Image Compression-III

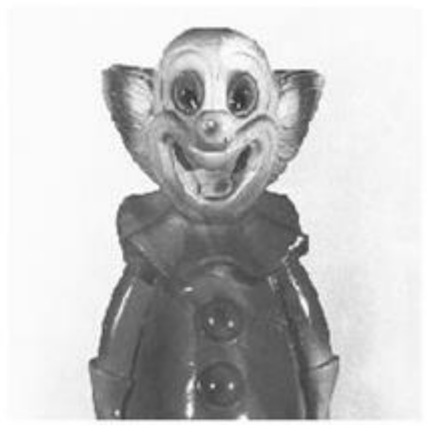
Contents

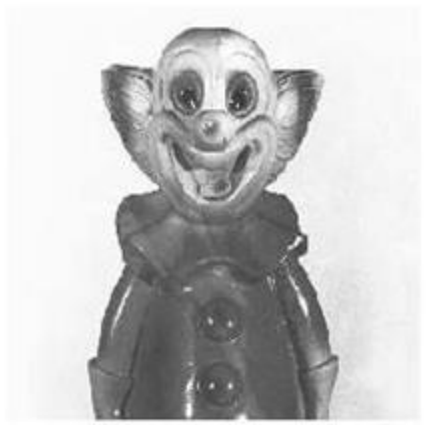
Lossy Compression Methods

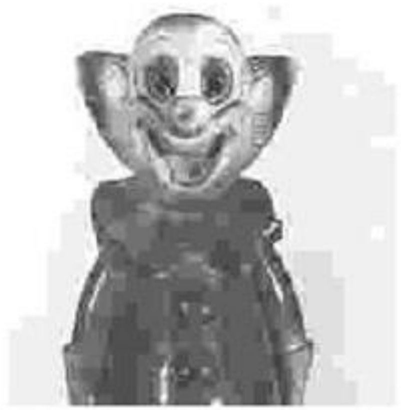
Lossy compression methods are required to achieve <u>high compression ratios</u> with complex images.

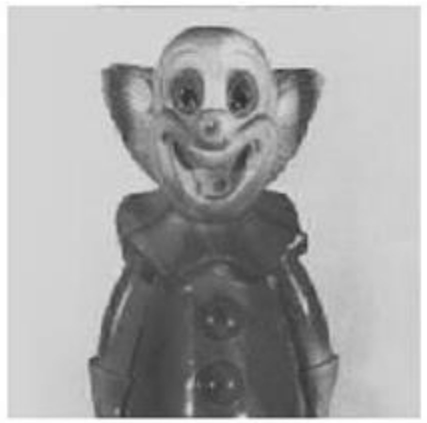
They provide <u>adjustments between image quality and degree of compression</u>, which allows the compression algorithm to be customized to the application.

Lossy Image Compression


a) Original image


b) JPEG compression, 10:1 ratio


c) JPEG compression, 48:1 ratio


d) Wavelet/vector quantization compression, 36:1 ratio

With more advanced methods, images can be compressed 10 to 20 times with virtually no visible information loss, and 30 to 50 times with minimal degradation.

Newer techniques, such as JPEG2000, can achieve reasonably good image quality with compression ratios as high as 100 to 200.

Image enhancement and restoration techniques can be combined with lossy compression schemes to improve the appearance of the decompressed image.

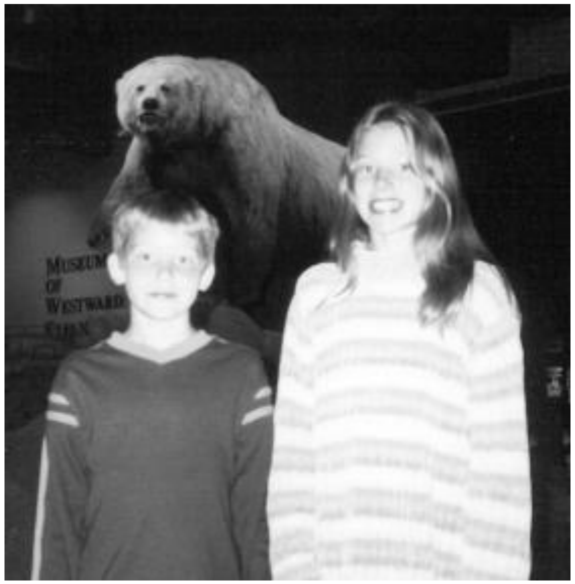
In general, a <u>higher compression ratio results in a poorer image</u>, but the results are highly image dependent – application specific.

Lossy compression can be performed in both the spatial and transform domains. Hybrid methods use both domains.

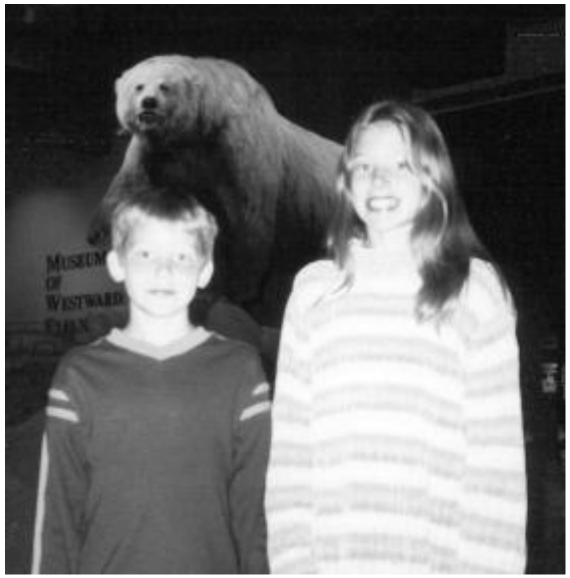
Gray-Level Run Length Coding

The RLC technique can also be used for lossy image compression, by reducing the number of graylevels, and then applying standard RLC techniques.

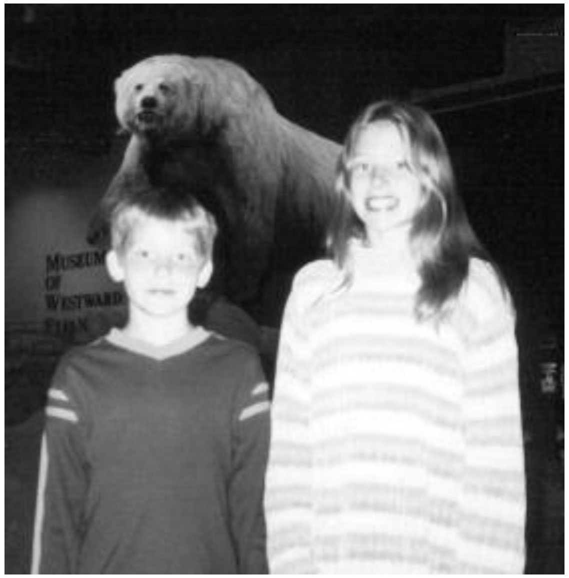
As with the lossless techniques, preprocessing technique will improve the compression ratio.



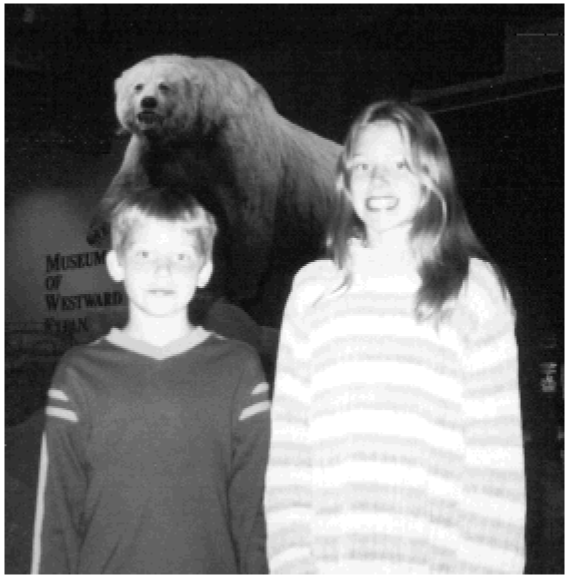
a) Original image, 8 bits/pixel, 256 gray levels



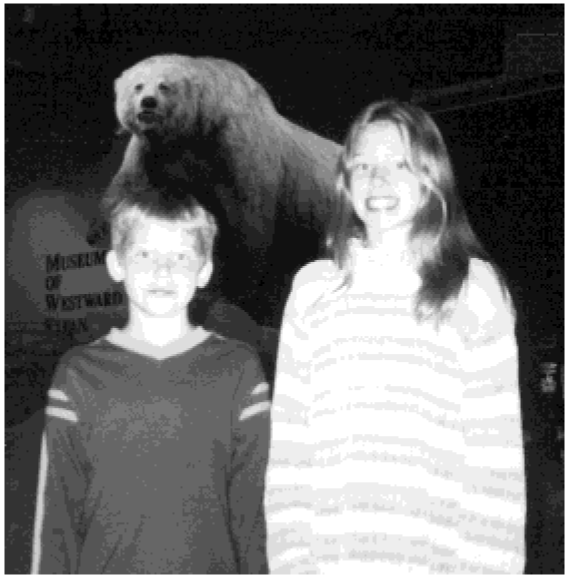
b) Image after reduction to 7 bits/pixel, 128 gray levels, compression ratio 0.55, with preprocessing 0.66



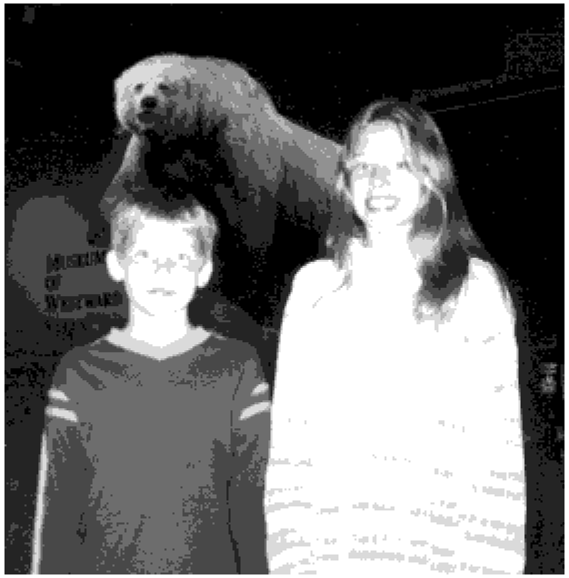
c) Image after reduction to 6 bits/pixel, 64 gray levels, compression ratio 0.77, with preprocessing 0.97



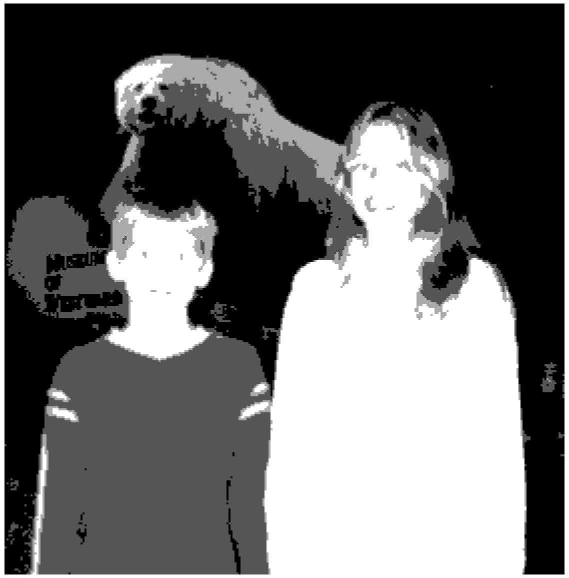
d) Image after reduction to 5 bits/pixel, 32 gray levels, compression ratio 1.20, with preprocessing 1.60

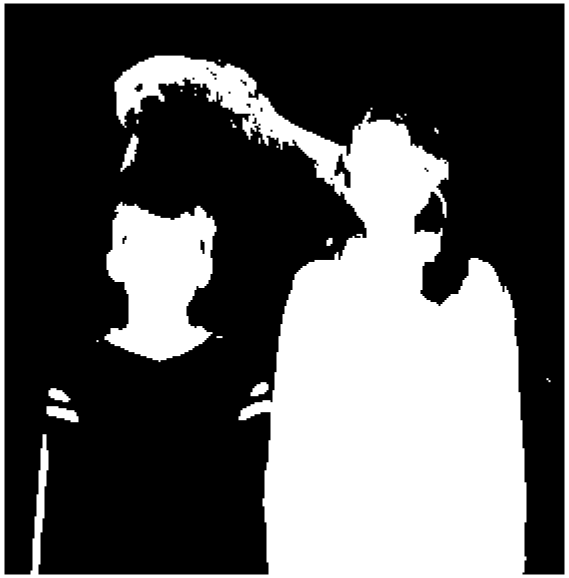


e) Image after reduction to 4 bits/pixel,
16 gray levels, compression ratio 2.17,
with preprocessing 2.79



f) Image after reduction to 3 bits/pixel,
8 gray levels, compression ratio 4.86,
with preprocessing 5.82


g) Image after reduction to 2 bits/pixel, 4 gray levels, compression ratio 13.18, with preprocessing 15.44


h) Image after reduction to 1 bit/pixel, 2 gray levels, compression ratio 44.46, with preprocessing 44.46

Block Truncation Coding

Block Truncation Coding

Block Truncation Coding (BTC) technique is based on preserving the first and second statistical moments of the image.

The BTC method was first presented in a paper by Delp and Mitchell in 1979.

Block Truncation Coding

BTC works by dividing the image into small subimages (blocks) and then reducing the number of graylevels within each block.

The graylevels are reduced by a quantizer that adapts to local statistics.

The levels for the quantizer are chosen to minimize a specified error criteria, and then all the pixel values within each block are mapped to the quantized levels.

The necessary information to decompress the image is then encoded and stored.

The basic form of BTC divides the image into N×N blocks and codes each block using a two-level quantizer.

The two levels are selected so that the mean and variance of the gray levels within the block are preserved.

Each pixel value within the block is then compared with a threshold, typically the block mean, and then is assigned to one of the two levels.

If it is above the mean it is assigned the high level code, if it is below the mean, it is assigned the low level code.

If we call the high value *H* and the low value *L*, we can find these values via the following equations:

$$H = m_b + \sigma_b \sqrt{\frac{n^2 - q}{q}}$$

$$L = m_b - \sigma_b \sqrt{\frac{q}{n^2 - q}}$$

Where the block size is $n \times n$

$b = \textit{the current block}$

$m_b = \textit{the block mean} = \frac{1}{n^2} \sum_{I(r,c)\in b} I(r,c)$

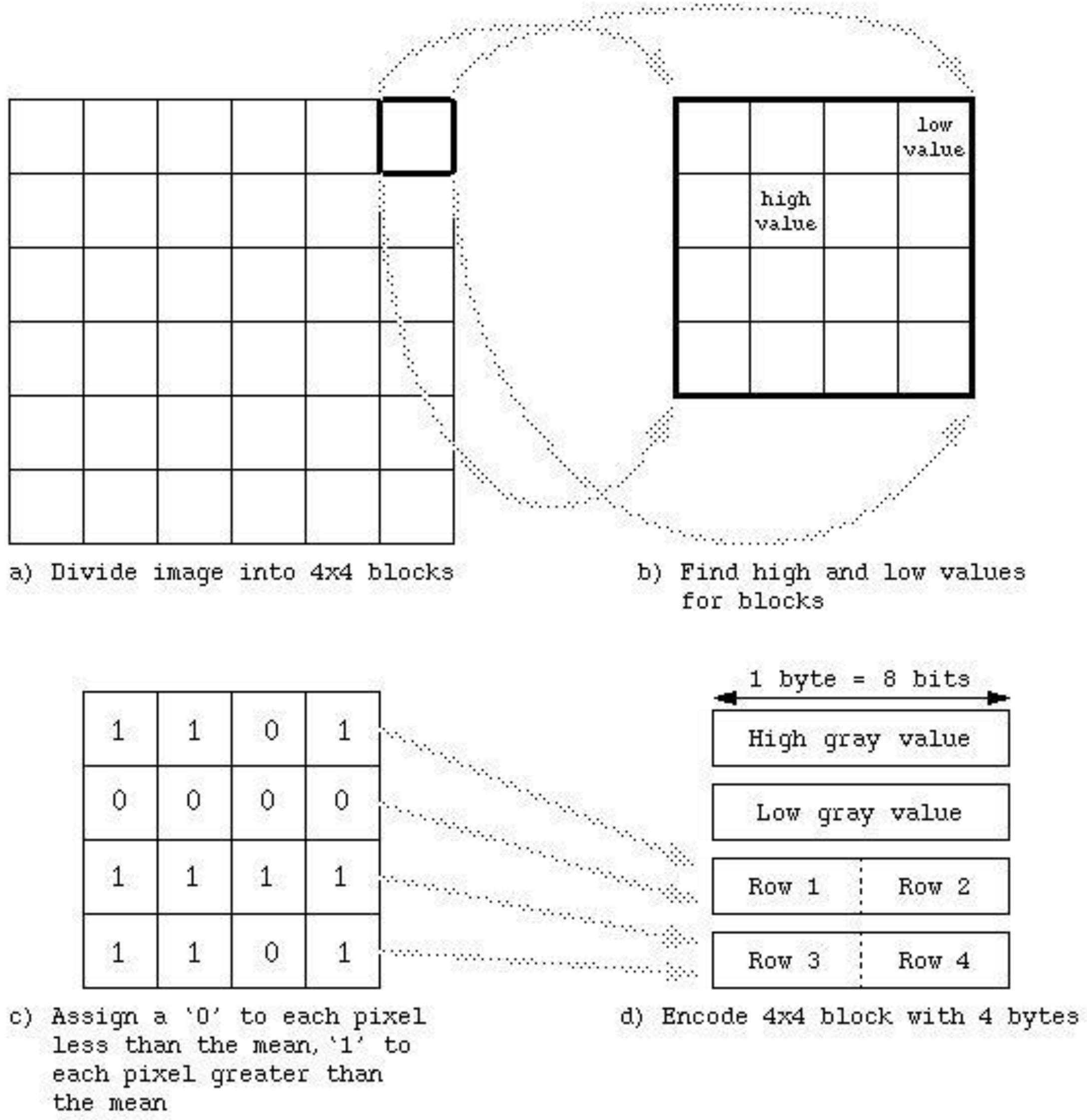
$\sigma_b = \text{the block standard deviaiton} = \sqrt{\frac{1}{n^2} \sum_{I(r,c)\in b} [I(r,c)]^2 - m_b^2}$

$q = \textit{the number of values in the block} \geq m_b$

If *n = 4*, then after the *H* and *L* values are found, the 4x4 block is encoded with four bytes.

Two bytes to store the two levels, *H* and *L*, and two bytes to store a bit string of 1's and 0's corresponding to the high and low codes for that particular block.

Figure 10.3-4: Basic Block Truncation Coding


low value
high value
a) Divide image into 4x4 blocks
b) Find high and low values for blocks
1 1 0 1
0 0 0 0
1 1 1 1
1 1 0 1
c) Assign a '0' to each pixel less than the mean, '1' to each pixel greater than the mean
1 byte = 8 bits
High gray value
Low gray value
Row 1
Row 2
Row 3
Row 4
d) Encode 4x4 block with 4 bytes

EXAMPLE 10.3.2:

Given the following 4x4 subimage, apply basic BTC and find the resulting values.

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix}$$

$$m_b = \frac{1}{n^2} \sum_{I(r,c)\in b} I(r,c) = \frac{1}{16}\left[12+16+15+17+13+16+17+17+4+4+35+35+42+42+12+12\right]$$

$$= 19.3125$$

$$\sigma_b = \sqrt{\frac{1}{n^2} \sum_{I(r,c)\in b} [I(r,c)]^2 - m_b^2}$$

$$= \sqrt{\frac{1}{16}\left[12^2+16^2+15^2+17^2+13^2+16^2+17^2+17^2+4^2+4^2+35^2+35^2+42^2+42^2+12^2+12^2\right]-(19.3125)^2}$$

$$\approx 11.85$$

EXAMPLE 10.3.2 (contd):

There are 4 pixels values greater than the mean, so q = 4.

$$H = m_b + \sigma_b\sqrt{\frac{n^2-q}{q}} = 19.3125 + 11.85\sqrt{\frac{16-4}{4}} \approx 40$$

$$L = m_b - \sigma_b\sqrt{\frac{q}{n^2-q}} = 19.3125 - 11.85\sqrt{\frac{4}{16-4}} \approx 13$$

Now, find the bit string by using 0 for values less than the mean and 1 for values greater than the mean:

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \Rightarrow (0000000000111100_2)$$

The high value, H, and the low value, L, will be stored along with the bit string. The subimage, when decompressed will be:

$$\begin{bmatrix} 13 & 13 & 13 & 13 \\ 13 & 13 & 13 & 13 \\ 13 & 13 & 40 & 40 \\ 40 & 40 & 13 & 13 \end{bmatrix}$$

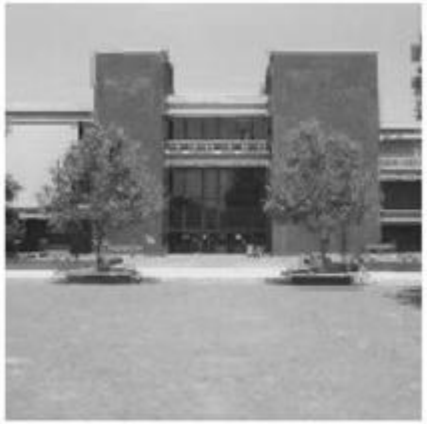
This algorithm tends to produce images with blocky effects.

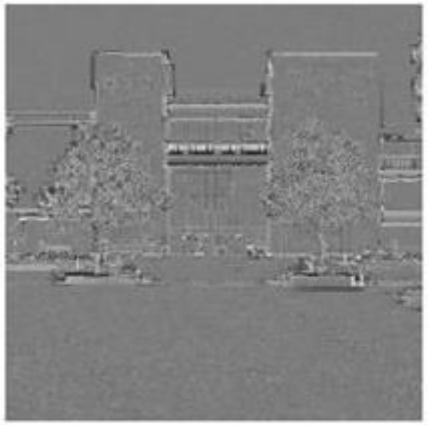
These artifacts can be smoothed by applying enhancement techniques such as median and average (lowpass) filters.

BTC lends itself very nicely to parallel processsing since the coding of each of the blocks is totally independent.

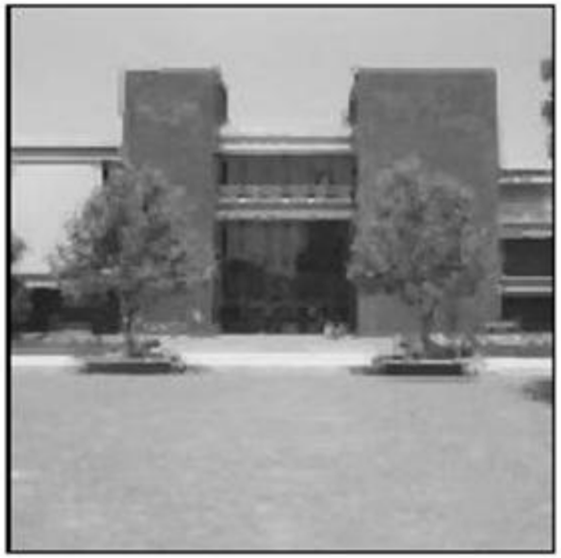
Block Truncation Coding (BTC)

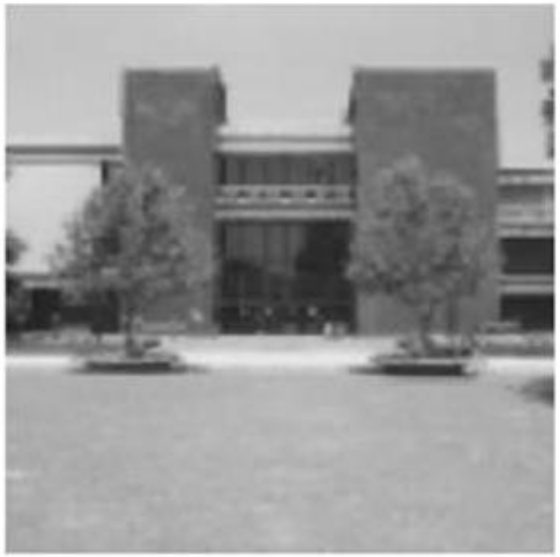
a) Original image


b) Block truncation coded image, compression = 4:1


c) Error image of (b), histogram stretched to show detail

continued

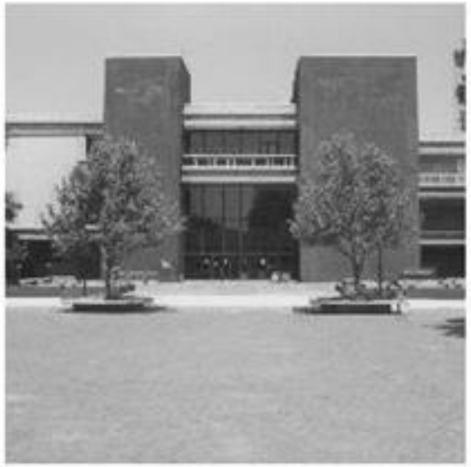

d) Image (b) post-processed with a 3x3 median filter


e) Image (b) post-processed with a 3x3 averaging filter

The multilevel BTC algorithm, which uses a 4-level quantizer, allows for varying the block size, and a larger block size should provide higher compression, but with a corresponding decrease in image quality.

Multilevel Block Truncation Coding

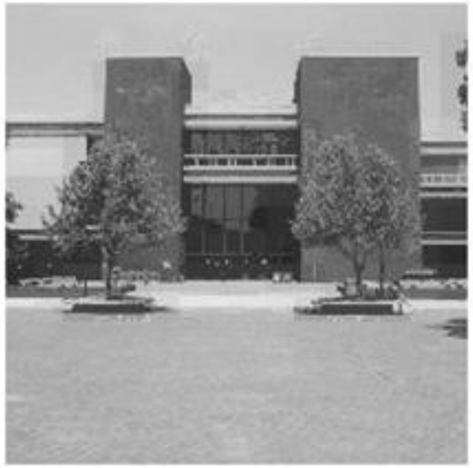
a) Original image



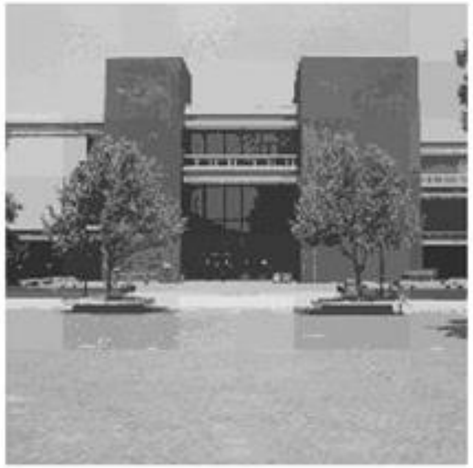
b) Multilevel BTC,
block size = 8x8

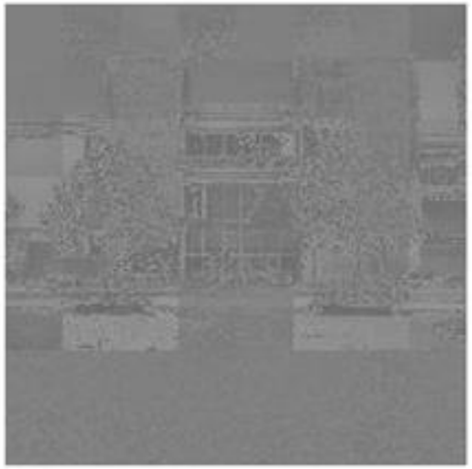
c) Error image of (b),
multiplied to show detail

continued


d) Multilevel BTC,
block size = 16x16,

e) Error image of (d)
multiplied to show detail


f) Multilevel BTC,
block size = 32x32


g) Error image of (f)
multiplied to show detail

Vector Quantization

Vector Quantization

Vector quantization (VQ) is the process of mapping a vector that can have many values to a vector that has a smaller (quantized) number of values.

For image compression, the vector corresponds to a small subimage or block.

EXAMPLE 10.3.3:

Given the following 4x4 subimage:

$$\begin{bmatrix} 65 & 70 & 71 & 75 \\ 71 & 70 & 71 & 81 \\ 81 & 80 & 81 & 82 \\ 90 & 90 & 91 & 92 \end{bmatrix}$$

This can be re-arranged into a 1-D vector by putting the rows adjacent as follows:

[row1 row2 row3 row4] = [65 70 71 75 71 70 71 81 81 80 81 82 90 90 91 92]

VQ can be applied in both the spectral or spatial domains.

Information theory tells us that better compression can be achieved with vector quantization than with scalar quantization (rounding or truncating individual values).

Vector quantization treats the entire subimage (vector) as a single entity and quantizes it by reducing the total number of bits required to represent the subimage.

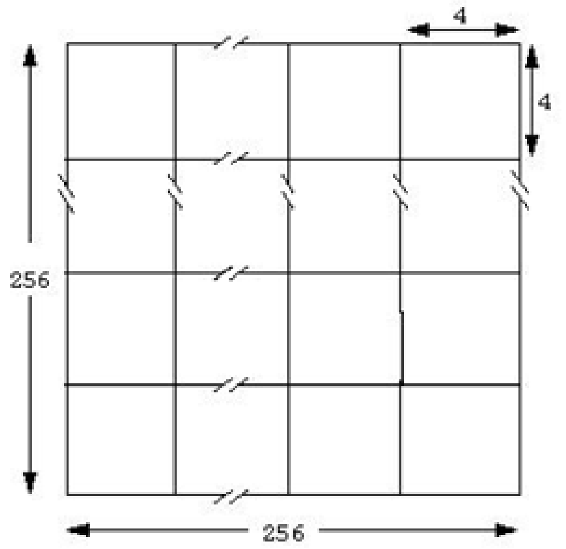
This is done by utilizing a codebook, which stores a fixed set of vectors, and then coding the subimage by using the index (address) into the codebook.

Example

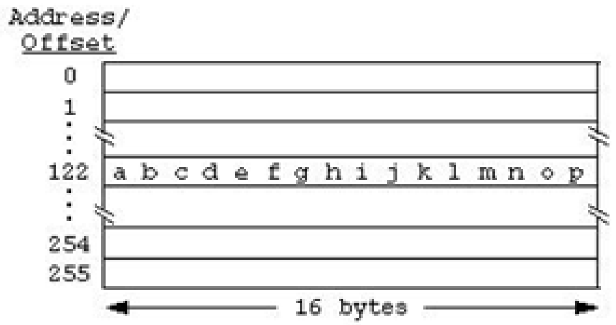
Given an 8-bit 256×256 image, we device a vector quantization scheme that will encode each 4×4 block with one of the vectors in a codebook of 256 entries.

We determine that we want to encode a specific subimage with vector number 122 in the codebook. For this subimage we then store the number 122 as the index into the codebook.

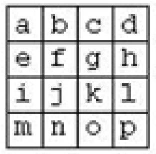
Then when the image is decompressed, the vector at the 122 address of the codebook, is used for that particular subimage.



a) Original 256x256 image divided into 4x4 blocks



b) Codebook with 256 16-byte entries



c) A subimage decompressed with vector #122

Compression ratio is 16:1

In the example we achieved a 16:1 compression, but note that this assumes that the codebook is not stored with the compressed file.

Codebook <u>needs to be stored</u> unless a <u>generic codebook</u> is devised which could be used for a particular type of image, in that case we need only store the name of that particular codebook file.

In the general case, better results will be obtained with a codebook that is designed for a particular image.

If we include the codebook in the compressed file from the previous example, the compression ratio will not be quite as good. For every 4x4 block we will have 1 byte. This gives us:

$$\left(\frac{256\ \text{pixels}}{4\ \text{pixels/block}}\right)\left(\frac{256\ \text{pixels}}{4\ \text{pixels/block}}\right)=4096\ \text{blocks}$$

At 1 byte for each 4x4 block, this give us 4096 bytes for the codebook addresses. Now we also include the size of the codebook, 256x16:

$$4096+(256)(16)=8192\ \text{bytes for the coded file}$$

The original 8-bit, 256x256 image contained:

$$(256)(256)=65{,}536\ \text{bytes}$$

Thus, we obtain a compression of:

$$\frac{65{,}536}{8192}=8\rightarrow 8:1\ \text{compression}$$

In this case, including the codebook cut the compression in half, from 16:1 to 8:1

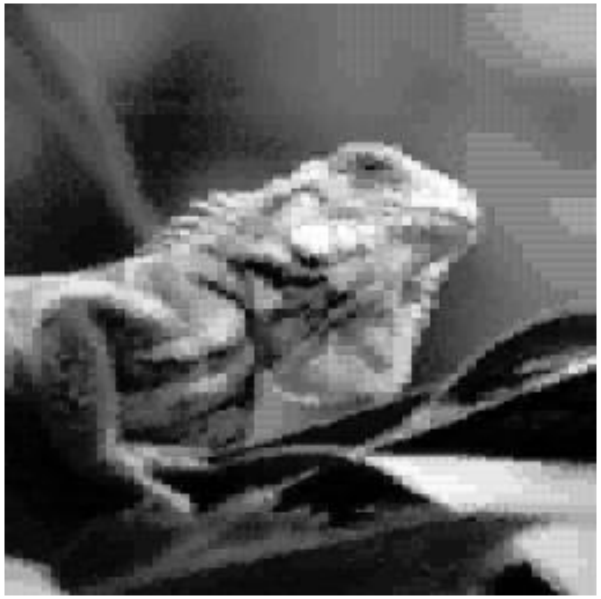
A training algorithm determines which vectors will be stored in the codebook by finding a set of vectors that best represent the blocks in the image.

This set of vectors is determined by optimizing some error criterion, where the error is defined as the sum of the vector distances between the original subimages and the resulting decompressed subimages.

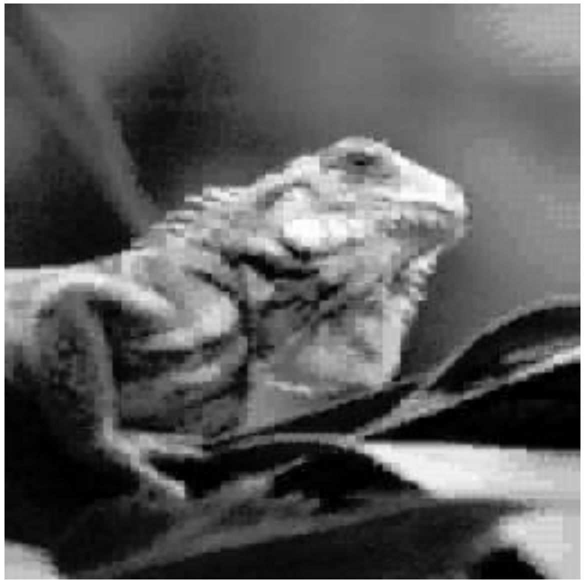
This type of compression is <u>useful for applications where the images are compressed once and decompressed many times</u>, such as images on an Internet site.

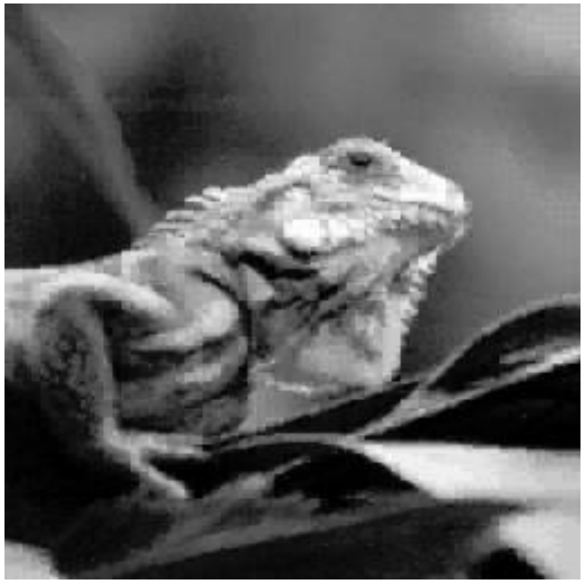
The primary advantage of vector quantization is simple and fast decompression, but with the high cost of complex compression.

a) Original image



b) VQ with 4x4 vectors, and a codebook of 128 entries, compression ratio = 11.49


c) VQ with 4x4 vectors, and a codebook of 256 entries, compression ratio = 7.93


d) VQ with 4x4 vectors, and a codebook of 512 entries, compression ratio = 5.09

Note: As the codebook size is increased, the image quality improves and the compression ratio decreases.

Transform Coding

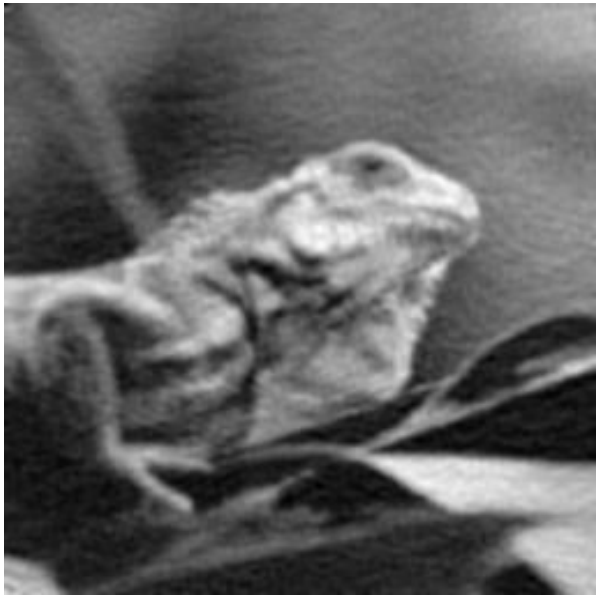
Transform Coding

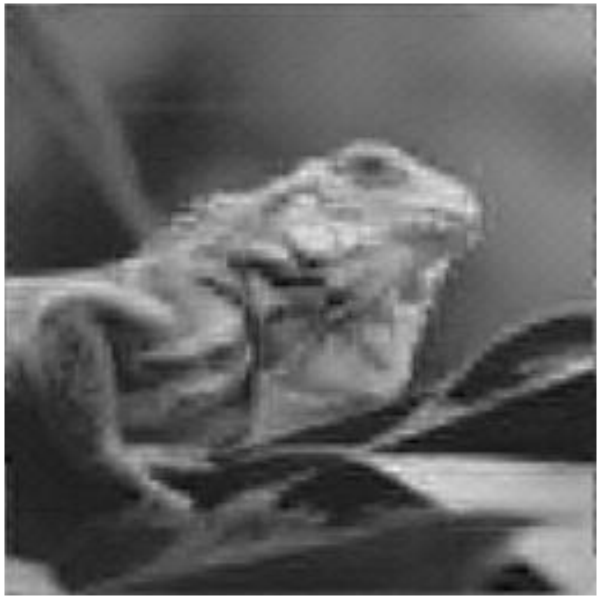
Transform coding, is a form of block coding done in the transform domain.

The image is divided into blocks, or subimages, and the transform is calculated for each block.

Any of the defined transforms can be used, e.g. Fourier or Walsh-Hadamard, but it has been determined that the discrete cosine transform (DCT) is optimal for most images.

The JPEG standard uses DCT and the newer version of this, i.e., JPEG2000 algorithms uses the wavelet transform, which has been found to provide even better compression.


a) VQ with the discrete cosine transform,
compression ratio = 9.21


b) VQ with the wavelet transform,
compression ratio = 9.21

After the transform has been calculated, the transform coefficients are quantized and coded.

This method is effective because the frequency/sequency transformation of images is very efficient at putting most of the information into relatively few of the low frequency coefficients, so many of the high frequency coefficients can be quantized to 0 (eliminated completely).

This type of transform is a special type of mapping that uses spatial-frequency concepts as a basis for the mapping.

The main reason for mapping the original data into another mathematical space is to pack the information (or energy) into as few coefficients as possible.