CS517: Digital Image Processing & Analysis

Final Lab Test

- **JPEC Encoding / Compression (For grayscale images ONLY!) (`MyJPEGEncoder`)**:
  - **Encoder**: The basic structure of the encoder is given in the following diagram. The encoder is a traditional 8 × 8 block size, DCT (discrete cosine transform) based encoder.

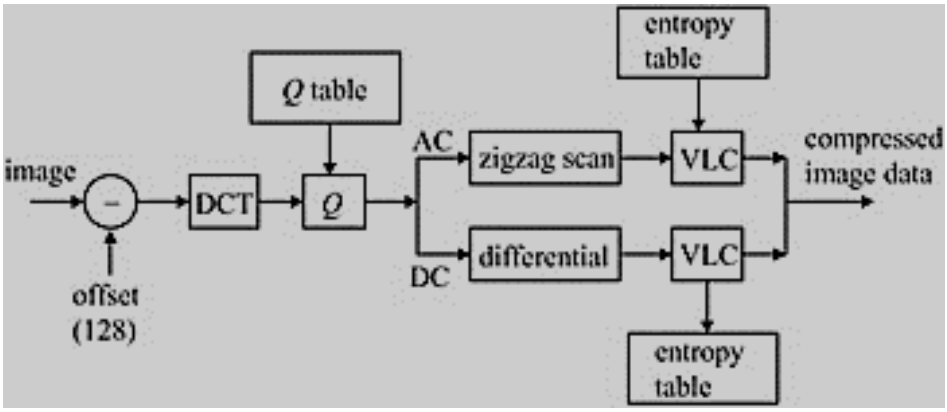


Figure 1: JPEG Encoder Block Diagram

  - **[5 Points]** Usually the processed image is larger than 8 × 8 pixels. Therefore, the processing order for the blocks is specified in the figure below. Your encoder **does not** have to be able to process images for which the size is not divisible by 8, i.e., border padding is not required. (`block` module)

1.	2.	3.	4.	5.	6.
7.	8.	9.	10.	11.	12.
13.	14.	15.	16.	17.	18.
19.	20.	21.	22.	23.	24.

Figure 2: Processing order for a sample 48 x 32 image with the 8 × 8 sub-blocks

  - **[2.5 Points]** **Offset**: The encoder can assume that the pixel values in the source image are within the range [0, 255] (8 bits). This also determines the offset, which in the 8-bit case has the value 128 (= 2^{8-1}). As the first stage of the encoder, this value is subtracted from all pixel values.

  - **[5 Points]** **DCT:** This block performs the 2D discrete cosine transform (DCT) for each 8 × 8 block of the input image. Your task is to call the corresponding function for each 8×8 block in the desired processing order. You can use the `dct` function provided with `openCV` package to accomplish this step.

  - **[5 Points]** **Quantization**: This block performs quantization for each 8 × 8 block of DCT coefficients. The quantization must be implemented using the formula:

$$F^q(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

where $F^q(u, v)$ is a quantized DCT coefficient with the horizontal and the vertical spatial frequencies of u and

v, F(u, v) is the DCT coefficient value prior to quantization and Q(u, v) is an element of the quantization matrix. The operator round(·) means rounding to the nearest integer. The quantization matrix is provided in the `quantmatrix.py` file for you to use.

- **[5 Points] Zig-zag Scanning:** The 63 AC coefficients inside each 8 × 8 block (all the coefficients in the figure below except the black dotted one in the top left corner) are scanned in the zig-zag order shown. In general, it allows for efficient run-length coding. In effect, this block performs a reordering operation on the AC coefficients. (`zigzag` module)

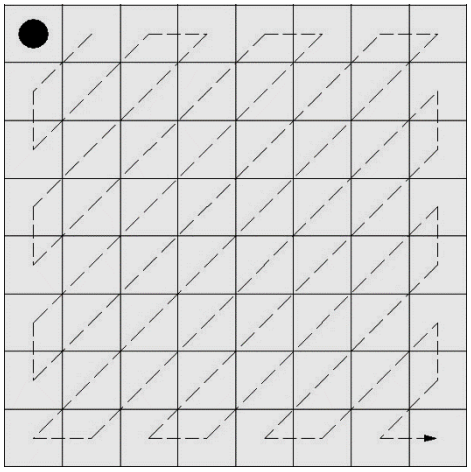


Figure 3: Zig-Zag Scanning Order for AC Coefficients

- **Differential Scanning:** The DC coefficients (marked with the black dot in the figure above) are scanned using differential coding. Because of the high correlation of DC values of adjacent blocks, this allows for efficient entropy coding. The method is illustrated in the figure below. Since for the first DC coefficient there is no preceding coefficient, the value zero is used for the preceding coefficient value (DC_{i-1}) in this case:

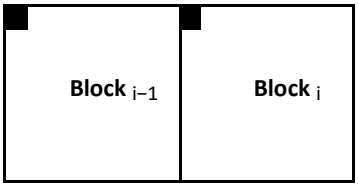


Figure 4: Differential Coding for DC Coefficients

$$DIFF = DC_i - DC_{i-1}$$

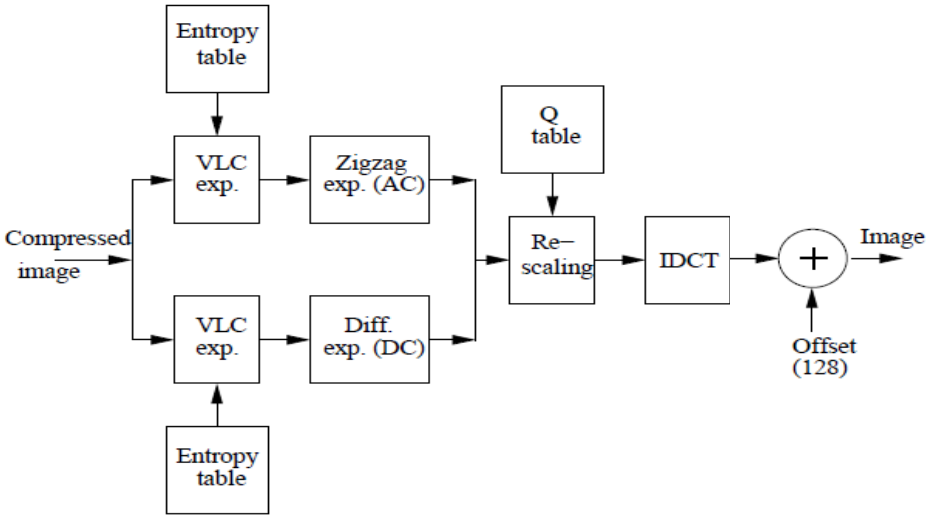
- **[5 Points] Encoding of DC Differential Values:** The DC differential (DIFF) values should be written to the output text file as string of numbers separated by comma or space, in order. (`jDCenc` module)

- **[10 Points] Encoding of AC Coefficients:** The 63 zig-zag ordered AC coefficients of each 8 × 8 block are run-length coded. Each code is a pair (RUN, COEF). Here, COEF is the amplitude of a non-zero coefficient in the zig-zag order, and RUN is the number of zeros preceding this nonzero coefficient. For example, a (RUN=36, COEF=7) pair implies 36 zeros following by non-zero coefficient 7. The pair (RUN=0, COEF=0) has a special meaning: it is the symbol for END-OF-BLOCK (EOB). The EOB symbol is written, when there are no more non-zero coefficients in the current block and there are zero valued coefficients that have not yet been coded in the current block.

  The 63 zig-zag ordered AC coefficients should be written to the output text file as string of RUN, COEFF pairs separated by comma or space, in order for each block. (`jACenc` module)

- **Compression** – As the output text file (`image.rle`) has lesser bytes than original image, it should result in compression.

- **Decoder (`MyJPEGDecoder.m`)**: Our decoder uses naturally the same 8 × 8 block size and the same processing order for the blocks as our encoder (see Figure 2). The decoder is illustrated in Figure 6. It is seen that each block of the encoder (see Figure 1) has a corresponding block in the decoder.



- **[5 Points] Decoding of DC Differential Values:** Read the DC differential (DIFF) values from the compressed image (`image.rle`) and re-calculate the original DC coefficients of all the blocks. (`jDCdec` module)

- **[10 Points] Decoding of AC Coefficients:** Read the Run-Length-Encoded AC coefficients from the compressed image (`image.rle`) and decode the original 63 zig-zag ordered AC coefficients of each 8 × 8 block (`jACdec` module)

- **[5 Points] Zig-zag Expansion:** The order in which the extracted AC coefficients should be placed in the current block is shown in Figure 3. (`izigzag` module)

- **Differential Expansion**: The reconstruction of the original DC coefficients from DC differential values is straightforward. The procedure is illustrated in Figure 4.

- **[5 Points] Rescaling (obligatory):** The purpose of the rescaling block is to restore the original scale of each 8 × 8 block of quantized DCT coefficients. Rescaling must be implemented using the formula:

$$F^{Q}(u,v) = F^{q}(u,v) \cdot Q(u,v),$$

Figure 5: Required Decoder

Where $F^{q}(u, v)$ is a quantized DCT coefficient with the horizontal and the vertical spatial frequencies of u and v and Q(u, v) is an element of the quantization matrix.

Note that because of the rounding operator used in encoding process, the original DCT coefficients are not restored. In fact, some of the quantized coefficients are equal to zero, and naturally remain zero also after the rescaling. This is actually what makes our compression codec lossy.

- **[5 Points] IDCT:** This block performs the inverse 2D discrete cosine transform (IDCT) for each 8 × 8 block of the input image. The transform is done in blocks of 8 × 8 pixels. The processing order for the blocks is the same as for DCT in the encoder (see Figure 2). You can use the `idct` function provided with `openCV` package to accomplish this step. Your task is to call this function for each 8×8 block in the desired processing order.

- **[2.5 Points] Offset:** In the last block of the decoder, the value 128 is added to all pixel values. Finally, all values that are smaller than zero are replaced by zero and all values that are greater than 255 are replaced by 255.

- **[5 Points] Reconstruction:** Reconstruct the image by rearranging and concatenating the individual decompressed blocks.

- **Testing:** For each of the sample images provided (`Lena512.gif, Img1.tiff, Img2.tiff`), try the encoding and decoding process and calculate the compression ratio and error (use `from sklearn.metrics import mean_squared_error).` **Repeat** the above testing process for different scaled versions of the quantization matrix (2, 4, 8, and 16).
    a. Store encoded and decoded files as `ImageFileName.RLE` and `ImageFilenameDEC.img`

- **Display:** For each sample image, display the original image followed by decoded versions of the sample image with different scaling factors (1, 2, 4, 8, 16) in Figure 1 using subplot. In Figure 2, plot the compression ratio vs. Scaling factors for the sample image and in Figure 3, plot the errors against scaling factors.

- **[5 Points]** for each figure and for **[10 Points]** clarity of code / comments.

- **Submitting your work:**
    a. Upload your exercises with your ID filename **2008CS1001–FinalLab.ipnyb** file on google classroom

        i. Your file should have the following modules:

```
MyJPEGEncoder, MyJPEGDecoder,
zigzag, izigzag,
jDCenc, jDCdec,
jACenc, jACdec, block, etc.
```

        ii. **README** (Should provide all the necessary details to the TA for ease of grading – what works, what doesn't, any assumptions made, etc.)
        iii. *Any other utility functions you might end up creating*

    b. ***Negative marks for any problems/errors in running your programs***

    c. Submit/Upload it to Google Classroom