Transform Coding of Still Images

February 2018

1 Introduction

1.1 Overview

A transform coder consists of three distinct parts: The *transform*, the *quantizer* and the *source coder*. In this laboration you will study all three parts and see how the choice of transform/quantizer/source coder affects the performance of the transform coder.

The laboration runs in a MATLAB environment. In MATLAB images are naturally represented as matrices. During the laboration certain test images will be compressed. The test images can be thought of as *original images* in the sense that they are stored as raw samples with 24 bits/pixel. This usually gives more colours than the human eye can discern on a computer screen.

1.2 Preparations

We assume that you have studied the chapters on transform coding, quantization and entropy coding in the course literature. Read through this document carefully before the laboration. We also suggest that you read the manual (`help <funcname>`) for each function before you use it.

2 Image manipulation in MATLAB

We will be working with colour images. An image can be read into MATLAB using the command `imread`. For future manipulation, we change the pixel values to be floating point values between 0 and 1:

```
>> im1=double(imread('image1.png'))/255;
```

The colour image is stored as a $512 \times 768 \times 3$ matrix, meaning that we have one 512×768 matrix for each of the three RGB colour components (red, green and blue). The image can be viewed using the command `imshow`:

```
>> imshow(im1)
```

We can also view each colour plane separately:

```
>> im1r=im1(:,:,1); im1g=im1(:,:,2); im1b=im1(:,:,3);
>> imshow(im1r), figure, imshow(im1g), figure, imshow(im1b)
```

When coding colour images, we usually use the YCbCr colour space (or another luminance/chrominance colour space), rather than the RGB colour space. To convert the image back and forth between the different colour spaces, use the functions `rgb2ycbcr` and `ycbcr2rgb`:

```
>> im1ycbcr=rgb2ycbcr(im1);
```

See what the luminance and chrominance components look like:

```
>> im1y=im1ycbcr(:,:,1); im1cb=im1ycbcr(:,:,2); im1cr=im1ycbcr(:,:,3);
>> imshow(im1y), figure, imshow(im1cb), figure, imshow(im1cr)
```

The luminance component is basically a greyscale version of the colour image, while the two chrominance components contain information about the colour of the image.

There are six images (named `image1.png` to `image6.png`) with varying content that you can use for your experiments. Take a look at the other images too.

3 Image transformations

The transforms that we will use in this laboration are block-based transforms. The image is divided into small rectangular blocks that are transformed using a separable two-dimensional transform. We will use two kinds of transforms: A *discrete cosine transform (DCT)* and a *discrete Walsh-Hadamard transform (DWHT)*. The MATLAB functions to do the transforms are called `bdct` and `bdwht`. As arguments, the functions take an image and the blocksize to use. Read the manuals for the functions before using them.

Example:

```
>> im1yd = bdct(im1y, [8 8]);
```

This divides the image into blocks of 8×8 pixels and does a DCT on each block.

Since we want to be able to treat each transform component separately, the result of the transform is a matrix where each transformed block is stored as one column (the columns of the block have just been rearranged into a single column).

Inverse transformation is performed with the functions `ibdct` and `ibdwht`.

Example:

```
>> im1yr = ibdct(im1yd, [8 8], [512 768]);
```

Without quantization, `im1y` and `im1yr` should be identical. Due to rounding errors in the computer, there might be a slight (but negligible) difference.

4 Quantization

The second part of a transform image coder is the quantizer. We will only be looking at uniform quantization, using the function `bquant`. This function takes a transformed image and a quantization parameter as arguments. The quantization parameter is the stepsize of the uniform quantizer. To perform inverse quantization (reconstruction), use the function `brec`.

Example:

```
>> Q1=0.1;
>> im1ydq = bquant(im1yd, Q1);
>> im1ydr = brec(im1ydq, Q1);
>> im1yr = ibdct(im1ydr, [8 8], [512 768]);
>> figure, imshow(im1yr);
```

It is also possible to have separate stepsizes for every transform component. The quantization parameter should then be given as a vector or matrix of quantization steps, one for each transform component. It is a hard problem to find the optimal quantization vector for a particular image. Usually it is desirable to have finer quantization for the low frequency components and coarser quantization for the high frequency components. A matrix (for blocksize 8×8) that varies linearly from low to high frequencies can be constructed by

```
>> QL=repmat(1:8, 8, 1); QL=(QL+QL'-9)/8
```

A quantization matrix can then be constructed in the following way.

```
>> k1=0.1; k2=0.3;
>> Q2=k1*(1+k2*QL);
>> im1ydq = bquant(im1yd, Q2);
>> im1ydr = brec(im1ydq, Q2);
>> im1yr = ibdct(im1ydr, [8 8], [512 768]);
>> figure, imshow(im1yr);
```

`k2` affects how big the difference between the smallest and the largest stepsize is and should be chosen between 0 and 1. `k1` adds a constant value to all stepsizes. Setting

`k2` to zero corresponds to having the stepsize `k1` for all transform components.

The distortion (the mean square error) between the original image and a reconstructed image can be calculated as

```
>> dist = mean((im1y(:)-im1yr(:)).^2)
```

Usually, distortion in images is measured by the PSNR (*Peak-to-peak Signal to Noise Ratio*). The PSNR is defined as

$$\text{PSNR} = 10 \cdot \log_{10} \frac{x_{pp}^2}{D} \quad [\text{dB}]$$

where x_{pp} is the peak-to-peak value of the signal, ie the difference between the largest and the smallest value of the signal. For our images, where the scaled pixels can take values between 0 and 1, it is given by

```
>> psnr = 10*log10(1/dist)
```

There is no standard way of defining distortion and PSNR for colour images. The simplest way is to just average the distortions of the three colour components (red, green and blue).

By changing the quantization step (`Q1` or `Q2`), you get reconstructed images with different quality. Try quantizing with different quantization steps and see how the reconstructed image is affected.

5 Source coding

The third part of a transform image coder is the source coder (or entropy coder). In this lab we're going to study two methods: Memoryless Huffman coding, using the function `huffman`, and JPEG-style coding using runlength coding of zeros combined with Huffman coding, using the function `jpgrate`. Neither of these functions perform the actual coding, they just calculate the number of bits needed.

The source coding is done on the quantized transform image. To be able to do Huffman coding, we first need to estimate the distribution by calculating a histogram of the data we want to code. This can be done with the function `ihist`.

The simplest case is if we use a single Huffman code for all transform components.

Example:

```
>> p = ihist(im1ydq(:));
>> bits = huffman(p)
```

This returns the total number of bits required to code the quantized data. Note that the `huffman` function doesn't include the cost to code the Huffman tree, and thus the real number of bits needed would be slightly higher.

Normally, we measure the rate in bits per pixel, which we can get by just dividing the total number of bits with the image size:

```
>> bpp = bits/(512*768)
```

The quantization will of course affect the bit rate: Coarser quantization gives a lower rate while finer quantization gives a higher rate. Change the quantization parameter (`Q1` or `Q2`) and see how the rate is affected.

Since the distribution typically is different for different transform components, a more efficient coding can be performed if we have separate Huffman codes for each component:

```
>> bits=0;
>> for k=1:size(im1ydq, 1)
p = ihist(im1ydq(k, :));
bits = bits + huffman(p);
end
>> bpp = bits/(512*768)
```

The `jpgrate` function returns a vector of the number of bits needed for each block. Just sum all the values to get the total number of bits.

```
>> bits=sum(jpgrate(im1ydq, [8 8]));
>> bpp = bits/(512*768)
```

6 Chrominance subsampling

Usually, the chrominance signals (Cb and Cr) can be subsampled before coding, without giving noticeable effects on the image quality. Subsampling can be done using the function `imresize`.

Example:

```
>> im1cb2 = imresize(im1cb, 0.5);
```

This will subsample the chrominance image with a factor 2 both horizontally and vertically. The subsampled image should then be coded (transformed, quantized and entropy coded). In the decoder, the reconstructed chrominance image is upsampled before transformation back into the RGB colour space

```
>> im1cbr = imresize(im1cb2r, 2);
```

assuming that the reconstructed subsampled image was called `im1cb2r`.

7 Own experiments

Now that you know how to use all three parts (transform, quantizer, entropy coder) it is time for you to experiment freely.

A simple MATLAB function to get you started can be found in `transcoder.m`. Copy it to your directory and edit it to suit your needs.

Try coding both greyscale images (use either the luminance component of an YCbCr image, or the green component of an RGB image) and colour images.

NOTE: When comparing two different coder configurations, you should compare them at the same rate and check to see which one gives the lowest distortion. Ideally, the comparison should be done for a whole range of rates. Quality comparisons should be done both objectively (by comparing PSNR figures) and subjectively (by looking at the decoded images).

You should find answers to the following questions:

- What source coding method gives the smallest rates?
- What choice of transform (DCT or DWHT) gives the best results?
- What choice of block size gives the best results?
- What quantization method gives the best results, using the same stepsize for all transform components or using different stepsizes for different transform components?
- How does chrominance subsampling affect the results?
- What is the lowest rate (in bits per pixel) that gives coded images that are indistinguishable from the original image at normal viewing distance?
- What is the lowest rate that gives an acceptable image quality?

8 Block schedule