## JPEG

We assume grey level images only, with 8 bits per pixel. JPEG partitions the image into blocks of size $8 \times 8$. The encoder computes the 2D DCT for each block, yielding 64 "DCT coefficients" $Y(k_1, k_2)$ discussed last lecture.

The $Y(0,0)$ are called the "DC" coefficients, and the remaining 63 coefficients are called the "AC" coefficients. The terms DC and AC come from electrical engineering and stand for "direct current" and "alternating current". The DC coefficient is the $\frac{1}{\sqrt{8*8}} = \frac{1}{8}$ times the sum of the intensities in the block, or 8 times the average. That is,

$$Y(0,0) = \sum_{j_2} \sum_{j_1} \frac{1}{8} I(j_1, j_2) \ .$$

These DC and AC values must be quantized (see below). The value of $\Delta$ is chosen by the user. If the user wants alot of compression and is willing to sacrifice accuracy, then a large $\Delta$ is used. (In fact, JPEG uses slightly different $\Delta$'s for different coefficients, but we omit this detail.)

### DC

The DC coefficients tend to be very similar from one block to the next. The reason is that they can be thought of as (8 times) the mean intensity in a block, and we are assuming that intensities of neighboring pixels are similar. The DC coefficients are encoded using differential coding. Notice that the DC coefficients (prior to quantization) are typically *not* integers, since they represent the sum of 64 intensities values *divided by 8*.

The encoder first computes the quantization level

$$l_1(0,0) = Q(Y_1(0,0)) = round(\frac{Y_1(0,0)}{\Delta})$$

for the first block, and the decoder's estimate of that $Y$ value is

$$\hat{Y}_1(0,0) = \Delta \ \text{round}(\frac{Y_1(0,0)}{\Delta}).$$

Then, given that the decoder has an estimate $\hat{Y}_j(0,0)$ for block $j$, the encoder sends the level for block $j+1$.

Let $Y_{j+1}(0,0)$ be DC coefficient for block $j+1$. (Let's just consider differential coding within a row of blocks.) The level is

$$l_{j+1}(0,0) \ = \ round(\frac{(Y_{j+1}(0,0) - \hat{Y}_j(0,0))}{\Delta})$$

and this level represents the quantized difference:

$$Q(Y_{j+1}(0,0) - \hat{Y}_j(0,0)) = \Delta \ l_{j+1}(0,0)$$

and so the decoder's estimate of the DC value of block $j+1$ would be

$$\hat{Y}_{j+1}(0,0) = \hat{Y}_j(0,0) + Q(Y_{j+1}(0,0) - \hat{Y}_j(0,0)).$$

One question that came up in class is what happens when the differences between $Y_{j+1}(0,0)$ and $Y_j(0,0)$ are smaller than the quantization level $\Delta$. The student asked whether, in this case, large errors could build up. The answer (which I was not quick enough on my feet to give in class) is that this is not a problem. Remember: the encoder is not encoding the quantized differences between $Y_{j+1}(0,0)$ and $Y_j(0,0)$, but rather it is encoding the quantized differences between $Y_{j+1}(0,0)$ and $\hat{Y}_j(0,0)$. See lecture 25 (bottom of page 1) for further info.

Finally, here are a few details about how the levels are encoded. The levels can be positive or negative, and they are more likely to be numbers near zero. The scheme is similar to those we saw in lectures 6 and 7. First, $l_{j+1}(0,0)$ is placed into one of the following groups:

$$\{0\},\ \{-1,1\},\ \{-3,-2,2,3\},\{-7,-6,-5,-4,4,5,6,7\}, etc$$

which have 1,2,4,8, etc elements. In JPEG, a value near zero is more likely than a value that is far from 0, which is why the group sizes are larger for numbers that are bigger in magnitude than zero. A Huffman code is used to specify the group number. (If the probabilities were such that the groups were equally likely, then a fixed length code could be used. In fact, the probabilities of falling into a particular group $g$ is a decreasing function of $g$, even though the group sizes grow like $2^g$.) Then, once the group number has been specified by the Huffman code, the element within the group is encoded (using $\log g$ bits).

## AC

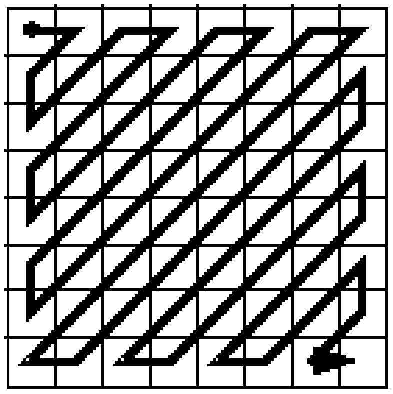
For each block, the encoder computes 63 AC quantization levels:

$$l(k_1,k_2) \equiv \text{ round } (\frac{Y(k_1,k_2)}{\Delta})$$

using a uniform midtread quantizer. Enough levels are allowed for to ensure there are no overload errors (as in the DC case). The AC levels do *not* use differential coding. Each block's AC levels are encoded on their own.

The AC coefficients within each block are encoded as a sequence of 63 values. Here I present a simplified scheme that captures some of the main concepts. The AC coefficients are ordered in a "zigzag" pattern, where $(0,0)$ is at the upper left corner. (The DC is not included, despite what the figure below implies.) For many images, the coefficients $Y(k_1,k_2)$ tend to become smaller in magnitude as $|k_1+k_2|$ becomes larger, and so the bottom-right tend to have smaller values.

For a large value of $\Delta$, i.e. high compression and high loss, many of the levels $l(k_1,k_2)$ are 0 (recall figure on page 1). Thus, we have a sequence of 63 levels to be encoded, many of which are 0. JPEG uses a run length code for these values, namely, runs of 0's. It alternates between encoding a run length, and encoding a non-zero value. Further details of the code are omitted here.



## MPEG

When I introduced video coding in lecture 23, I only mentioned the lossless compression case. In fact, MPEG is a lossy compression scheme! How does that change what I told you earlier?

Recall the key elements of the lossless video scheme:

- frames were I,P or B
- for each block in a P or B frame, the block is encoded in two parts: (1) a motion compensation vector $(v_1, v_2)$ is used to predict the intensities in the block from the intensities of an already-coded frame; (2) the pixel-by-pixel intensity differences are encoded.

The only change for lossy encoding is that the decoder doesn't know the previous I/P frame[1] but rather only knows an approximation of the previous I/P frame. Thus, when the encoder tries to encode the intensity differences between the block and the predicted intensties of the block, these predicted intensities must be available to the decoder. Thus, the predicted intensities should use the decoder's estimates. So the difference to be encoded is:

$$X_{j+1}(k_1 + i_1, k_2 + i_2) - \hat{X}_j(k_1 + i_1 + v_1, k_2 + i_2 + v_2)$$

where you should note the hat over the $X_j$.

How does it encode these differences? Similar to JPEG, the encoder performs a 2D DCT on this difference image (block), quantizes the resulting $Y$ values, and then transmits the quantization levels. Unless in JPEG, it does not treat the DC values separately, however. Instead, it quantizes and encodes all 64 $Y(k_1, k_2)$ values using the same method.

One final note: I didn't mention how the "I" frames are encoded. These are encoded one frame at a time and so a scheme similar to JPEG is used.

[1]The word "previous" refers to bit order here, not display order (see lecture 23 p. 4)."