Image Compression Simulation

by

Zhihao DAI

COP501 Advanced Programming
Coursework Report

Loughborough University

Oct. 2019

Abstract

In this coursework, I implement a JPEG Image Compression Simulation using MATLAB as frontend GUI and Python as backend JPEG CODEC. There are 2 simulation parameters K and Q' in the application. Several specific design considerations are introduced to the implementation, inclding an end-to-end MATLAB interface, an ”Video Compression” functionality and DCT as Matrix Computation. I conclude that both K and Q' can significantly affect the quality of the compressed image.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Image Compression

Image compression is to decrease the size of the image file without dramatically downgarding the quality of the image.

In this courework, I set out to implement a simulation of the JPEG (Joint Photographic Experts Group) image compression process[1]. The implementation uses MATLAB as frontend interface and Python as backend.

1.2 JPEG Standard

The JPEG Still Picture Compression Standard is widely used on modern digital devices, ranging from computers, cameras to smartphones. For a grayscale image, a JPEG CODEC (Encoder and Decoder) typically involves 6 main steps as illustrated in Figure 1.1.

In my application, I implement a simplified JPEG CODEC in Python programming language. The CODEC consists of both foward and inverse steps of Discrete Cosine Transform (DCT) and Quantization and gets rid of both forward and inverse steps of Entropy Coding.

Since Entropy Codeing is reversible through Inverse Entropy Coding and does not affect the quality of the image, my implementation should be able to simulate the full effects of JPEG coding on any given still image despite its simplicity.

As shown in Figure 1.2, given an input grayscale image in uint8 matrix format, the application first crops the image array to multiply of 8 in both column (width) and row (height). Then, a value of 128 is subtracted from the image for each pixel value. After that, the matrix is divided into 8 by 8 blocks. Each block then goes through FDCT, Quantization, Inverse Quantization and IDCT individually. All resulted blocks are placed in their previous

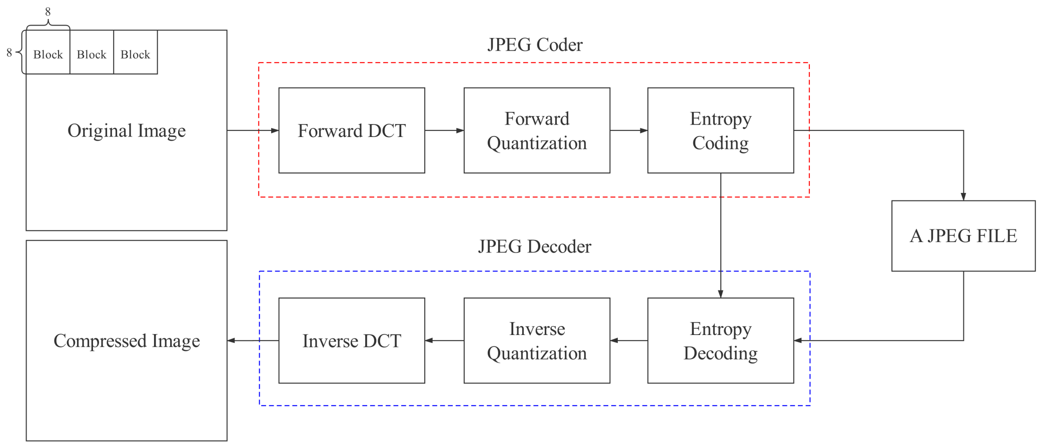

Figure 1.1: Six Main Steps of a JPEG CODEC.

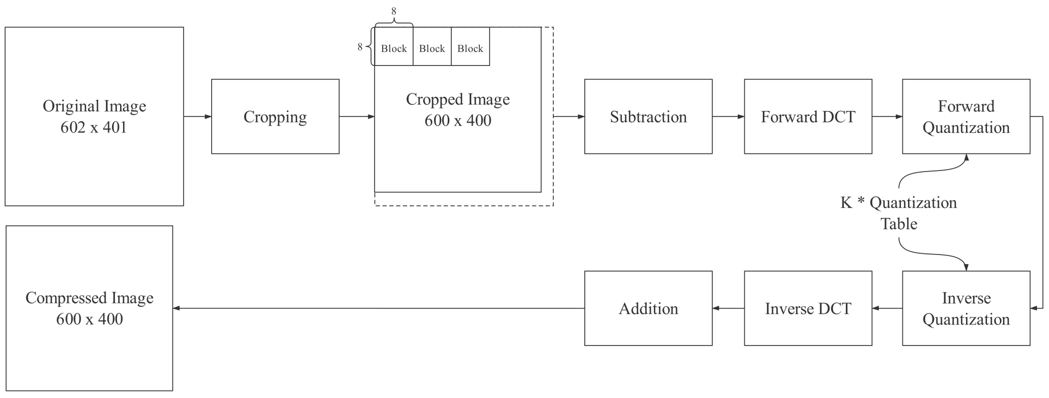

Figure 1.2: Main Steps of the Application.

position and a new matrix is thus formed. Finally, a value of 128 is added back to the new matrix to get the compressed grayscale image.

For a color RGB image, the same process is carried out on each color channel respectively.

1.2.1 Discrete Cosine Transform (DCT)

The forward step of DCT is computed using the following equation.

$$F(u,v) = \frac{1}{4}C(u)C(v)[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16}] \tag{1.1}$$

where $C(t) = 1/\sqrt{(2)}$ for $t = 0$; $C(t) = 1$ otherwise. Both input f and output F is an 8 by 8 block.

The inverse step of DCT is computed using the following equation.

$$f(x,y) = \frac{1}{4}[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v) * cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16}] \tag{1.2}$$

where $C(t) = 1/\sqrt(2)$ for $t = 0$; $C(t) = 1$ otherwise. Both input F and output f is an 8 by 8 block.

1.2.2 Quantization

The forward step of Quantization is computed using the following equation.

$$F^Q(u,v) = round(\frac{F(u,v)}{Q(u,v)}) \tag{1.3}$$

where Q is the Quantization Table specified in the JPEG standard. Both input F and ouput F^Q is an 8 by 8 block.

The inverse step of Quantization is computed using the following equation.

$$F(u,v) = F^Q(u,v) * Q(u,v)) \tag{1.4}$$

where Q should be the same as in the forward step. Both input F^Q and ouput F is an 8 by 8 block.

By default, the value of Q is specfied as followed.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{1.5}$$

References

[1] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

Appendix A

Source Code

A.1 JPEG CODEC Code in Python

A.1.1 matlab.py

```python
# matlab.py
'''
A Python-MATLAB Interface to "compression" package.
'''
from compression.CompressionCodecs import CompressionCodecs
import numpy as np

def compress(x, shape, k, qTable):
  '''
  Compress an image x through CompressionCodecs
  in package "compression".

  Arguments:
    x -- A flattened 1-D array representing
      the input image.
    shape -- Original shape of x before being
      flattened.
    k, qTable -- Arguments for Quantization, see
      QuantizationCodec in compression.Codec
      for further description.
  '''
  # Reshape x back into its original shape.
  x = np.reshape(x, shape, 'F')
  # Reshape qTable back into shape (8, 8).
  qTable = np.reshape(qTable, [8, 8], 'F')
```

```
# Compress the image through CompressionCodecs.
c = CompressionCodecs(k=k, qTable=qTable)
y = c.compress(x)
# Return the compressed image.
return y
```