Image and Video Compression

course: Image and Video Processing
week: 2
Date: 2013-06-07

Why do we need Image compression?

Suppose we have an image of resolution 1000 x 1000 pixels. Now lets calculate the size required to store a video

```
1000 x 1000 x 24 (bits) x 30 (fps) x 60 (seconds) x 120 (2 hours) = 5.184e+12 bits = 603.
```

This is a very very huge size and it wouldn't be feasible to store many of such videos and also transmit them over a network. So, we need compression to lower the size of images / videos.

How can we do image / video compression?

- Coding Redundancy. If there are only a very few color levels used in an image, then is a waste to represent then as 8 bit values. A separate value table can used to represent colors in such situation.
- Spatial Redundancy which means that there are many pixels in the image that represent the same value so it is kind of a waste to repeat all such pixels. Suppose there is a horizontal line containing thousands of pixels of a constant color value, then we can somehow say that repeat the color x number of times. This representation is much smaller and efficient than repeating the pixel again and again.
- Irrelevant Information. If there is irrelevant information in the image, then it can easily be thrown away thus lowering the size of the image.

Image Compression Standards

There are many established standards for image and video compressions. If you compress your image using a certain algorithm and then you share the image, others must have the same algorithm with them to decompress and view the image otherwise your compression would be a waste if no one can see it. That is why we need standards so that everyone can easily compress and decompress images / videos and share them.

Some of the most popular compression standards are -

- Images: JPEG, JPEG-LS (lossless), JPEG-2000, BMP, GIF, PNG, TIFF, etc.
- Videos: H.264, MPEG-1, MPEG-2, MPEG-3, MPEG-4, MPEG-4 AVC, DV, etc.

Image Compression Steps

Input - Raw Image $\rightarrow$ Mapper $\rightarrow$ Quantizer $\rightarrow$ Symbol Coder $\rightarrow$ Output - Compressed Data

Input - Compressed Image $\rightarrow$ Symbol Decoder $\rightarrow$ Inverse Mapper $\rightarrow$ Output - Raw Image

JPEG Standard

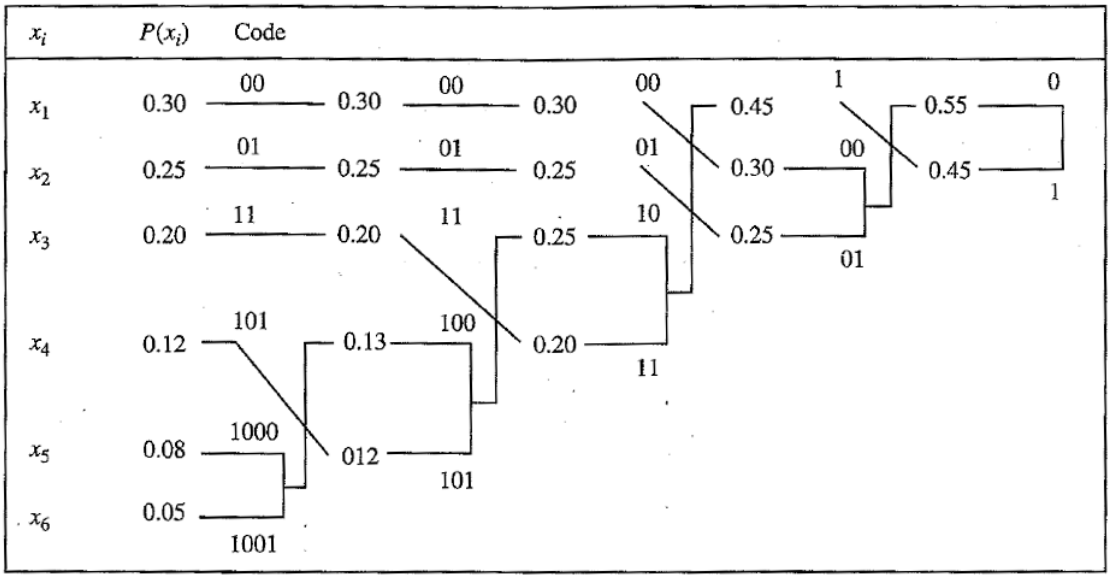
Input Image (`M x N`) $\rightarrow$ Construct `n x n` subimages $\rightarrow$ Forward Transform $\rightarrow$ Quantizer $\rightarrow$ Symbol Encoder $\rightarrow$ Compressed Image

Compressed Image $\rightarrow$ Symbol Decoder $\rightarrow$ Inverse Transform $\rightarrow$ Merger `n x n` subimages $\rightarrow$ Decompressed Image

Huffman Coding

This is the Symbol Encoding step that is described above.

Some of the pixel values in an image are repeated many times and the most repeated values are assigned shorter code values. All these repeated values are actually represented using 8 bit values. For example, suppose a pixel value x appears 500 times in an image and then we assign it a 2 bit value. So, now the image is taking (8 x 500 - 2 x 500 =) 3000 bits less space to store. The higher the probability of occurence of a value, shorter will be its code. This is called Huffman Coding.



Huffman Coding was a bit difficult to explain in words, so look it up in Week 2 video `3 - 2 - 2 - Huffman coding - Duration 2011.mp4`.

To get the amount of compression that can be done with Huffman Coding, Entropy should be calculated-

$$Entropy = H = -\sum_{sumbols} p(s) log_2 p(s)$$

JPEG 8 x 8 blocks

As the first step of compression, the JPEG image (each of the R, G and B parts) is broken in to many blocks of size 8 x 8.

Then, the RBG encoding is converted into YC_bC_r where Y is luminance and the next two are the colors. This conversion is done by multiplying the RGB matrix with a 3 x 3 matrix and that yields the YC_bC_r matrix.

Discrete Cosine Transform

Too mindfuck ~shudders~. Just watch the video `3 - 4 - 4 - The Discrete Cosine Transform (DCT) - Duration 2532`.mp4`.

Lets see why 8 x 8 pixel block is used for DCT. When DCT is done for lower block values such as 2 x 2 and 4 x 4, there are noticeable artifacts (pixelation) in the image. At blocks of 8 x 8, the image looks good enough. But why not just go up and do the DCT of the entire image? The problem with that approach is that DCT computation of multiple smaller blocks is cheaper than one large block. Also, with small blocks, Markovian condition applies which is when Kahunen-Loeve Transform is assumed to be same as DCT. So, 8 x 8 was a good compromise which was studied for long and works good for JPEG.

Quantization

Quantization is not just important because it helps us use less number of bits to represent the colors. But also, when we quantize, we increase the probability of occurrence of certain values which is helpful for Huffman coding.

Not all the values in a 8 x 8 matrix are quantized with the same value. The values towards top left are quantized with lesser numbers to preserve more information about them but as we move away from the top left part, values are quantized with higher values since even if much of the information is lost in these part, it won't affect the quality of the image (for this to happen, the 8 x 8 block must go through Discrete Cosine Transformation). The matrix containing the quantization values are constant and specified under JPEG spec. If the user desires higher compression, only the this quantization matrix has to be scaled up. So, when softwares like Photoshop asks for image quality when saving a JPG, its basically asking how much to scale the quantization matrix.

JPEG-LS and MPEG

JPEG-LS is a compression standard in which there is no loss of the original values. Suppose some of the first few pixels have been encoded, then the next pixel can be predicted based on the previous pixels and then error in this predicted value (predicted - origical) is calculated and stored. Decoder then uses this error to reconstruct the original matrix. Error in prediction happens when there is a drastic change in the color values between the adjacent pixels. This error matrix can then be quantized to achieve even higher compression.

MPEG uses a similar technique. Here, the MPEG encoder tries to predict the next frame based on the previous ones. If a region from the previous frames is found be somewhere near the same position in the next frame, then it will try to encode only the error. This is reason why a video with very similar frames such as the music lyrics videos, person with static background, presentation images, etc. stream smoothly on slow connections (such as 512kbps) whereas videos with most of the element in it changing take a long time to buffer.

Run-length Compression

Consider te image below. It is a binary image containing only two images. It is a perfect candidate for Huffman Coding and can be compressed to very small file. But to get even better result, Run-length Compression can be used. Lets say the color A appears from pixel 0 to pixel 200 in a given column, then instead of repeating its value (or even its shorter version of Huffman Coding) 200 times, what we can do it mention the color just one and then say the number of times the color is repeated. This will help us achieve even higher level of compression.

Note: Missed out too much due to excess amount of maths. :(