Project Work on Multimedia Services and Applications

by Mathias Rønning

Introduction

The aim of this project was to create a functional baseline JPEG encoder using Python. This was achieved using several libraries, namely OpenCV for image loading, Numpy for handling of image arrays and SciPy for discrete cosine transform (DCT) functions. The encoder is a baseline JPEG encoder using 4:2:0 chroma subsampling and huffman encoding. The development process was done following the slides given during the course in addition to the JPEG standard for details on the specific parts.

Running the code

To run the code, first the required libraries must be installed. This is most easily done using the command `pip install -r requirements.txt` from the base folder. Following this, the encoder can be run using the command `python jpeg_encoder.py`. The default image used is kodim23.png, however this can be changed by altering the `encode()` call at the end of the file. The script outputs the encoded JPEG image with the filename `test.jpg`.

Process

Image loading and chroma subsampling

The first step of the process was loading the image, where PNG images (or any other lossless format) are loaded and converted to the YCbCr colorspace using OpenCV functions. Following this, the Cb and Cr channels are subsampled using the 4:2:0 mode taking the average intensity of 2x2 blocks. Each channel is then divided into 8x8 blocks for further processing.

DCT and quantization

The function `run_dct` iterates over each block of a channel and applies the 2D DCT by using SciPy's `fftpack.dct` twice, once for each of the axes. The transformed blocks can then be quantized using the `q_factor` parameter, by dividing each cell by the quantization matrix entry times the scaling factor. The quantization matrices used are taken from the JPEG standard Annex K.

Reordering of blocks

Following the DCT transformation, the blocks are reordered in the zig-zag order specified by the JPEG standard. The luminance blocks are further reordered by ordering each 2x2 block from top left to bottom right. This is because the data is later encoded in interleaved mode, where there are 4 consecutive luminance blocks followed by 2 chrominance.

Lossless coding

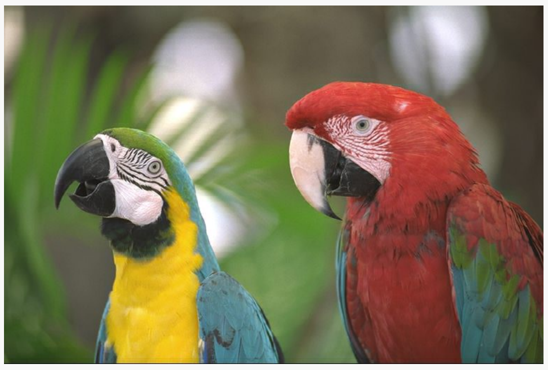
Following the reordering, the DC components of each channel are extracted for separate coding from the AC counterparts. The DC components are DPCM coded in the `dpcm` function, where the differences are calculated with the first DC component as reference. Each of the differences is converted to a size and amplitude following the standard. For the AC components, run length coding is employed with the format `((RUNLENGTH, SIZE), AMPLITUDE)`. Following the standard, runs of 16 zeros are encoded as (15, 0).

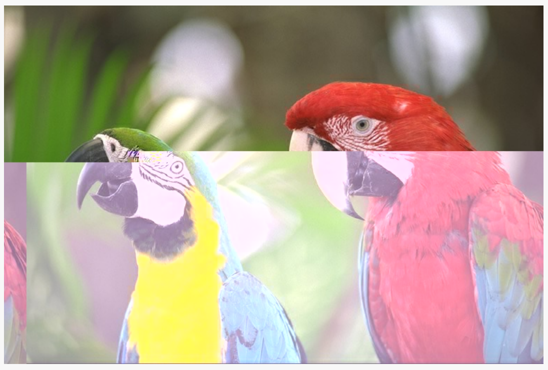
File writing and Huffman coding

The last part of the encoder was encoding the data using Huffman coding and writing the actual JPEG file. The huffman tables used are the ones provided from Annex K of the JPEG standard, and are located in the file `huffman.py`. The various headers of the different sections are directly provided in hexadecimal form since they do not change, while the data section are encoded using the Huffman tables for DC and AC luminance and chrominance. As previously mentioned, the data is encoded in interleaved mode, having 4 consecutive luminance blocks followed by 2 chrominance ones.

Results

The resulting encoder is working and correctly encodes the images to the JPEG format, however only for Q-factors up to a certain value. Above this value (which changes depending on the image), a large artifact appears as shown in the following images (image to left has Q-factor 80 while right has 90).





The reson for this is unknown, however it is assumed to be in the lossless encoding part. It may be the difference coding that is wrong, causing a few blocks to be wrongly encoded, and the following ones to have a wrong reference for the difference. This is assumed since there is a "jump" in the image corresponding to around 2 blocks, and the following blocks seem to have a wrong brightness level. Also since it only happens with high Q-factor values, it may have something to do with high difference values being encoded wrongly.

References

- JPEG standard
- Kodak image dataset