**Министерство науки и высшего образование Российской Федерации**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ИОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЕ

**Национальный исследовательский университет ИТМО**

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРСМАЦИОННЫХ ТЕХНОЛОГИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

**ЛАБОРАТОРНЯ РАБОТОТА №2**

**По дисциплине Введение в цифровую культуры и программирования**

**Название работы: Эффективное кодирование**

Выполнил: Тарасов Михаил Евгеньевич

Проверил: Страдина Марина Владимировна



Санкт- Петербург, 2020 г.

**Эффективное кодирование**

Картинки до и после обработки 128х128 пиксилей:

*До:*   *После:*

Код для получения цифровой последовательности квантования строки.

import numpy

import numpy as np

from PIL import Image

image = Image.open('палец.jpg')

image = image.convert('L') # Перевод картинки в чёрно-белое

image.save("палец чб.jpg") # Сохранение картинки для работы

imgp = numpy.array(image)

imgp = imgp[64] # Расчёт средней строки

for i in range(128): # Квантования

imgp[i] = round(imgp[i] // 20) \* 20

for i in range(128): # Вывод значений

print(imgp[i], end=' ')

print()

print() # Отступ

for i in range(127): # Сортировка

for j in range(127-i):

if imgp[j] > imgp[j+1]:

imgp[j], imgp[j+1] = imgp[j+1], imgp[j]

for i in range(128):

print(imgp[i], end=' ')

print("")

Вывод до сортировки:

220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 240 240 240 240 240 240 240 240 240 240 240 240 200 100 100 100 100 80 140 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 80 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 80 80 60 80 100 100 100 100 100 100 100

После:

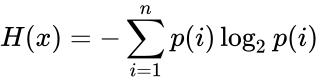
60 80 80 80 80 80 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 140 200 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 240 240 240 240 240 240 240 240 240 240 240 240

Кол-во различных значений – 7.

|  |  |  |
| --- | --- | --- |
| Номер | Кол-во пикселей цвета | Вероятность |
| 240 | 20 | 0,15625 |
| 220 | 2 | 0,015625‬ |
| 200 | 1 | 0,0078125 |
| 140 | 2 | 0,015625‬ |
| 120 | 1 | 0,0078125 |
| 160 | 1 | 0,015625‬ |
| 180 | 1 | 0,03125 |

Расчёт энтропии:

Формула для расчета энтропии:



Значение энтропии = 1,828931 (шесть знаков после запятой)

Метод Шеннона:

|  |  |  |
| --- | --- | --- |
| Номер | Дробь | Код |
| 240 | 0.0000000 | 0 |
| 0 | 0.1010010 | 10 |
| 220 | 0.1100110 | 110 |
| 200 | 0.1101111 | 11101 |
| 180 | 0.1110100 | 11110 |
| 160 | 0.1111000 | 111100 |
| 60 | 0.1111010 | 111101 |
| 20 | 0.1111100 | 111110 |
| 120 | 0.1111110 | 1111110 |
| 40 | 0.1111111 | 1111111 |

Метод Хаффмана:

|  |  |  |
| --- | --- | --- |
| Номер | Кол - во | Код |
| 0 | 20 | 11 |
| 20 | 2 | 011000 |
| 40 | 1 | 0110100 |
| 60 | 2 | 011001 |
| 120 | 1 | 0110101 |
| 160 | 2 | 011011 |
| 180 | 4 | 01110 |
| 200 | 5 | 01111 |
| 220 | 9 | 010 |
| 240 | 82 | 0 |

Закодированное равномерным кодом сообщение:

Длина сообщения: 512

Размер сообщения: 64 байтов

Длина кодового слова: 4 бита

100110011001100110011001100110011001100110011001100110011001100110011001001100000000000000000000000001011001100110011001100110011001100110011001100010001000011100000100‬10011001100110011001100110011001100110011001100110011001100110010010‬000000000000100010011001100110011001100101100000000000000011100110011001100110011001100100010000000000001001100110011001100110011001100110001000011100000000000000000001011001010110011001110111011110001000100010011001100110011001100110011001100110011001100110011001100110011001

Код Python:

for i in range(128):

if imgp[i] == 0:

print('0000', end='')

if imgp[i] == 20:

print('0001', end='')

if imgp[i] == 40:

print('0010‬', end='')

if imgp[i] == 60:

print('0011', end='')

if imgp[i] == 120:

print('0100‬', end='')

if imgp[i] == 160:

print('0101', end='')

if imgp[i] == 180:

print('0110', end='')

if imgp[i] == 200:

print('0111', end='')

if imgp[i] == 220:

print('1000', end='')

if imgp[i] == 240:

print('1001', end='')

Закодированное методом Шеннона сообщение:

Длина сообщения: 244

Размер сообщения: 31 байтов

Длина кодового слова: 1, 90625 бита

0000000000000000001111011010101010101111000000000000110110110111011011111100000000000000000111111110101011000000011101101010111101000000011111010101000000000110110111011010101011111011101111100111011110111101111011110111011011000000000000000000

Код Python:

for i in range(128): #Кодирование методом Шеннона

if imgp[i] == 0:

print('10', end='')

if imgp[i] == 20:

print('111110', end='')

if imgp[i] == 40:

print('1111111', end='')

if imgp[i] == 60:

print('111101', end='')

if imgp[i] == 120:

print('1111110', end='')

if imgp[i] == 160:

print('111100', end='')

if imgp[i] == 180:

print('11101', end='')

if imgp[i] == 200:

print('11101', end='')

if imgp[i] == 220:

print('110', end='')

if imgp[i] == 240:

print('0', end='')

Закодированное методом Хаффмана сообщение:

Длина сообщения: 244

Размер сообщения: 31 байтов

Длина кодового слова: 1,90625 бита

0000000000000000000110011111111111110110110000000000010010010011111101101010000000000000000011010011111101000000001110111111011001000000001100011111100000000010010011111111111101100001110011011011100111001111011110111101001001000000000000000000

Код Python:

for i in range(128): #Кодирование методом Хаффмана

if imgp[i] == 0:

print('11', end='')

if imgp[i] == 20:

print('011000', end='')

if imgp[i] == 40:

print('0110100', end='')

if imgp[i] == 60:

print('011001', end='')

if imgp[i] == 120:

print('0110101', end='')

if imgp[i] == 160:

print('011011', end='')

if imgp[i] == 180:

print('01110', end='')

if imgp[i] == 200:

print('01111', end='')

if imgp[i] == 220:

print('010', end='')

if imgp[i] == 240:

print('0', end='')

Оценка степени сжатия при разных методах кодирования

1. Степень сжатия кода сообщения Шеннона *K1=0,48 (округленно)*  
   K1 = число знаков в исходном сообщении, которое закодировано кодом Шеннона / число знаков в исходном сообщении, которое закодировано равномерным кодом.

Средняя длина кода Шеннона *1,90625 бит.*

Относительная избыточность кода Шеннона *Q1=0,0405608.*

1. Степень сжатия кода сообщения Хаффмана K2 =*0,48 (округленно)*  
   K2 = число знаков в исходном сообщении, которое закодировано кодом Хаффмана / число знаков в исходном сообщении, которое закодировано равномерным кодом.

Средняя длина кода Хаффмана *1,90625 бит.*

Относительная избыточность кода Хаффмана *Q2=0,0405608.*

Вывод:

Мы проанализировали два метода кодирования и выяснили, что степень сжатия кода сообщения и относительная избыточность кода при использовании метода Шеннона и метода Хаффмана соответственно равны. Помимо этого, мы поняли, что использование вышеуказанных способов кодирования информации гораздо эффективнее, чем кодирования той же информации с помощью равномерного двоичного кода.