

# JPEG-Algorithm

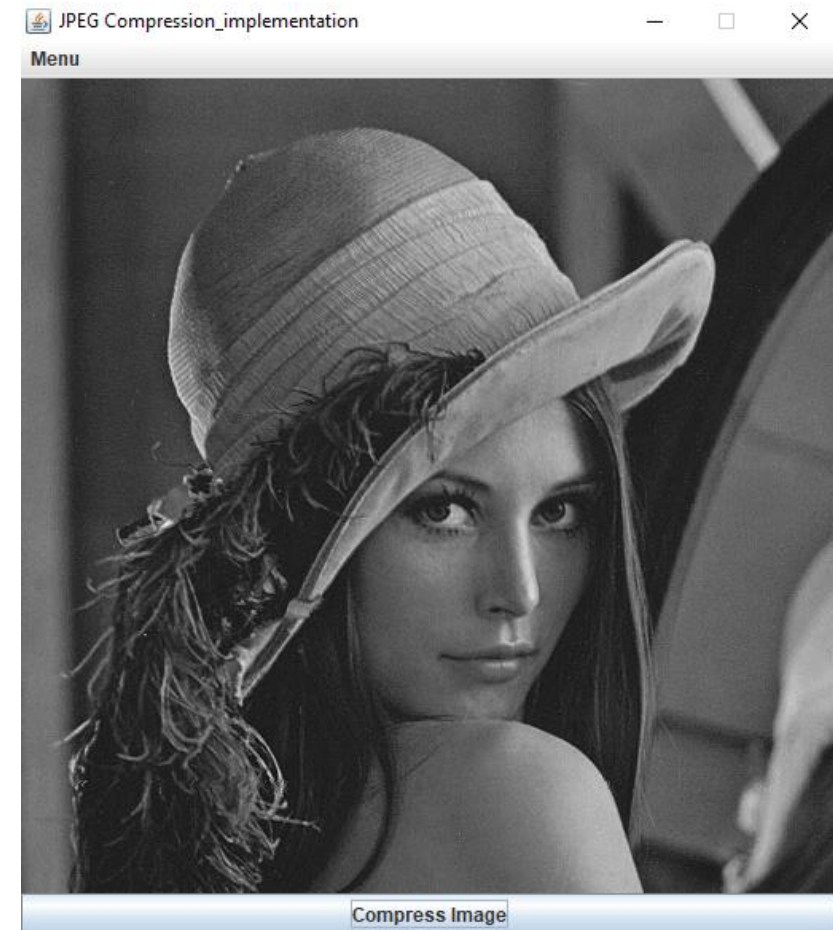
Simonluca Merlante, Ulrike Niederstätter, Stephan Unterrainer

# Introduction

- JPEG-algorithm using JAVA environment.
- Integration with Image-processing library OpenCV.
- Simple GUI
- Only grey-scale images.

# GUI

- JavaSwing
- Simple Menu with a FileChooser
- Default image



# Compression-Part 1

– Compression algorithm is divided into multiple steps.

1. Convert the image to a Matrix, and in 8x8 blocks for processing.

**public static Mat imgToMat(String path);**

2. DCT: transform the Mat from spatial domain into frequency domain.

**Core.dct(m, dct\_convert);**

3. Quantization: for each block (8x8) punctual division

**public static List<Mat> quanise(List<Mat> mat);**

4. Zig Zag scanning: to find sequences of zeros

```
public static List<double> [] zigZag(List<Mat> mat) { //gets each block 8*8 block of DCT
    List<double> [] zigZagResult = new ArrayList<>();

    for (Mat m : mat)
        zigZagResult.add(zigZagMatrix(m));

    return zigZagResult;
}

public static double [] zigZagMatrix(Mat mat) {

    double [] result= {mat.get(0,0)[0],mat.get(0,1)[0], mat.get(1,0)[0],mat.get(2,0)[0],mat.get(1,1)[0],mat.get(0,2)[0],mat.get(0,3)[0],mat.get(1,2)[0],mat.get(2,1)[0], mat.get(3,0)[0],
        mat.get(4,0)[0], mat.get(3,1)[0], mat.get(2,2)[0],mat.get(1,3)[0],mat.get(0,4)[0],mat.get(0,5)[0],mat.get(1,4)[0],mat.get(2,3)[0],mat.get(3,2)[0],mat.get(4,1)[0],
        mat.get(5,0)[0], mat.get(6,0)[0], mat.get(5,1)[0],mat.get(4,2)[0],mat.get(3,3)[0],mat.get(2,4)[0],mat.get(1,5)[0],mat.get(0,6)[0],mat.get(0,7)[0],mat.get(1,6)[0],
        mat.get(2,5)[0], mat.get(3,4)[0], mat.get(4,3)[0],mat.get(5,2)[0],mat.get(6,1)[0],mat.get(7,0)[0], mat.get(7,1)[0],mat.get(6,2)[0],mat.get(5,3)[0],mat.get(4,4)[0],
        mat.get(3,5)[0], mat.get(2,6)[0], mat.get(1,7)[0],mat.get(2,7)[0],mat.get(3,6)[0],mat.get(4,5)[0],mat.get(5,4)[0],mat.get(6,3)[0],mat.get(7,2)[0],mat.get(7,3)[0],
        mat.get(6,4)[0], mat.get(5,5)[0], mat.get(4,6)[0],mat.get(3,7)[0],mat.get(4,7)[0],mat.get(5,6)[0],mat.get(6,5)[0],mat.get(7,4)[0],mat.get(7,5)[0],mat.get(6,6)[0],
        mat.get(5,7)[0], mat.get(6,7)[0], mat.get(7,6)[0],mat.get(7,7)[0]};

    return result;
}
```

# Compression-Part 2

– Now the encoding part takes place.

5. Separating DC and AC-elements perform RLE for DC.

- `public static JPEGCategory RLEDC(double DCElement);`
- `Coefficient; Category; Precision; Runlength;`

6. Assign a category to each DC element and AC element.

`public static JPEGCategory assignCategory(double coeff)`

7. Perform RLE for each AC-element.

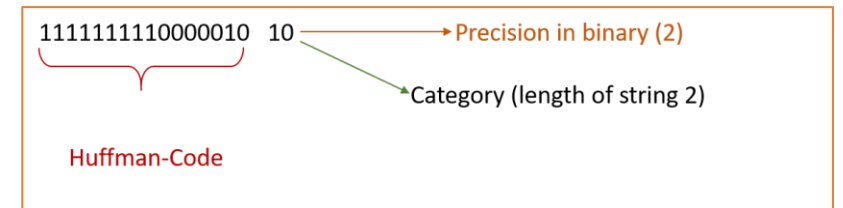
`public static ArrayList<JPEGCategory> RLE (double [] arr);`

8. Huffman encoding using the usual table for DC and AC-elements.

# Decompression-Part 1

- Simulate reception of signal, perform decompression immediate after the compression.
- Likewise as the compression, decompression divided into same steps:

## INVERSE



1. List of binary Strings from Huffman organising into blocks of Strings (EOB)

```
public static List <String []> createBlocks(List <String > encodedList);
```

2. Isolate the DC-elements and the AC-elements and perform prediction.

```
this.coeff = error + Utils.getDecodedPred();
```

3. Zig Zag inverse and convert JPEGCategory to a full array.

- ```
public static Mat invert(JPEGCategory [] input).
```
- ```
public static double [] convertToDouble (JPEGCategory input).
```

# Decompression-Part 2

4. Dequantization: invert process of Quantization, multiplying each block with the quantisation matrix.
5. Inverse DCT: using inverse build-in function of openCV.
6. Convert each block back to Matrix and back to image.

`Imgcodecs.imwrite(„compressed_image.png“, finalIMG);`

# Evaluation

## Test Case 1:

Image: **Lena-grey.png**

Quality Factor: **100**



Original



Compressed

Compression power:  $165/148 \text{ Kbyte} = 1,11$

Bitrate:  $8/1,11 = 7,20 \text{ bpp}$



# Evaluation

## Test Case 2:

Image: **Lena-grey.png**

Quality Factor: **85**



Original



Compressed

Compression power:  $165/62,1 \text{ Kbyte} = 2,65$

Bitrate:  $8/2,65 = 3,01 \text{ bpp}$

# Evaluation

## Test Case 3:

Image: **Barbara.tif**  
Quality Factor: **100**



Original



Compressed

Compression power:  $89\text{KB}/50\text{KB} = 1,78$   
Bitrate:  $8\text{bpp}/1,78 = 4,494\text{bpp}$

# Evaluation

## Test Case 4:

Image: **Barbara.tif**

Quality Factor: **85**



Original



Compressed

Compression power:  $89\text{KB}/32\text{KB} = 2,781$

Bitrate:  $8\text{bpp}/2,781 = 2,877\text{bpp}$

Thank you!