**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 5**

**по дисциплине «Цифровая обработка изображений»**

**Тема: РАЗРАБОТКА ПРОГРАММЫ ДЛЯ**

**СЖАТИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9105 |  | Шаривзянов Д. Р. |
|  |  | Басманов А. А. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2024

**РАЗРАБОТКА ПРОГРАММЫ ДЛЯ**

**СЖАТИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ**

Код программы:

#include <iostream>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\imgproc\imgproc.hpp>

#include <vector>

using namespace std;

using namespace cv;

struct Image {

Mat bgr;

Mat gray;

int compression;

Mat dct\_orig;

Mat dct\_decoded;

Mat gray\_decoded;

};

Mat getHist(const Mat &src) {

int hist\_h = 400, hist\_w = 256\*3;

Mat hist = Mat::zeros(1, 256, CV\_64FC1);

for (int i = 0; i < src.cols; i++)

for (int j = 0; j < src.rows; j++) {

int r = src.at<unsigned char>(j, i);

hist.at<double>(0, r) = hist.at<double>(0, r) + 1.0;

}

double m = 0, M = 0;

minMaxLoc(hist, &m, &M);

hist = hist / M;

Mat hist\_img = Mat::zeros(100, 256, CV\_8U);

for (int i = 0; i < 256; i++)

for (int j = 0; j < 100; j++) {

if (hist.at<double>(0, i) \* 100 > j) {

hist\_img.at<unsigned char>(99 - j, i) = 255;

}

}

bitwise\_not(hist\_img, hist\_img);

resize(hist\_img, hist\_img, Size(hist\_w, hist\_h), 0 , 0, INTER\_NEAREST);

return hist\_img;

}

vector<double> probality(const Mat &src) {

vector<double> prob(256);

for (int i = 0; i < src.cols; i++)

for (int j = 0; j < src.rows; j++) {

int r = src.at<uchar>(j, i);

prob[r] += 1.0;

}

for (int i = 0; i < prob.size(); i++) prob[i] /= (src.rows\*src.cols);

return prob;

}

double entropy(vector<double> &prob) {

double H = 0.0;

for (int i = 0; i < prob.size(); i++)

if (prob[i] != 0) H -= prob[i] \* log2(prob[i]);

return H;

}

double redundancy(double H) {

return 1 - (H / log2(256));

}

void gammaDivision(Mat &dst, Mat &gamma, int block\_size) {

for (int row = 0; row < block\_size; row++)

for (int col = 0; col < block\_size; col++)

dst.at<double>(row, col) /= gamma.at<double>(row, col);

}

void gammaMultiply(Mat &dst, Mat &gamma, int block\_size) {

for (int row = 0; row < block\_size; row++)

for (int col = 0; col < block\_size; col++)

dst.at<double>(row, col) \*= gamma.at<double>(row, col);

}

void makeGamma(Mat &dst, int quality, int block\_size) {

dst = Mat::zeros(block\_size, block\_size, CV\_64FC1);

for (int row = 0; row < block\_size; row++)

for (int col = 0; col < block\_size; col++)

dst.at<double>(row, col) = block\_size + (row + col) \* quality;

}

void createBasisMat(Mat &basisMat, int block\_size) {

basisMat = Mat::zeros(block\_size, block\_size, CV\_64FC1);

for (int row = 0; row < block\_size; row++)

for (int col = 0; col < block\_size; col++) {

if (row == 0) basisMat.at<double>(row, col) = 1 / sqrt(block\_size);

else if (row > 0) basisMat.at<double>(row, col) = sqrt(2. / block\_size) \* cos(((CV\_PI \* row) / block\_size) \* (col + 0.5));

}

}

void DCT\_direct(Mat &src, Mat &dst, int quality) {

int block\_size = 8;

int delta\_h8 = src.rows % block\_size;

int delta\_w8 = src.cols % block\_size;

dst = Mat::zeros(src.rows - delta\_h8, src.cols - delta\_w8, CV\_8UC1);

Mat img\_flt;

src.convertTo(img\_flt, CV\_64FC1);

Mat gamma;

makeGamma(gamma, quality, block\_size);

Mat basisMat;

createBasisMat(basisMat, block\_size);

//перебираем изображение по блокам 8х8

for (int block\_row = 0; block\_row < dst.rows; block\_row += block\_size)

for (int block\_col = 0; block\_col < dst.cols; block\_col += block\_size) {

//выделяем область 8х8 исходного изображения

Mat ROI8U = img\_flt(Rect(block\_col, block\_row, block\_size, block\_size));

// Преобразуем в double для ДКП

Mat ROI64F;

ROI8U.convertTo(ROI64F, CV\_64FC1);

//применяем DCT

Mat DCT64F = basisMat \* ROI64F \* basisMat.t();

//применяем гамму

gammaDivision(DCT64F, gamma, block\_size);

//собираем изображение по блокам 8х8

DCT64F.copyTo(dst(Rect(block\_col, block\_row, block\_size, block\_size)));

}

// Конвертируем в 8 бит

dst.convertTo(dst, CV\_8UC1);

}

void DCT\_inverse(Mat &src, Mat &dst, int quality) {

int block\_size = 8;

dst = Mat::zeros(src.size(), CV\_64FC1);

Mat gamma;

makeGamma(gamma, quality, block\_size);

Mat basisMat;

createBasisMat(basisMat, block\_size);

// Перебираем изображение по блокам 8х8

for (int block\_row = 0; block\_row < src.rows; block\_row += block\_size)

for (int block\_col = 0; block\_col < src.cols; block\_col += block\_size) {

// Выделяем область 8х8 из DCT-изображения

Mat DCT8U = src(Rect(block\_col, block\_row, block\_size, block\_size));

// Преобразуем в double для обратного ДКП

Mat DCT64F;

DCT8U.convertTo(DCT64F, CV\_64FC1);

// Применяем обратную гамму

gammaMultiply(DCT64F, gamma, block\_size);

// Выполняем обратное ДКП

Mat ROI64F = basisMat.t() \* DCT64F \* basisMat;

// Собираем изображение по блокам 8х8

ROI64F.copyTo(dst(Rect(block\_col, block\_row, block\_size, block\_size)));

}

// Конвертируем обратно в 8 бит

dst.convertTo(dst, CV\_8UC1);

}

vector<uchar> block2vec(const Mat& block) {

const uchar n = 8; // Размер блока 8x8

vector<uchar> output(n \* n, 0);

int i = 0, j = 0;

bool goingUp = true; // Флаг направления движения

for (int k = 0; k < n \* n; k++) {

output[k] = block.at<uchar>(i, j); // Использование типа uchar

if (goingUp) {

if (j == n - 1) {

i++; // Двигаемся вниз, если достигли правой границы

goingUp = false;

} else if (i == 0) {

j++; // Двигаемся вправо, если достигли верхней границы

goingUp = false;

} else {

i--;

j++;

}

} else {

if (i == n - 1) {

j++; // Двигаемся вправо, если достигли нижней границы

goingUp = true;

} else if (j == 0) {

i++; // Двигаемся вниз, если достигли левой границы

goingUp = true;

} else {

i++;

j--;

}

}

}

return output;

}

vector<uchar> mat2vec(const Mat& src) {

vector<uchar> output;

for (int i = 0; i < src.rows; i += 8) {

for (int j = 0; j < src.cols; j += 8) {

vector<uchar> block = block2vec(src(Rect(j, i, 8, 8)));

output.insert(output.end(), block.begin(), block.end());

}

}

return output;

}

vector<pair<uchar, uchar>> vec2RLE(vector<uchar>& data, int blockSize = 64) {

vector<pair<uchar, uchar>> encodedData;

int count = 0;

int prevValue = (data.empty() ? -1 : data[0]);

for (int i = 0; i < data.size(); ++i) {

if (data[i] == prevValue) {

count++;

} else {

if (count > 0) {

encodedData.emplace\_back(count, prevValue);

}

prevValue = data[i];

count = 1;

}

// Вставляем код конца блока после обработки каждого блока 8x8

if ((i + 1) % blockSize == 0) {

if (count > 0) {

encodedData.emplace\_back(count, prevValue);

count = 0;

}

// Код конца блока

encodedData.emplace\_back(255, 255);

prevValue = (i + 1 < data.size() ? data[i + 1] : -1);

}

}

// Добавляем оставшиеся данные, если они есть

if (count > 0 && (data.size() % blockSize) != 0) {

encodedData.emplace\_back(count, prevValue);

}

return encodedData;

}

vector<uchar> RLE2vec(vector<pair<uchar, uchar>>& rle) {

vector<uchar> decoded;

for (const auto& pair : rle) {

if (pair.first == 255 && pair.second == 255) {

// Окончание блока, но продолжаем обработку, если есть еще данные

continue;

}

for (int i = 0; i < pair.first; i++) {

decoded.push\_back(pair.second);

}

}

return decoded;

}

// Функция для преобразования зигзаг-последовательности в блок 8x8

Mat vec2block(vector<uchar>& zigzag, int startIdx) {

Mat block(8, 8, CV\_8UC1); // Используем double для хранения значений

vector<uchar> indexMap = {

0, 1, 5, 6, 14, 15, 27, 28,

2, 4, 7, 13, 16, 26, 29, 42,

3, 8, 12, 17, 25, 30, 41, 43,

9, 11, 18, 24, 31, 40, 44, 53,

10, 19, 23, 32, 39, 45, 52, 54,

20, 22, 33, 38, 46, 51, 55, 60,

21, 34, 37, 47, 50, 56, 59, 61,

35, 36, 48, 49, 57, 58, 62, 63

};

for (int i = 0; i < 64; ++i) {

int x = i / 8;

int y = i % 8;

block.at<uchar>(x, y) = zigzag[startIdx + indexMap[i]];

}

return block;

}

// Функция для создания изображения из всех блоков

Mat vec2mat(vector<uchar>& zigzag, int width, int height) {

CV\_Assert(width % 8 == 0 && height % 8 == 0);

Mat image(height, width, CV\_8UC1);

int blocksPerRow = width / 8;

int blocksPerColumn = height / 8;

int index = 0;

for (int i = 0; i < blocksPerColumn; ++i) {

for (int j = 0; j < blocksPerRow; ++j) {

Mat block = vec2block(zigzag, index);

block.copyTo(image(Rect(j \* 8, i \* 8, 8, 8)));

index += 64;

}

}

return image;

}

void lab5(const Mat &img\_bgr) {

Image img;

img.bgr = img\_bgr;

// resize(img.bgr, img.bgr, Size(300, 300), 0, 0, INTER\_CUBIC);

imshow("image bgr", img.bgr);

cvtColor(img.bgr, img.gray, COLOR\_BGR2GRAY);

imshow("image gray", img.gray);

imwrite("../../Images/Lab 5/image gray.jpg", img.gray);

vector <double> prob\_orig = probality(img.gray);

double H\_orig = entropy(prob\_orig);

cout << "H ref = " << log2(256) << '\t';

cout << "H orig = " << H\_orig << '\t';

double R = redundancy(H\_orig);

cout << "R orig = " << R << endl;

//-------------------------------------вычисление DCT-------------------------------

img.compression = 5;

DCT\_direct(img.gray, img.dct\_orig, img.compression);

imshow("image dct", img.dct\_orig);

imwrite("../../Images/Lab 5/image dct.jpg", img.dct\_orig);

Mat hist = getHist(img.dct\_orig);

imshow("Histogram", hist);

imwrite("../../Images/Lab 5/histogram.jpg", hist);

vector <double> prob\_dct = probality(img.dct\_orig);

double H\_dct = entropy(prob\_dct);

cout << "H dct = " << H\_dct << '\t';

double R\_dct = redundancy(H\_dct);

cout << "R dct = " << R\_dct << endl;

//-------------------------------------------кодирование-------------------------------------------

cout << "Total bytes " << img.dct\_orig.total() << endl;

vector <uchar> vec\_orig = mat2vec(img.dct\_orig);

cout << "ZigZag origin bytes " << vec\_orig.size() << endl;

vector<pair<uchar, uchar>> RLE = vec2RLE(vec\_orig);

cout << "Encoded bytes " << RLE.size() \* 2 << endl;

cout << "Compression ratio " << double(vec\_orig.size()) / (RLE.size() \* 2) << endl;

//-------------------------------------------декодирование-------------------------------------------

vector<uchar> vec\_decoded = RLE2vec(RLE);

cout << "ZigZag decoded bytes " << vec\_decoded.size() << endl;

img.dct\_decoded = vec2mat(vec\_decoded, img.dct\_orig.cols, img.dct\_orig.rows);

imshow("image dct decoded", img.dct\_decoded);

imwrite("../../Images/Lab 5/image dct decoded.jpg", img.dct\_decoded);

DCT\_inverse(img.dct\_decoded, img.gray\_decoded, img.compression);

imshow("image gray decoded", img.gray\_decoded);

imwrite("../../Images/Lab 5/image gray decoded.jpg", img.gray\_decoded);

waitKey();

}

Исходное изображение представлено на рис. 1.

Изображение выглядит как облако, статуя, мультфильм, небо

Автоматически созданное описание

Рис. 1. Портрет.

**Ход работы.**

1. Оценим возможности по энтропийному кодированию исходного изображения, квантованного на 256 уровней. Используем данные гистограммы распределения уровней яркости и просчитаем вероятности появления этих уровней; энтропию источника сообщения, а также избыточность сообщений.

Энтропия: Horig = 7.54554

Избыточность: Rorig = 0.0568071

2. Построим гистограмму (рис. 2) появления коэффициентов ДКП по всем блокам; просчитаем энтропию источника сообщения, а также избыточность сообщений.

Энтропия: Hdct = 0.491461

Избыточность: Rdct = 0.938567

Изображение выглядит как снимок экрана, белый, Прямоугольник

Автоматически созданное описание

Рис. 2. Гистограмма появления коэффициентов ДКП по всем блокам.

На рис. 3 представлено изображение после ДКП в каждом блоке.

Изображение выглядит как снимок экрана, пространство, черный, шаблон

Автоматически созданное описание

Рис. 3. Блоки ДКП.

3. Используя функцию makeGamma, сгенерируем кодовую таблицу и поделим на неё коэффициенты ДКП в блоках, округлив результаты. Выполним зигзаг-сканирование коэффициентов в блоках и RLE-кодирование. Подсчитаем число бит закодированного «Портрета» и оценим полученный выигрыш:

исходное изображение содержит 498432 байт;

результат зигзаг-сканирования содержит 498432 байт;

результат RLE-кодирования содержит 93944 байт;

выигрыш составляет 5.30563 раза.

6. Восстановим изображение и сравните с оригиналом (рис. 4 и рис. 5).

Изображение выглядит как снимок экрана, пространство, черный, шаблон

Автоматически созданное описание

Рис. 4. Блоки ДКП после восстановления.

Изображение выглядит как мультфильм, Зуб, статуя, улыбка

Автоматически созданное описание

Рис. 5. Изображение после восстановления.

**Выводы.**

В ходе данной лабораторной работы была разработана программа, которая способна сжимать изображение от 3 до 10 раз в зависимости от входного параметра quality. Сжатие осуществляется во многом за счёт удаления высокочастотных компонент из блоков ДКП, последствием такого метода является наличие блочных структур на восстановленном изображении (см. рис. 5).