DFT/DCT Error Calculation and Comparison in C++ Using OpenCV

R. Joseph Rupert

December 16th, 2021

Abstract:

Image compression systems often use discrete cosine transforms and discrete Fourier transforms as a part of block transform coding, where the image being compressed is divided into a number of nxn subimages, each of which is transformed and quantized before the results are concatenated back into one cohesive image. The goal of this project was to build tools with which we could quantify information loss, then use those to:

1) Compare the DFT to the DCT to determine which method of lossy compression resulted in more information loss.
2) Determine how many coefficients the DCT compressor can discard before the decompressed image quality becomes unacceptable.

To achieve these goals, the author used the OpenCV image processing libraries, implementing them in C++ along with standard C++ general-use libraries. The results showed that while the DFT compressor was superior to the DCT compressor, the DCT could retain only four coefficients and still maintain an acceptable level of quality.

Work Accomplished:

The first steps of this project were to implement C++ programs to calculate the root-mean-squared error (RMS error) and the mean-square signal-to-noise ratio (SN ratio) of a compressed-decompressed image.

$$e_{rms} = \sqrt{\frac{1}{MN}\sum \left(f(x,\ y)\ -\ f'(x,\ y)\right)^2}$$ (Equation 1) RMS error

$$sn_{ms} = \frac{\sum (f'(x,y))^2}{\sum (f(x,y) - f'(x,y))^2}$$ (Equation 2) SN ratio

To obtain the images for comparison, the OpenCV imread function was used to read an image into an OpenCV matrix (cv::Mat) object. From there, the numerical value of each pixel could be accessed and used in the equations above. The implementation for this was relatively simple (can be found in 'headers/error_calc_funcs.cpp' in the source code directory provided).

However, a problem was immediately discovered when these were tested with identical images and both returned NaN. After investigation, it was ascertained that the test image contained a miniscule, scattered quantity of missing pixels in the image that the imread function was therefore unable to assign numerical values to. To rectify this, the author used the OpenCV cv::patchNaNs function to write a small program that would replace all missing pixels in the image with a specified value. After running this on the sample image, both error calculation programs returned the expected results.

The next steps were to implement compressors, using both discrete cosine transforms and discrete Fourier transforms, that could be tested. The compressor structure was fairly simple:

1) Read the image into a Mat object, and split it (using the cv::split function) into a set of 1-channel Mat objects representing each channel of the image.
2) Divide each object into N 8x8 subimages.
3) Run the OpenCV transform (cv::dct or cv::dft) on each subimage.
4) Quantize it using an 8x8 quantization table (pulled from the JPEG image standard): {16, 11, 10, 16, 24, 40, 51, 61},
   {12, 12, 14, 19, 26, 58, 60, 55},
   {14, 13, 16, 24, 40, 57, 69, 56},
   {14, 17, 22, 29, 51, 87, 80, 62},
   {18, 22, 37, 56, 68, 109, 103, 77},
   {24, 35, 55, 64, 81, 104, 113, 92},
   {49, 64, 78, 87, 103, 121, 120, 101},
   {72, 92, 95, 98, 112, 100, 103, 99}
5) Discard all but a specified number (N) of coefficients from the resulting matrix. (The subroutine to do this iterated through the matrix, populating a C++ vector object with the N largest coefficients in order, then compared each matrix element to the smallest of the coefficients collected and zeroed it if it was smaller.)

6) Place it in an output matrix object representing one channel of the output image.
7) Merge each channel matrix (using cv::merge) back into a single matrix and write the result out to a file.

Following this, the corresponding decompressors had to be written. The process was similar to the process detailed above for the compressor, but with certain differences:

- The OpenCV inverse transform function (cv::idct or cv::idft) was used instead of the forward transform function.
- No quantization or coefficient removal was performed.

The implementations described above can be found in the provided source code directory in the file 'headers/compressor_decompressor.cpp', and were used for simple compressor and decompressor programs. These were tested by compressing and decompressing a sample image, which achieved the expected result.

From here, the final goal of the project could be achieved: testing the compressors to compare the information loss from the DCT and DFT compressors, as well as the information loss from the DCT compressor with descending numbers of retained coefficients. The results are as follows (selected images included, all images submitted separately):

DFT (8-Largest) RMS:	3.6985e+15
“ “ “ “ SN:	95.1543
DCT (8-Largest) RMS:	4.61326e+15
“ “ “ “ SN:	57.0374
DCT (7-Largest) RMS:	4.70007e+15
“ “ “ “ SN:	54.9518
DCT (6-Largest) RMS:	4.75806e+15
“ “ “ “ SN:	53.61
DCT (5-Largest) RMS:	4.89729e+15
“ “ “ “ SN:	50.5761
DCT (4-Largest) RMS:	5.046e+15
“ “ “ “ SN:	47.5864
DCT (3-Largest) RMS:	5.20273e+15
“ “ “ “ SN:	44.6937
DCT (2-Largest) RMS:	5.92823e+15
“ “ “ “ SN:	34.2133
DCT (1-Largest) RMS:	8.95486e+15
“ “ “ “ SN:	14.2203

(^Original image)

(^Compressed with DCT 8-largest)

(^Compressed with DCT 2-largest)

The results are indicative of two conclusions: The DFT compressor is superior to the DCT variant, and the DCT variant can maintain surprisingly good image quality with a low number of retained coefficients. A SN ratio of 45 was chosen as the threshold for acceptability.

How to Build and Run Code:

On Linux (possibly usable on other OSes, but untested), run 'make' in the Final Project source code directory. This builds four executables in this directory:

compressor

decompressor

comparator

nanpatcher

'compressor' takes five arguments, in this order: the input filename (or relative path if it's in another directory, the data/input directory as submitted contains the example image used in the tests), the output filename (again, relative path), the type of transform (dft or dct), the block size (currently, only 8 is supported), and the number of coefficients to retain (8 or fewer currently supported).

'decompressor' takes four arguments, in this order: the input filename and output filename (same style as 'compressor'), the type of transform (dft or dct), and the block size (again, only 8 is supported).

‘comparator’ takes three arguments, in this order: the input filename and output filename (same style as ‘compressor’ and ‘decompressor’), and the error measurement system (rms or sn). It then prints the error value requested. ‘nanpatcher’ takes only one argument, the relative path of the file to be patched.

To perform the same tests as the author, first run nanpatcher on the image to be used, then run compressor on it, then decompressor (be sure to specify the same transform technique and block size, and use the output path from compressor as the input), then comparator with the desired error measurement system flag.

To clean executables from the directory (if necessary), run ‘make clean_all’.

Contributions:

All work done in this project was completed by the sole author, R. Joseph Rupert. He would like to thank the OpenCV development team for their excellent documentation, the authors of the course textbook for the aforementioned course textbook, and Prof. Hong Man for the lecture slides. All three proved to be invaluable sources.