Lecture 8 - Compression

Gidon Rosalki

2025-12-07

1 Introduction

The purpose of compression is to reduce the number of bits needed to represent an image. Consider, a still image of $1K \times 1K$, each pixel comprised of 24 bits, and we will need 3MB for each image. Therefore, 1 second of HD video, at 25 frames per second will need 75MB per second, and so an hour would need 270GB. We know that it is possible to compress this, because today an hour of HD video only needs around 1GB of storage space.
There is possibility of spatial correlation, pixels that are close in space are similar, so less information can be stored. Additionally there is temporal correlation in videos, frames that are close to each other in time are often similar.

There are two main types of compression, lossless and lossy. In lossless, there is no information loss, and the image may be reconstructed exactly as the original. This is often applied in medical imaging, archiving, and may be applied to any file, such as text. An image example is png files.
However, lossy compression allows small changes from the original. This may be applied to all photography, and is designed specifically for images and video.

2 Huffman encoding

Huffman is a form of lossless compression. In the first pass, it measures the distribution of the values, and in teh second pass uses a variable length code where rare values get longer codes, and popular values get shorter codes. This reduces the total number of bits required to represent an image, and is effective when different values have different probabilities. It can be used to encode all of English with approximately 1 bit per letter.

So, how do we perform this? It involves a path in a tree. Let us assume that our pixels have colours $a_1, a_2, \ldots, a_n$ with probabilities $p_1, \ldots, p_n$. We then build a tree by iteratively joining two nodes with the lowest probabilities. The colours of the pixels are now the leaves in the tree, and the Huffman code is an encoding of the path from the root of hte tree to each leaf. Colours that appear frequently get a short code, and rarer values get longer codes.

Consider the following table:

Pixel value	1	2	3	4
Probability	0.1	0.05	0.05	0.8
Huffman code	11	101	100	0
Code length	2	3	3	1

Table 1: Huffman table

So here, if we encode 1,000 pixels with this distribution, a fixed length code would need 2 bits to encode 4 values, resulting in a total of 2,000 bits. However, with Huffman, we would only need

$$800 \cdot 1 + 100 \cdot 2 + 100 \cdot 3 = 1300$$

A significant reduction. If all the colours have equal probabilities, then we will produce a uniform length code, and there will be no compression. A question from an exam, how can one compress a binary image using Huffman? One cannot, since one still needs at least two bits. By definition, Huffman codes cannot help binary images.

3 Entropy

3.1 Shannon encoding

Given a probability distribution $p(\cdot)$ over events $0 \leq k \leq n$, how much new information is received when we are told that event k occurred? Common events carry very little information, but rare events, much more. Shannon encoded information as $-\log_2 p(k)$. So a high value of $p(k)$ is very little information, but a low $p(k)$ implies a lot of information. Since each k comes with the probability $p(k)$, the average new information for the distribution is given as

$$\text{Entropy } = H(X) = \sum_{k=0}^{n-1} p(k) \log p(k)$$

Entropy gives the information per pixel for a distribution. The distribution with the highest certainty would be $p(k) = 0$ except for $p(0) = 1$. This has the entropy of 0, so absolute certainty. The distribution with the maximum uncertainty is the uniform distribution, with $p(k) = \frac{1}{n}$, and the entropy of $\log(n)$.
To draw an analogy to physics, systems with lower temperatures have lower entropy, and systems with higher temperatures have higher entropy.

3.2 Predictive encoding

This predicts a pixel value, and then transmits the *difference* from the prediction. For example, a linear predictor from earlier raster pixels:

$$\begin{bmatrix} y_2 & y_3 \\ y_1 & x \end{bmatrix} x = \frac{y_1 + y_3}{2} \tag{1}$$

The original picture may then be recovered from the predicted differences. The histogram of differences has much lower entropy than the original histogram, and so we can get a much better Huffman encoding of it than of the original image.

4 Lossless compression

to perform lossless compression we:

1. create a predictive encoding to reduce entropy
2. Select and perform a favoured lossless encoding:
   (a) Huffman coding
   (b) Lempel Ziv
3. Pack the results in a dedicated format with the checksum, and so on

Let us consider the following two images:

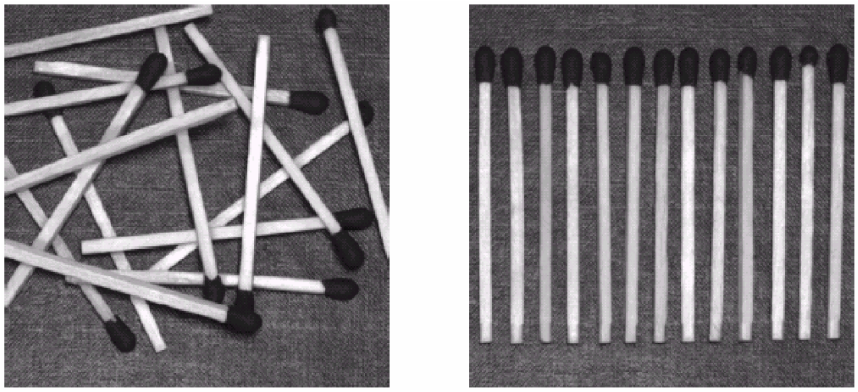

Figure 1:

Despite the fact that the left image has a high entropy, and the right a low, they have a very similar histogram, and so Huffman will give very similar encodings for the two of them. We can do better by using Lempel-Ziv compression (LZW).

4.1 LZW

This is particularly powerful with repetitive patterns. It is a one pass, online algorithm, that builds an on the fly dictionary of strings. The index of entries in the dictionary are transmitted (here the pixel value of 0 to 255 is treated s a character). The same dictionary is built at the receiving side. However, be warned, there are several different variants. Let us consider the following example algorithm:

LZW 1

Input: *input*
Output: *output*

```
1: Initialise a dictionary with all single characters (0..255)
2: P = first input character
3: while Not at end of input strem do
4:     C = next input character
5:     if PC is in dictionary then
6:         P = PC
7:     else
8:         output the code for P
9:         Add PC to the dictionary
10:        P = C
11:    end if
12: end while
13: return Output index for P
```

The important thing to note here is that LZW is very good at repetitive patterns, since they will be encoded once, and referenced many times.

4.2 Run Length Encoding - RLE

This is mainly used in binary images. It encodes pairs, indicating the colour, and the length of that colour, for example:

$$(1, 63), (0, 87), (1, 37), \ldots$$

This works very well on large uniform regions like text. We can even remove the colour bit, since we know that it changes every time there is a new input. We will note that this is not always an effective method for compression, sometimes it may in fact increase the amount of data, depending on the input. We will also note that repeated compression **does not help**, and will often **increase** the amount of data required.

5 Lossy compression

Here, we take the image, transform it, quantise it, encode, and then store / transmit it. To decompress we simply decode, and run the inverse transform. The quantisation is the lossy part, and is the main source for compression but also for errors. Errors are not easy to measure. We can use things like MLE, but it is imperfect. Consider the following simple example:

1. Image Transform: Convolve with (1, -1)
2. Quantization: None
3. Coding: Huffman Coding
4. Lossless coding
5. Improved compression: after convolution with (1, -1) the colour distribution will be concentrated around zero.

5.1 JPEG

For JPEG, we split an image into non overlapping $n \times n$ blocks, and perform a transform on those blocks to give $n \times n$ transform values. We then quantise, and encode these values. JPEG uses 8×8 blocks, with the Discrete Cosine Transform (DCT), uniform quantisation, and Huffman encoding.

For the 64 (8×8) DCT basis function, each 8 by 8 image block is represented as a weighted sum of the basis functions. The 2D DCT transform finds those weights, and low weights (of high frequencies) are zeroed.

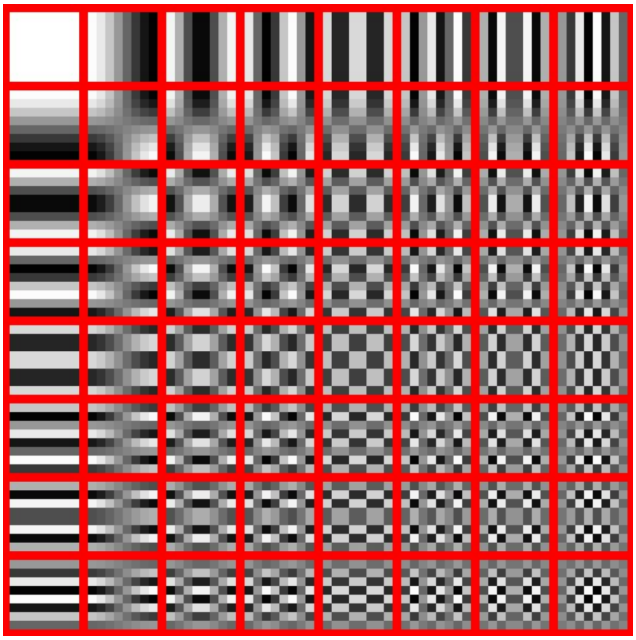

Figure 2: DCT Basis

We use the DCT, in place of the DFT since DCT is better for compression than the DFT. Reflection in place of duplication at the edges reduces the high frequencies, and the DCT has real coefficients, in place of the DFT's complex coefficients.

5.1.1 Colours

For colours in JPEG we transform from RGB to $Y'C_BC_R$. Here, Y' is the intensity, a weighted sum of R, G, and B. C_B, C_R are chromaticity values by colour differences, so

$$C_B = B - (R + G) \tag{2}$$
$$C_R = R - (B + G) \tag{3}$$
$$\tag{4}$$