a \times b \) block can be considered to be a vector of dimension \( ab \).

\[
\begin{align*}
\text{block} & \quad = \quad (w, x, y, z) \text{ vector} \\
\text{Nearest means in terms of Euclidian distance or Euclidian squared distance. Both equivalent.} \\
\text{Distance} & \quad = \quad \sqrt{(w - W_i)^2 + (x - X_i)^2 + (y - Y_i)^2 + (z - Z_i)^2} \\
\text{Squared Distance} & \quad = \quad (w - W_i)^2 + (x - X_i)^2 + (y - Y_i)^2 + (z - Z_i)^2 \\
\text{Squared distance is easier to calculate.}
\end{align*}
\]

Vector Quantization Facts

- The image is partitioned into \( a \times b \) blocks.
- The codebook has \( n \) representative \( a \times b \) blocks called codewords, each with an index.
- Compression with fixed length codes is
  \[
  \log_2 \frac{ab}{n} \text{ bpp}
  \]
- Example: \( a = b = 4 \) and \( n = 1,024 \)
  - compression is \( 10/16 = .63 \text{ bpp} \)
  - compression ratio is \( 8 : .63 = 12.8 : 1 \)
- Better compression with entropy coding of indices

Examples

- 4 x 4 blocks: .63 bpp
- 4 x 8 blocks: .31 bpp
- 8 x 8 blocks: .16 bpp

Codebook size = 1,024

Scalar vs. Vector

- Pixels within a block are correlated.
  - This tends to minimize the number of codewords needed to represent the vectors well.
- More flexibility.
  - Different size blocks
  - Different size codebooks
Encoding and Decoding

- **Encoding**:
  - Scan the $a \times b$ blocks of the image. For each block find the nearest codeword in the codebook and output its index.
  - Nearest neighbor search.
- **Decoding**:
  - For each index output the codeword with that index into the destination image.
  - Table lookup.

The Codebook

- Both encoder and decoder must have the same codebook.
- The codebook must be useful for many images and be stored someplace.
- The codebook must be designed properly to be effective.
- Design requires a representative training set.
- These are major drawbacks to VQ.

Codebook Design Problem

- Input: A training set $X$ of vectors of dimension $d$ and a number $n$ ($d = a \times b$ and $n$ is number of codewords)
- Output: $n$ codewords $c(0), c(1), \ldots, c(n-1)$ that minimize the distortion.

$$
D = \sum_{x \in X} \| x - c(\text{index}(x)) \|^2
$$

where $\text{index}(x)$ is the index of the nearest codeword to $x$.

$$
\| X_0, X_1, \ldots, X_{d-1} \|_2 = x_0^2 + x_1^2 + \cdots + x_{d-1}^2
$$

GLA

- The Generalized Lloyd Algorithm (GLA) extends the Lloyd algorithm for scalars.
  - Also called LBG after inventors Linde, Buzo, Gray (1980)
- It can be very slow for large training sets.

GLA Example (1)

Choose a training set $X$ and small error tolerance $\epsilon > 0$.
Choose start codewords $c(0), c(1), \ldots, c(n-1)$
Compute $X(j) := \{ x : x \text{ is a vector in } X \text{ closest to } c(j) \}$
Compute distortion $D$ for $c(0), c(1), \ldots, c(n-1)$
Repeat
  Compute new codewords $c'(j) := \text{round}\left( \frac{1}{|X(j)|} \sum_{x \in X(j)} x \right)$ (centroid)
  Compute $X'(j) := \{ x : x \text{ is a vector in } X \text{ closest to } c'(j) \}$
  Compute distortion $D'$ for $c'(0), c'(1), \ldots, c'(n-1)$
  if $|D - D'| < \epsilon$ then quit
  else $c := c'$; $X := X'$; $D := D'$
End(repeat)
Codeword Splitting

- It is possible that a chosen codeword represents no training vectors, that is, \(X(j)\) is empty.
  - Splitting is an alternative codebook design algorithm that avoids this problem.
- Basic Idea
  - Select codeword \(c(j)\) with the greatest distortion.
    \[
    D(j) = \sum_{i=1}^{m} ||x_i - c||^2
    \]
  - Split it into two codewords then do the GLA.
Example of Splitting

- **Codeword**: c(0)
- **Training Vector**: c(0) + \( \epsilon \)

- **Split**: c(1) = c(0) + \( \epsilon \)

---

Example of Splitting

- **Codeword**: c(0)
- **Training Vector**: c(0)
- **Apply GLA**: c(2)

- **X(0)**
- **X(1)**

---

Example of Splitting

- **Codeword**: c(0)
- **Training Vector**: c(0)
- **Apply GLA**: c(3)

- **X(0)**
- **X(1)**
- **X(2)**

---

Example of Splitting

- **Codeword**: c(2)
- **Training Vector**: c(2)
- **Apply GLA**: c(3)

- **X(0)**
- **X(1)**
- **X(2)**
- **X(3)**
GLA Advice
• Time per iteration is dominated by the partitioning step, which is \( m \) nearest neighbor searches where \( m \) is the training set size.
  – Average time per iteration \( O(m \log n) \) assuming \( d \) is small.
• Training set size.
  – Training set should be at least 20 training vectors per code word to get reasonable performance.
  – Too small a training set results in “over training”.
• Number of iterations can be large.

Encoding
• Naive method.
  – For each input block, search the entire codebook to find the closest codeword.
  – Time \( O(Tn) \) where \( n \) is the size of the codebook and \( T \) is the number of blocks in the image.
  – Example: \( n = 1024, T = 256 \times 256 = 65,536 \) (2 \( \times \) 2 blocks for a 512 \( \times \) 512 image)
  \( nT = 1024 \times 65536 = 2^{26} \approx 67 \) million distance calculations.
• Faster methods are known for doing “Full Search VQ”. For example, k-d trees.
  – Time \( O(T \log n) \)

VQ Encoding is Nearest Neighbor Search
• Given an input vector, find the closest codeword in the codebook and output its index.
• Closest is measured in squared Euclidian distance.
• For two vectors \((w_1, x_1, y_1, z_1)\) and \((w_2, x_2, y_2, z_2)\):
  \[
  \text{Squared Distance} = (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2
  \]

k-d Tree
• Jon Bentley, 1975
• Tree used to store spatial data.
  – Nearest neighbor search.
  – Range queries.
  – Fast look-up
• k-d tree are guaranteed \( \log_2 n \) depth where \( n \) is the number of points in the set.
  – Traditionally, k-d trees store points in \( d \)-dimensional space which are equivalent to vectors in \( d \)-dimensional space.

k-d Tree Construction
• If there is just one point, form a leaf with that point.
• Otherwise, divide the points in half by a line perpendicular to one of the axes.
• Recursively construct k-d trees for the two sets of points.
• Division strategies
  – divide points perpendicular to the axis with widest spread.
  – divide in a round-robin fashion.

divide perpendicular to the widest spread.
k-d Tree Construction (2)

k-d Tree Construction (3)

k-d Tree Construction (4)

k-d Tree Construction (5)

k-d Tree Construction (6)

k-d Tree Construction (7)
k-d Tree Construction Complexity

- First sort the points in each dimension.
  - $O(dn \log n)$ time and $dn$ storage.
  - These are stored in $A[1..d,1..n]$
- Finding the widest spread and equally dividing into two subsets can be done in $O(dn)$ time.
- Constructing the k-d tree can be done in $O(dn \log n)$ and $dn$ storage.
k-d Tree Codebook Organization

2-d vectors (x, y)

Node Structure for k-d Trees

- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)

k-d Tree Nearest Neighbor Search

\[
\text{NNS}(q, \text{root}, p, \infty) \quad \text{initial call}
\]

\[
\text{if \ } n.left = n.right = \text{null then (leaf case)}
\]

\[
\text{w}^* := ||q - n.point||;
\]

\[
\text{if } w^* < w \text{ then } w := w^*; p := n.point;
\]

\[
\text{else}
\]

\[
\text{if } w = \infty \text{ then}
\]

\[
\text{if } q(n.axis) \leq n.value \text{ then}
\]

\[
\text{NNS}(q, n.left, p, w);
\]

\[
\text{if } q(n.axis) + w > n.value \text{ then NNS}(q, n.right, p, w);
\]

\[
\text{else}
\]

\[
\text{NNS}(q, n.right, p, w);
\]

\[
\text{else (w is finite)}
\]

\[
\text{if } q(n.axis) - w < n.value \text{ then NNS}(q, n.left, p, w);
\]

\[
\text{else (w is finite)}
\]

\[
\text{if } q(n.axis) - w < n.value \text{ then NNS}(q, n.left, p, w);
\]

\[
\text{if } q(n.axis) + w > n.value \text{ then NNS}(q, n.right, p, w);
\]

Explanation

- \( q(n.axis) - w < n.value \) means the circle overlaps the left subtree.
- \( q(n.axis) + w > n.value \) means the circle overlaps the right subtree.

k-d Tree NNS (1)

- query point

k-d Tree NNS (2)

- query point
k-d Tree NNS (15)

k-d Tree NNS (16)

k-d Tree NNS (17)

k-d Tree NNS (18)

k-d Tree NNS (19)

k-d Tree NNS (20)
Notes on k-d Tree NNS

- Has been shown to run in \(O(\log n)\) average time per search in a reasonable model. (Assume \(d\) a constant)
- For VQ it appears that \(O(\log n)\) is correct.
- Storage for the k-d tree is \(O(n)\).
- Preprocessing time is \(O(n \log n)\) assuming \(d\) is a constant.

Alternatives

- Orchard’s Algorithm (1991)
  - Uses \(O(n^2)\) storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses \(O(n)\) storage. Does many more distance calculations.
- PCP Principal Component Partitioning
  - Zatloukal, Johnson, Ladner (1999)
  - Similar to k-d trees
  - Also very fast

Principal Component Partition

Comparison in Time per Search

4,096 codewords
Notes on VQ

- Works well in some applications.
  - Requires training
- Has some interesting algorithms.
  - Codebook design
  - Nearest neighbor search
- Variable length codes for VQ.
  - PTSVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  - ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)