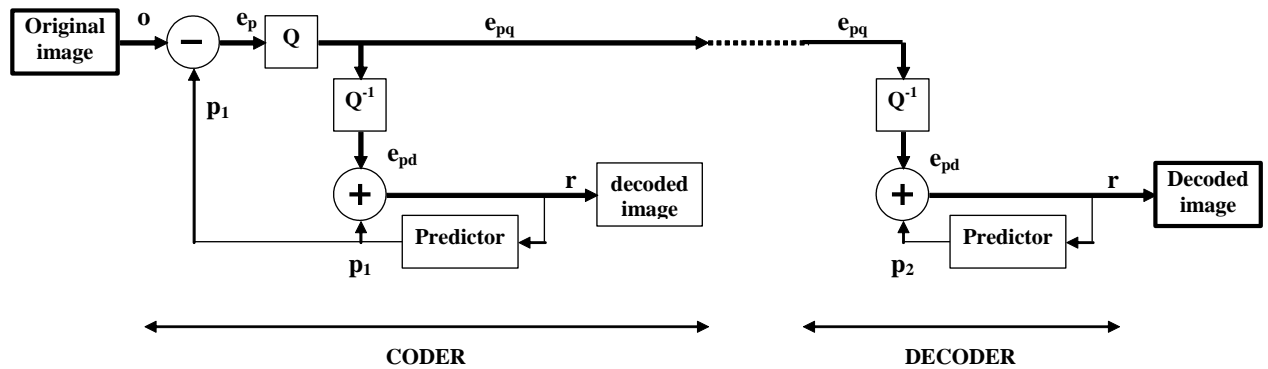


# Near-Lossless predictive coder

## 1. Basic architecture



The near-lossless predictive architecture

## 2. Predictors to be used

- 0: 128
- 1: A
- 2: B
- 3: C
- 4:  $A + B - C$
- 5:  $(A + B - C) / 2$
- 6:  $B + (A - C) / 2$
- 7:  $(A + B) / 2$
- 8: jpegLS

	C	B	
	A	?	

**JpegLS predictor definition:**

$\min(A, B)$  if  $C \geq \max(A, B)$   
 $\max(A, B)$  if  $C \leq \min(A, B)$   
 $A+B-C$  otherwise

**Notices:** for all predictors (except for the 128 one)

- for the first pixel we predict the value 128
- for the first line we predict the value A
- for the first column we predict the value B

Prediction is limited to the  $[0-255]$  interval.

## 3. Quantization - dequantization

The **quantization** rule:

$$e_{pq} = \left\lfloor \frac{e_p + k}{2 \times k + 1} \right\rfloor$$

The **dequantization** rule:

$$e_{pd} = e_{pq} \times (2 \times k + 1)$$

Here the integer value  $k$  (the near-lossless parameter, the accepted error) takes values in the  $[0 - 10]$  interval.

Some examples can be found in the following table:

Original value $e_p$		Quantized value $e_{pq}$		Dequantized value $e_{pd}$	Error $e_p - e_{pd}$
.....		.....		.....	
7	=>	1	=>	5	2
6		1		5	1
5		1		5	0
4		1		5	-1
3		1		5	-2
2	=>	0	=>	0	2
1		0		0	1
0		0		0	0
-1		0		0	-1
-2		0		0	-2
-3	=>	-1	=>	-5	2
-4		-1		-5	1
-5		-1		-5	0
-6		-1		-5	-1
-7		-1		-5	-2
.....		.....		.....	

Table 2 Quantization example (for k=2)

#### 4. Save modes

The quantized prediction errors are saved in the compressed file by three methods:

**F** - Using a **fix** number of bits (9 or 16 or 32)

**T** - Using the following **JPEG table** (taken from the JPEG standard where used for coding the DC coefficients)

**A** - Using **arithmetic coding** (values are shifted from [-255...+255] into [0...+510] by adding 255)

Coding using the Jpeg table.

	Unary code of the line	Index on the line											
		0	1	2	3	4	5	6	7	...	...	...	...

0	0	0											
1	10	-1	1										
2	110	-3	-2	2	3								
3	1110	-7	-6	-5	-4	4	5	6	7				

4	11110	-15	-14	-13	...	-9	-8	8	9	...	13	14	15
5	111110	-31	-30	-29	...	-17	-16	16	17	...	29	30	31
6	1111110	-63	-62	-61	...	-33	-32	32	33	...	61	62	63
7	11111110	-127	-126	-125	...	-65	-64	64	65	...	125	126	127
8	111111110	-255	-254	-253	...	-129	-128	128	129	...	253	254	255

Table 1 JPEG coding table

For each value of the quantized prediction error we save the **unary code of corresponding line** from the table followed by the **index of the value in the corresponding line**.

#### 5. Working with BMP images

For working with BMP images you can use the following (very simple) approach: read from the grayscale test images provided for this lab the header (1078 bytes) and save it. The next 256\*256 bytes

represent row-based pixel values for the image (you can load then directly into the corresponding matrix). The original bmp header can be written in our compressed files to be available at the decoder and used for reconstructing the decoded bmp image.

The idea proposed above can be used in order to avoid specific bmp format topics. If you want you can access bmp files by specific functions, not by this simplified version (which is working only in the specific bmp format of the provided bmp images).

## 6. Compressed file format

The compressed file must contain (the exact order and format is at your choice):

- the original header – only if you use the simplified bmp access as presented above
- the predictor used
- the k parameter value used
- the save mode used
- the 256\*256 quantized error values.

## 7. Interface

You can have a single program that includes both the coder and the decoder (but each has to work also independently).

On your interface you have to have the following components (at least the following functionalities, the exact way you implement them is at your choice):

// mainly for coding

**Original image** (256\*256 grayscale)

**Load button** - to load an original image (\*.bmp)

**Encode button** – to make the encoding process

**Save button** – to save the compressed file (the program should propose a name of the compressed file like in the following example: \*.bmp.p3k2F.prd where 3 means the 3-rd predictor, 2 means k=2, F means Fixed save mode)

**Prediction selector** - used to select the 0 to 8 predictors as described above

**Accepted error selector** - the k value, as described above, in the [0-10] range

**Save mode selector** - F / T / A as described above

// mainly for error inspection

**Error image** (256\*256 grayscale)

**Source selector** - for the error image so we can choose as source:

Prediction error

Quantized prediction error

**Scale value input** – in order to enter a floating point scale value. The values in the error image is computed as  $\text{error} * \text{scale} + 128$  and limited to the [0-255] range.

**Refresh button** - to redraw the error image

// mainly for decoding

**Decoded image** (256\*256 grayscale)

**Load button** - to load a \*.prd file

**Decode button** - to make the decoding process

**Save button** – to save the decoded image (named \*.prd.bmp)

// mainly for histogram inspection

**Histogram image** – a 511 wide (and fixed height) image that represents a histogram (the frequencies of values from -255 to +255)

**Histogram input selector** - to select the image that is the source for the histogram

- original image
- prediction error
- quantized prediction error
- decoded image

**Histogram scale** – a floating point value that multiplies the frequencies in the histogram in order to magnify or reduce the height of the histogram. DO NOT USE AUTOMATIC SCALING OF THE HISTOGRAM.

**Histogram refresh** button

// mainly for error validation

**Compute error** button - to compute the minimum and maximum error between the original and the decoded image

**Maximum error** label – the computed value

**Minimum error** label – the computed value

## 8. Other advices

Use a function/method/etc. to put the code for the prediction made for 1 pixel.