Transform coding: Encoding, quantization, decoding

The last two lectures I have discussed the general method of transform coding, and given a specific example: the discrete cosine transform (DCT). Let's make sure we understand the general method before we push further into the use of the DCT.

To briefly summarize, here is what the encoder and decoder do. We assume a uniform mid-tread quantizer.

Encoder: (assume Δ is agreed upon by encoder and decoder)

For each block of $m = 8$ samples

1. $\vec{Y} = \mathbf{C}^T \vec{X}$
2. $l_k := \text{round}(\frac{Y_k}{\Delta})$
3. Encode l_k's

Decoder:

1. decode l_k's
2. $Q(Y_k) := l_k \ \Delta$
3. $estimated(\vec{X}) := \mathbf{C}\ Q(\vec{Y})$

To reconstruct its best estimate of the original block $\vec{X}$, the decoder applies the *inverse discrete cosine transform matrix* $\mathbf{C}$. If there were no quantization, the decoder could compute $\mathbf{X}$ exactly since

$$\vec{X} = \mathbf{C}\vec{Y} = \mathbf{C}\mathbf{C}^T\vec{X}$$

since the matrix $\mathbf{C}$ is orthonormal. The decoder only is given the quantized $\vec{Y}$ values, however. So there is no guarentee that $estimated(\vec{X}) = \vec{X}$.

Example

Suppose we have a block of eight values taken from a row in an image. Let's say the values happen to be increasing approximately (but not exactly) linearly:

$$\vec{X} = [25, \ \ 26, \ \ 28, \ \ 29, \ \ 31, \ \ 33, \ \ 34, \ \ 36]^T$$

When we take the DCT of this block, we get:

$$\vec{Y} \ \approx \ [85.60, \ \ -10.30, \ \ 0.27, \ \ -0.78, \ \ 0, \ \ -0.45, \ \ 0.65, \ \ -0.05]^T$$

If we use a uniform midtread quantizer with $\Delta = 1$ (which rounds the values), then we get

$$Q(\vec{Y}) \ = \ [86, \ \ -10, \ \ 0, \ \ -1, \ \ 0, \ \ 0, \ \ 1, \ \ 0]^T$$

This suggests we can compress this block quite well.

The decoder's lossy estimate of $\vec{X}$ is:

$$\mathbf{C}Q(\mathbf{Y}) = estimate(\vec{X}) = [25.3,\ 25.9,\ 28.6,\ 29.5,\ 30.9,\ 33.2,\ 34.0,\ 35.9\ ].$$

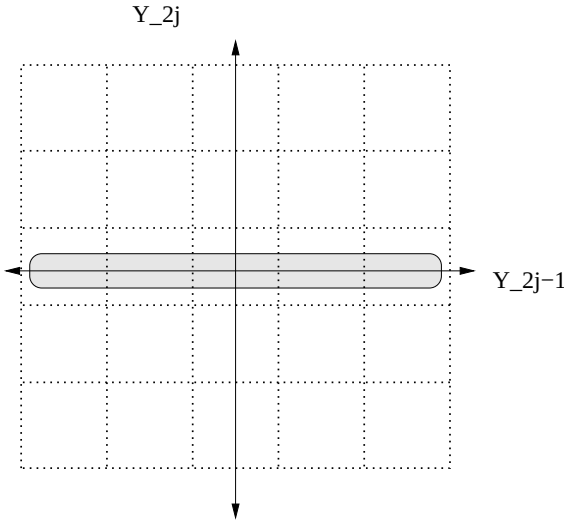
If the decoder wishes to represent $lossy(\vec{X})$ using the same precision as the original $\vec{X}$ (for example, 8 bits per pixel), then a further quantization would be necessary. In the above example, the decoder would round off the values to the nearest integer, lying in the range 0 to 255,

$$25,\ 26,\ 29,\ 30,\ 31,\ 33,\ 34,\ 36$$

which is almost (but not exactly) the same values we started with.

Choice of Δ

In the previous example, we let $\Delta = 1$, and we saw that the original signal was not reconstructed exactly. If we are willing to make larger errors, but use fewer bits to encode the signal then we could use larger values of Δ. The intuition here is that if $\mathcal{E}(Y_i^2)$ is small for some i's, then if we use a quantization Δ such that $\mathcal{E}(Y_i^2) \ll \Delta^2$ then the values of $Q(Y_i)$ will typically be zero, so we can compress very well. This is illustrated in the sketch below. (Note that Δ does not need to be an integer.)



For example, if we used $\Delta = 2$ instead of 1, then we would have

$$Q(\vec{Y}) \ = \ round(\frac{\mathbf{C^T}\vec{X}}{\Delta}) = [43,\quad -5,\quad 0,\quad 0,\quad 0,\quad 0,\quad 0,\quad 0]^T$$

which can be compressed better than the values of $Q(\vec{Y})$ since the values are closer to 0 and indeeed more of the values *are* zero. The estimate would be

$$round(\mathbf{C}(2Q(Y))) = [26,\ 26,\ 28,\ 29,\ 31,\ 33,\ 35,\ 35]$$

so

$$X - round(\mathbf{C}(2Q(Y))) = [1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ -1]$$

2D DCT

The definition of the discrete cosine transform can be generalized to 2D and higher dimensional images. Let's take the 2D case, namely 2D images.

Partition the given image into disjoint blocks of size $m \times m$ pixels. (For JPEG, $m = 8$.) We index the intensities in each block by $X(j_1, j_2)$, where $j_1, j_2 \in 0, 1, \ldots, m-1$. For each block, compute the DCT on each row. This gives 8 values for each row, indexed by the k_2 column variable. Then, for each column k_2, compute the DCT across the m rows. The net effect is the following:

$$Y(k_1, k_2) = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{m-1} \mathbf{C}_{j_1,k_1} \mathbf{C}_{j_2,k_2} X(j_1, j_2)$$

where $\mathbf{C}_{i,k}$ was defined on page 1.

How does this summation relate to the $\vec{Y} = \mathbf{C}^T \vec{X}$ transform we have seen up to now? Each image block $X(j_1, j_2)$ is a 2D array of size $m \times m$. To treat this array X as a vector, i.e. $\vec{Y} = \mathbf{C}^T \vec{X}$, we need to unroll the 2D array into a $m^2 \times 1$ vector. Thus the matrix $\mathbf{C}$ should be a $m^2 \times m^2$ matrix. The rows of $\mathbf{C}^T$ are indexed by $mk_1 + k_2$ and the columns are indexed by $mj_1 + j_2$, where $j_1, j_2, k_1, k_2 \in \{0, 1, \ldots, m-1\}$. The transformed variables $Y(k_1, k_2)$ can also be treated as a $m^2 \times 1$ vector $\vec{Y}$, unrolled from an $m \times m$ block

You may be wondering why we take the DCT transform of rows and columns. Why not just take the transform of $1 \times m$ blocks within an image row? The answer is that if we ignore inter-row correlations, then there will be exist correlations between the transformed values in neighboring rows. Taking the DCT between rows as well as within rows helps get rid of these correlations.

Finally, here is the inverse of the 2D DCT:

$$X(j_1, j_2) = \sum_{k_1=0}^{m-1} \sum_{k_2=0}^{m-1} \mathbf{C}_{j_1,k_1} \mathbf{C}_{j_2,k_2} \; Y(k_1, k_2)$$

where again the $\mathbf{C}_{i,k}$ were defined on page 1. Treating the DCT as a $m^2 \times m^2$ dimensional linear transform, the above inverse transform is (not surprisingly) just the transpose of the DCT matrix.

On the next page is a visual representation of the rows of $\mathbf{C}^T$, or columns of $\mathbf{C}$. Each of the "images" shows an 8×8 matrix,

$$\cos(\frac{\pi}{8}(j_1 + \frac{1}{2})k_1) \quad \cos(\frac{\pi}{8}(j_2 + \frac{1}{2})k_2)$$

for a fixed (k_1, k_2). The different pixels in these 8×8 images correspond to different values of (j_1, j_2). Bright is positive; dark is negative; grey is near zero.

When we take the DCT of an image block, we are taking the inner product of $X(j_1, j_2)$ by each of these images, giving an 8×8 result $Y(k_1, k_2)$.

The top left image is $(k_1, k_2) = (0, 0)$. The bottom left is $(k_1, k_2) =$ (7,0). The top right is $(k_1, k_2) =$ (0,7) etc. These images are literally the *basis images* for an 8×8 block. Within each of these basis images, the 8×8 pixels are indexed by j_1 (row) and j_2 (column).