CMPT 365 Project2 Report

Junchen Li

301385486

2021/4/21

Introduction:

We need to write a simply compression-decompression application to read tif. format image files. Do the lossless compression and lossy compression to each group of input sample respectively. Compared and discussed differences between original images and decompression image. (Quality and compression ratio).

Experiment environment:

I decided to use Java in IntelliJ. Most libraries which I used in this report is about JavaFx. JavaFx library is used for setting the UI and this library contains a lot of UI elements. Like setting ActionEvent and EventHandler for menu button, make a file chooser, scene, border pane, stage.

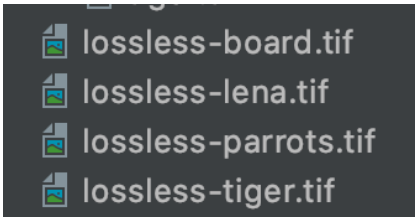
There is a library for the file "java.io.File" When user choose a tiff file and we set a variable which its data type is File. And we can decode this File variable.

"java.io.IOExpection" When we select a tif. File and if it is not existing, we need to throw an exception, leave an error message and terminate the program.

The version of whole library jar that I used is "Javafx-sdk-11.0.2", there are two more external libraries is "jai-codec-1.1.3.jar" and "jai-core-1.1.3.jar". They are working for loading images and facilitates manipulation of pixels.

Lossless compression:

For the binary file, every time we compress an image, it will automatically generate an intermediate result. We could directly use the sublime text to open them and save them as a binary file. The follow pictures just an example of "lossless-board"



The way that calculate the compression ratio is we get the original and compressed image. Then we use the number of bytes of the compressed file divided by the number of bytes of the pre-compressed file. Then we can standardize the result format range and get the compression ratio. The ratio will show in the resulting part.

```
double result = (targetFile.length() * 100.0 / selectFile.length());
```

We let board.tif and lena.tif are our examples:

Ratio(board) = $\frac{292838*100}{293262} = 99.86$ $100 - 99.86 = 0.14 = 14\%$

Ratio(board) = $\frac{786440*100}{786572} = 99.98$ $100 - 99.98 = 0.02 = 2\%$

Key technique of compress:

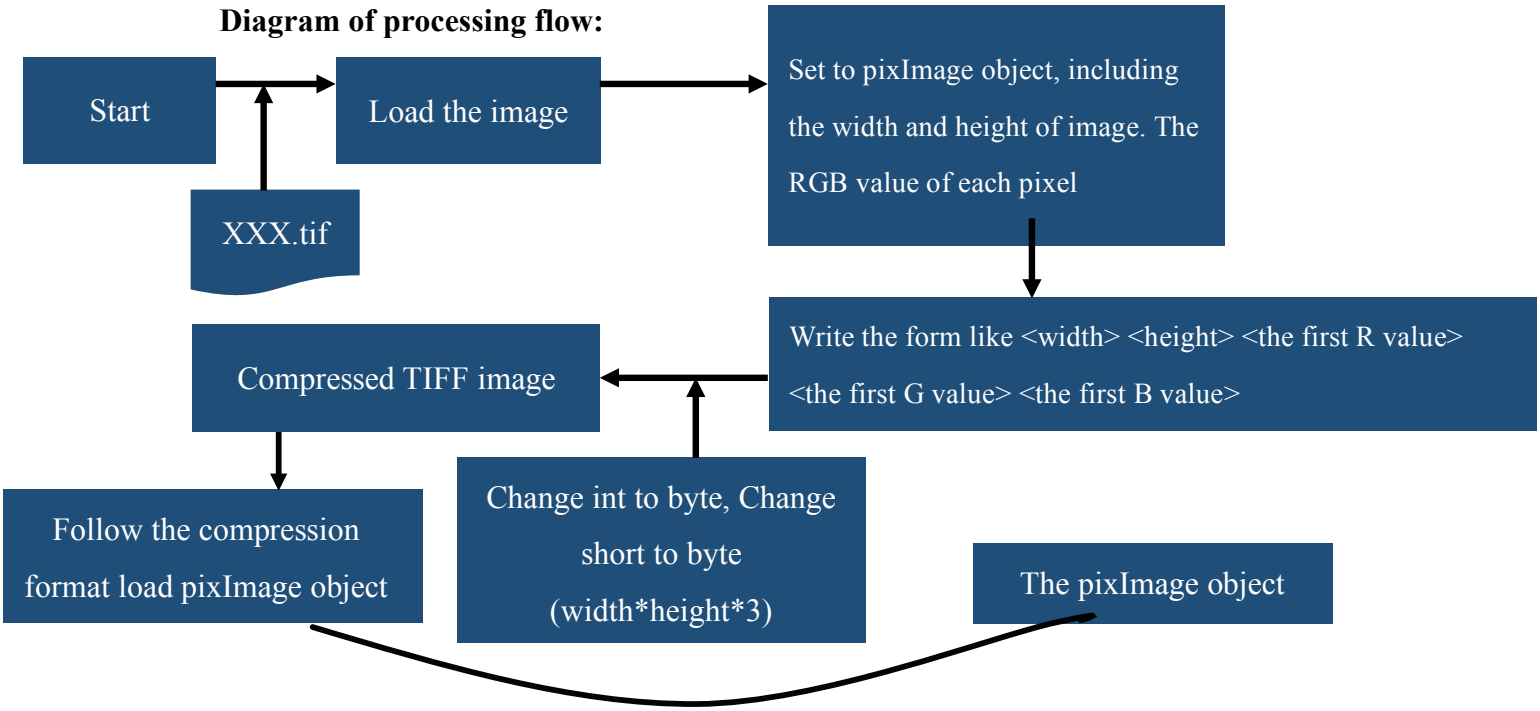
Initially I used the LZW compression method to compress, but for some files, especially the additional image files, the size of compressed image will be larger than the original image. After online searching, for some binary files, such as the streaming media and high compression ratio of the picture, they are already compressed by using some compression algorithms to compress. However, they will cause the size of original file will become larger after compression operation. So I choose straight to the data structure approach method. The advantage of this is directly using data stream output is fine. The width and height of the image and the three values (RGB) of each pixel block are output directly to the file using binary streams. In this way, it will reduce the size of the image, but the disadvantage is that the compression ratio is not very high.

Key technique of decompressing the images:

We know that TIFF is one of the most complex image file formats in existence. A TIFF image file consists of three data structures: the file header, one or more directories called IFDs that contain marker Pointers, and the data itself. Since the compressed data of the three channels are all placed in the same array, it is necessary to first find the starting positions offset_g and offset_r of the G channel and the R channel. The 8bit data is first read twice during decompression and converted to width and height.

Then read the 16-bit data of width*height, extract the 5-bit binary number of each channel and move it to the left by 3 bits, and restore the low-bit 0 to 8 bits, so the data of the low-bit was lost. After the decompression, a value is taken from each channel can form a pixel at a time, until each channel is taken at the same time. The decompressed data is the original data before compression, and the lossless compression of the image is realized.

Diagram of processing flow:



The running time of each module:

The follow image is the running time of each module of board.tif.

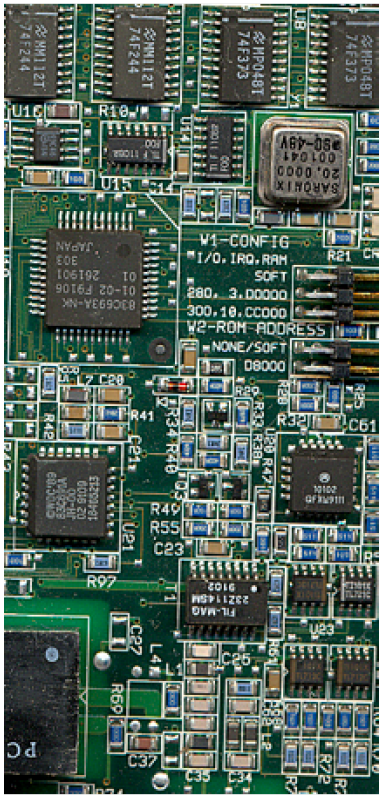
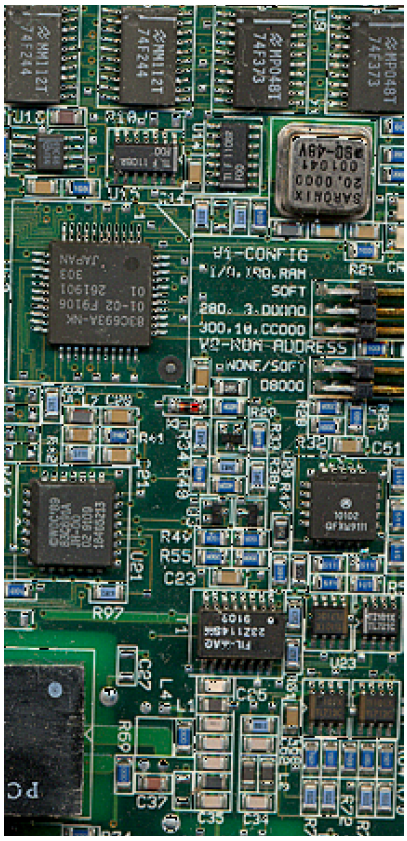
```
Running time of lossless-board.tif is 16110.59547
Running time of decompressing lossless-board.tif is 3041.0095
Running time of lossy-10 board.tif is 6718.03012
Running time of decompressing lossy-10 board.tif is 978.9564
Running time of lossy-20 board.tif is 5894.23074
Running time of decompressing lossy-20 board.tif is 971.79849
```

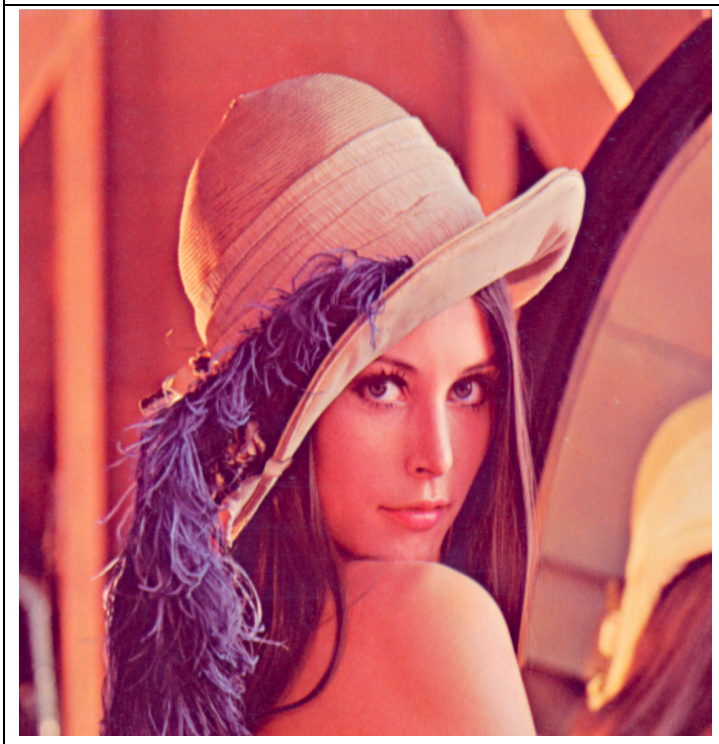
The follow table is the summary of each input TIFF image. Note that unit of all entries is millisecond (ms). All the measurement working on the same equipment.

	Board.tif	Lena.tif	Parrots.tif	Tiger.tif
Compress(lossless)	16110.59547	49329.08256	61068.35566	11453.80072
Decompress(less)	3041.0095	7798.53827	10954.65943	2204.04907

The way I measure them is I wrote some lines in the code and run them.

```
long start = System.nanoTime();long end = System.nanoTime();
long elapsedTime = end - start;
```

The original image (board.tif)	Decompression (Lossless compression) Ratio: 0.14%
	
The online converter	**Size summary**
	

The original image (lena.tif)	Decompression (Lossless compression) Ratio: 2%
	
The online converter	**Size summary**
	

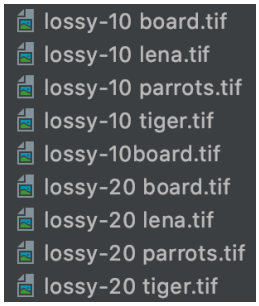
The original image (parrots.tif)	Decompression (Lossless compression) Ratio: 2%
	
The online converter	**Size summary**
	

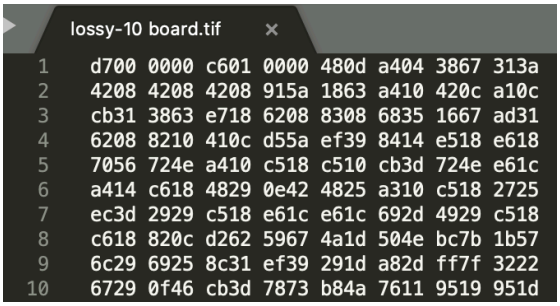
The original image (tiger.tif)	Decompression (Lossless compression) Ratio: 2%
	
The online converter	**Size summary**
	

Through all above comparisons, it is clearly to see the online converter has the best quality and compression rate. Also the Photoshop is much smoother and controls are best. The lossless compression has a slight different with the original image. So this is the main disadvantage, also for the part of data conversion: Every pixel has three short value and Byte pixel should convert to three short value. This step will easily lead a rounding error. In the example of parrots.tif, it will lead some black zone, I need to perfect the details.

Lossy compression:

As the same as above, the binary file for each ratio equal to 10 and 20, they will show in the submission file. I make screenshots to show files and pick one as the example.



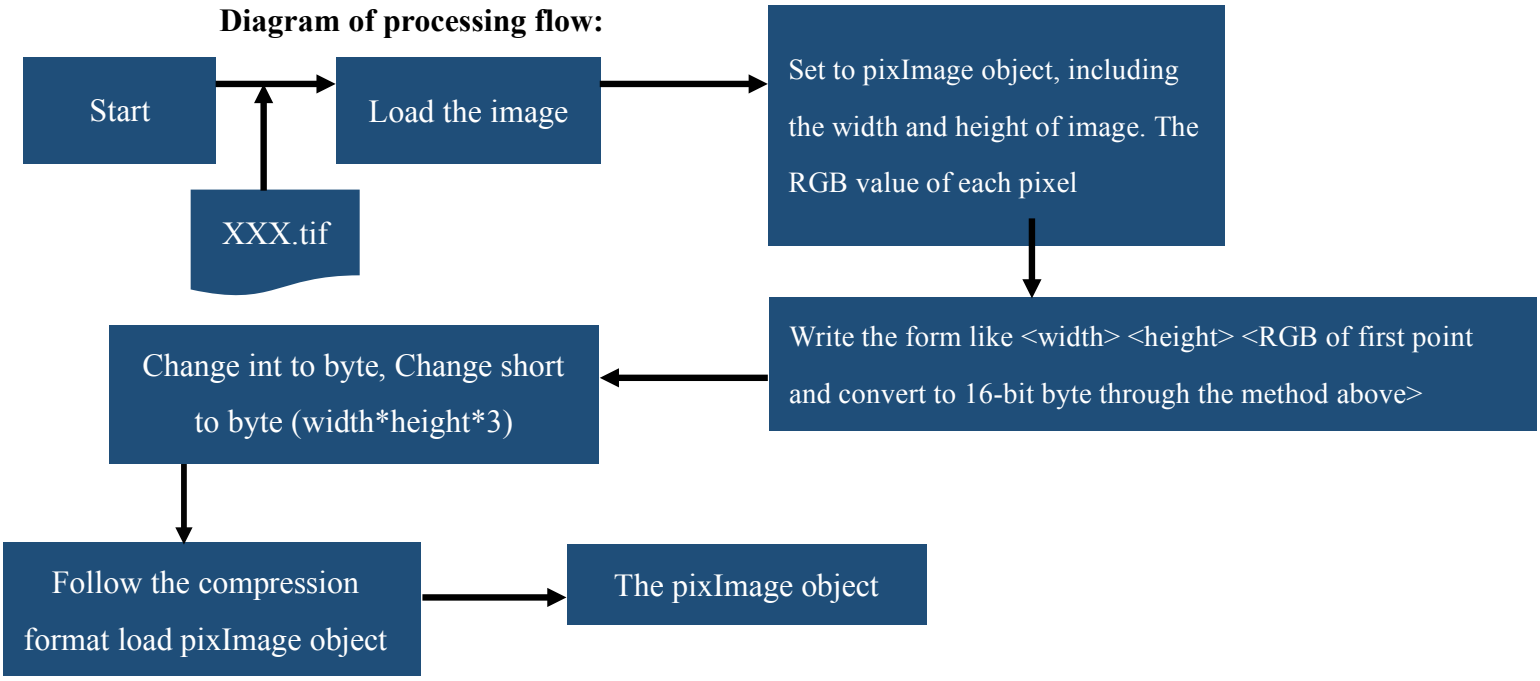


The same calculation ratio procedure as the mention above, so I am not show the same stuff again. The ratio will show in the resulting part.

Key technique of compress:

Lossy compression uses the 5-5-5 method for compression, converting 24-bit RGB to 16-bit values, so that compression can be achieved. When compressing, 5-5-5 moves the binary values of each channel to the right by 3 bits (divided by 8), keeping the remaining 5 bits, and then putting the lower 15 bits of the 16 bits in turn. During decompression, the 5-bit binary number of each channel is taken out and moved to the left by 3 bits, and the low-bit complement 0 is restored to 8 bits, so the low-bit data is lost. The Key technique of decompressing the images as mentioned before, all points are same. So I am not mention here again. The advantage of this method is that the image can be compressed simply through the displacement of RGB values, and the steps of image compression can be more guaranteed without damaging the overall structure of the image. And the compression ratio has been significantly improved. The disadvantage is that the compression ratio cannot be increased again, and the stability of this method is not very well.

Diagram of processing flow:



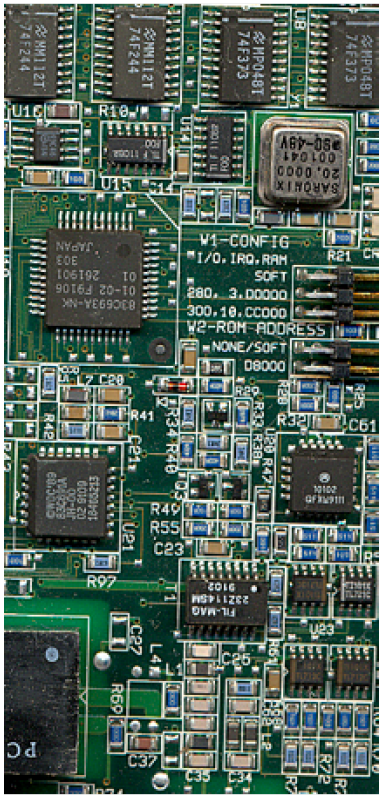
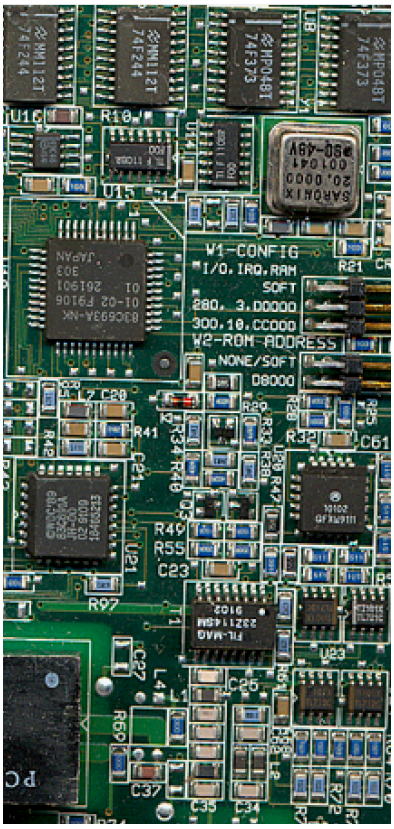
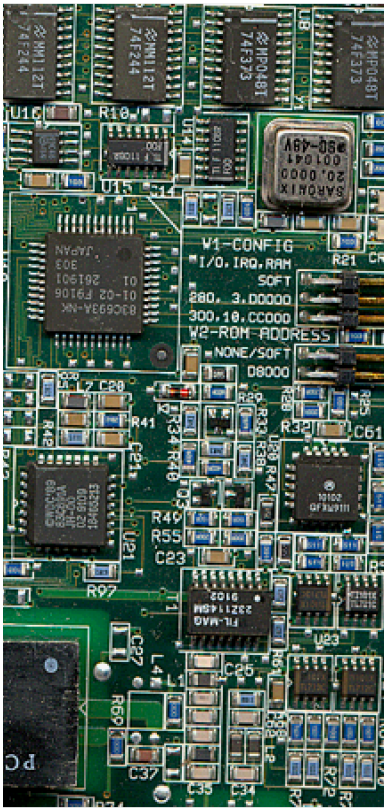
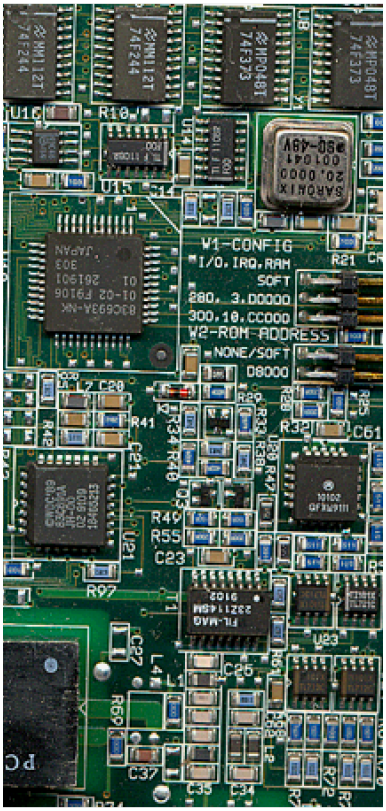
The running time of each module:

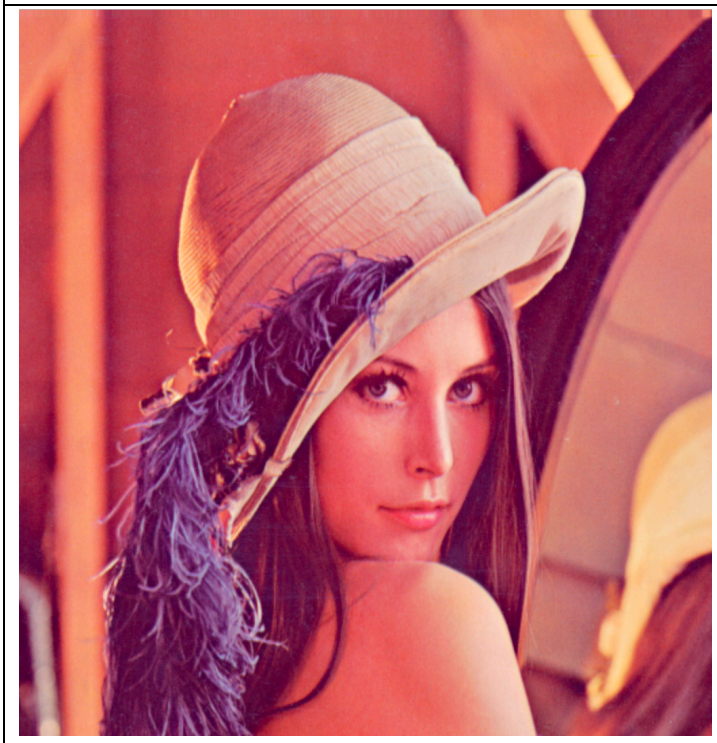
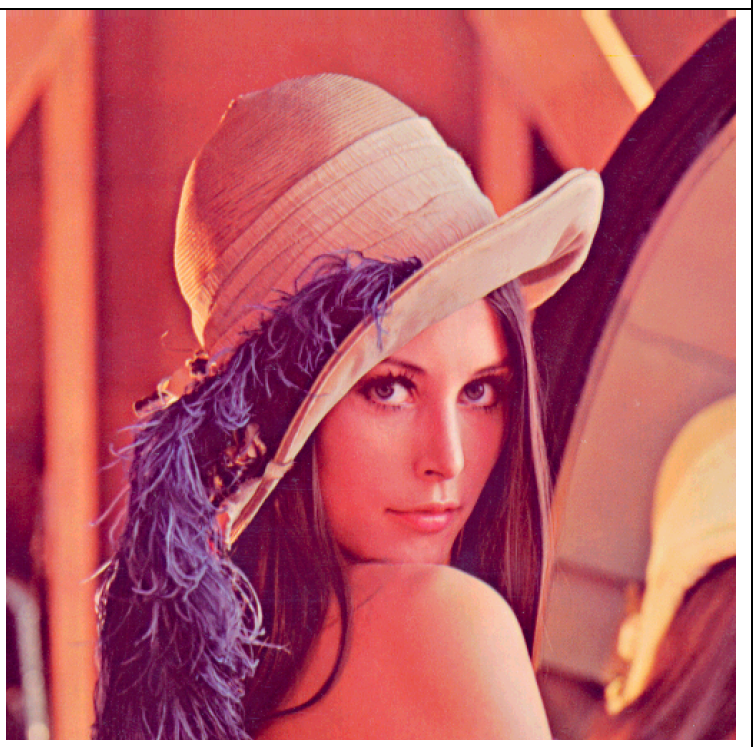
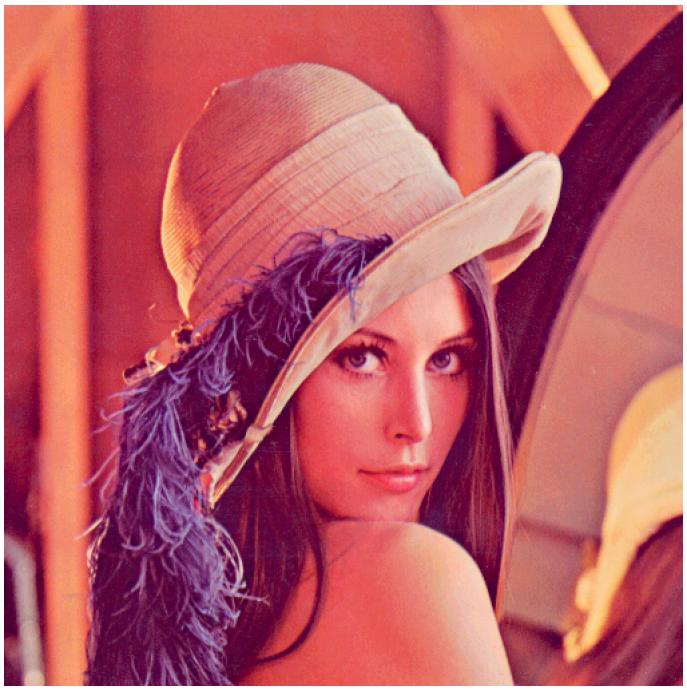
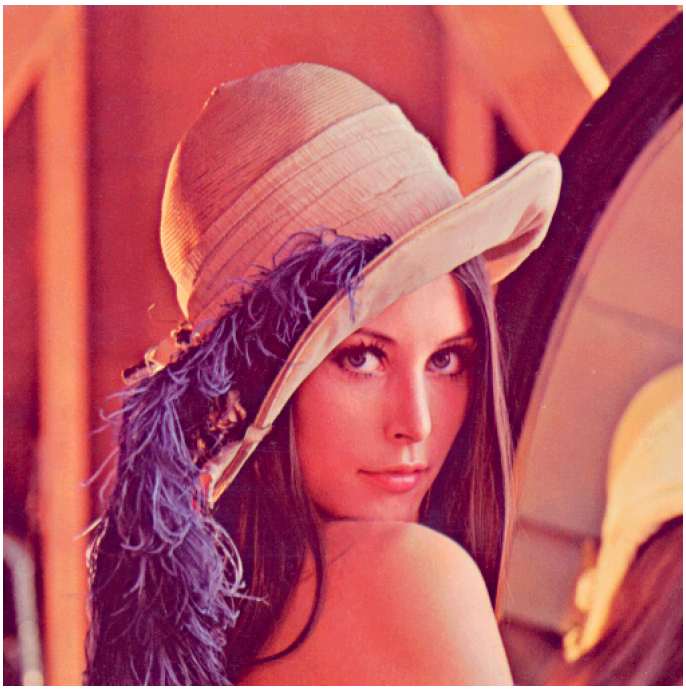
The follow image is the running time of each module of tiger.tif.

```
Running time of lossless-tiger.tif is 11453.80072
Running time of decompressing lossless-tiger.tif is 2204.04907
Running time of lossy-10 tiger.tif is 5649.55005
Running time of decompressing lossy-10 tiger.tif is 1276.77909
Running time of lossy-20 tiger.tif is 4769.99753
Running time of decompressing lossy-20 tiger.tif is 828.6126
```

The method of this part as same as the above. I just show the running time related to this part

	Board.tif	Lena.tif	Parrots.tif	Tiger.tif
Compress-10(lossy)	6718.03012	18558.88881	29308.55185	5649.55005
Decompress(Lossy)	978.9564	2589.1435	4029.86305	1276.77909
Compress-20(lossy)	5894.23074	16105.44728	22897.01181	4769.99753
Decompress(Lossy)	971.79849	2586.24242	3984.60659	828.6126

The original image (board.tif)	(Lossy compression-10) Ratio: 10.26%
	
Decompression (20)	(Lossy compression-20) Ratio: 20.23%
	

The original image (lena.tif)	(Lossy compression-10) Ratio: 10.04%
	
Decompression (20)	(Lossy compression-20) Ratio: 20.07%
	

The original image (parrots.tif)	(Lossy compression-10) Ratio: 10.05%
	
Decompression(20)	(Lossy compression-20) Ratio: 20.07%
	

The original image (tiger.tif)	(Lossy compression-10) Ratio: 10.02%
	
Decompression (20)	(Lossy compression-20) Ratio: 20.02%
	

There are sixteen images above and the decompression using the compression ratio of 20. As we can see the performance of lossy compression is better and the compressed image size looks very well. I just use the tiger as our example and we can clearly see the size is decreasing in the follow picture.

Although it will can not reach the level of online converter, it has a better performance than lossless compression.

If we want to use traditional way to do the lossy compression

Step1: The image is divided into non-overlapping 8 by 8 blocks. If the image width or size does not divide evenly into 8, the image may be cropped or pixels may be added to make the image divisible by 8.

Step2: The DCT method is applied to each 8 by 8 block. For the purposes of applying the DCT, the values in each block are first shifted by -128 to center around zero.

Step3: Make a nice quantization matrix and using simple division to get Quantized DCT coefficients

Step 4: Follow the Zig-zag scan rule to perform is entropy encoding

The main question or idea is how to choose the best compression technique? It really depends on the application. We need to consider the image quality, operational bit rate, complexity, channel error…Different image input also affect the quality of compressing

Self-examination:

Also because I am not familiar with TIFF parsing rules, so decompression can only start from the file level, here is also the use of user-defined image width height and pixel value storage structure for storage, which brings the consequence is only their own parsing and compression ratio fixed. Whatever the lossless compression or lossless compression or decompression that I'm doing here, cannot as high and as smooth as any existing image processing software. Also, I did not use the LZW for lossless

compression and DCT and Huffman coding here, the reason is after doing some preliminary research, I discovered that there are better options for showing lossless compression and lossy compression. Another point is I can only read the image in the project dictionary, not in the whole laptop. What's more, in the process of these two compresses, decompressing the file was still a problem. I was not able to uncompressed the file through the TIF file format very well. This is the part I need to focus and improve.

Reference

Note: These references are about all images that additional adding, some ideas about lossless compression, lossy compression and decompression.

https://forums.adafruit.com/viewtopic.php?f=47&t=35042

https://poi.apache.org/apidocs/dev/org/apache/poi/ss/util/ImageUtils.html

https://docs.oracle.com/middleware/1213/adf/api-reference-dvt/oracle/dss/util/ImageUtils.html

https://www.onlineconverter.com/convert/23ca06122901426fa678c3cc4a50cf100f

https://www.math.cuhk.edu.hk/~lmlui/CompressionTutorial.pdf

https://docs.gimp.org/en/gimp-tutorial-quickie-jpeg.html

https://forum.image.sc/t/save-lzw-compressed-tiff-in-java/4041/2

https://coderanch.com/t/611164/java/Reduce-image-size-converting-TIFF

https://docs.oracle.com/cd/E17802_01/products/products/java-media/jai/forDevelopers/jai-apidocs/com/sun/media/jai/codec/TIFFEncodeParam.html