


Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Кафедра Школа бакалавриата (школа)

Оценка работы 
Руководитель от УрФУ: Ананичев Д.С.
Отлично

Тема задания на практику

Модуль архивации JPEG файлов

ОТЧЕТ

Вид практики Производственная практика

Тип практики Преддипломная практика

Руководитель практики

Ананичев Дмитрий Сергеевич
ФИО руководителя


Подпись

Студент Кулаков Владислав Сергеевич
ФИО студента

Специальность (направление подготовки) 10.05.01 Компьютерная безопасность

Группа МЕН-651015

Екатеринбург 2020

Введение

В мире наблюдается огромный рост цифровой информации и особенно актуально создание алгоритмов её сжатия, так как ресурсы для её хранения ограничены. Существует много алгоритмов архивации данных, большинство из них работают не зависимо от типа и формата данных. Но иногда нужны алгоритмы более узкого класса. В нашем случае акцентируется внимание на алгоритм JPEG. Это основной алгоритм хранения фото информации. Сейчас фото информации очень много, почти у каждого в кармане устройство, например телефон, который умеет делать фотографии. Социальные сети, такие как Instagram, которые в первую очередь хранят фотографии нуждаются в хороших методах их архивации. Фотографы, которые даже на паспорт делают несколько снимков, чтобы человек имел возможность выбора хранят очень много фото. Много сканированных документов. И всё это говорит об актуальности архивации JPEG файлов.

Основная цель работы: представить варианты алгоритма архивации JPEG файлов.

Задачи работы:

1. Изучить процесс получения квантованных матриц в JPEG.
2. Представить варианты алгоритмов кодирования квантованных матриц.
3. Проанализировать различные алгоритмы.
4. Сформировать возможный вариант архивирования JPEG.

Алгоритм преобразования в JPEG

1. Цветовое пространство

В работе будет рассматриваться структура большинства JPEG файлов, однако сам стандарт JPEG описывает множество различных способов хранения. Так, например, можно хранить однокомпонентные изображения в градации серого, из четырёх компонент в цветовом пространстве CMYK. Можно хранить стандартную структуру RGB, однако так как JPEG является примером формата сжатия с потерями основное цветовое пространство состоит из 3 компонент YCbCr.

Допустим у нас есть сырой кадр в 3-х компонентном пространстве RGB, то есть мы имеем для каждого пикселя 3 байта данных. Далее по формуле цветовое пространство RGB преобразуем в YCbCr.

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

Где Y – определяет яркость цвета, Cb – доля синего цвета (Color blue), Cr – доля красного цвета (Color red). Этот формат был разработан для телевидения, чтобы структура цветного изображения была похожа на структуру чёрно-белого.

2. Дискретное косинусное преобразование

Далее получившееся изображение разбивают на блоки 8x8 пикселей, также разбиваем по компонентам, получаем 3 матрицы 8x8 для Y , Cb и Cr соответственно. К каждой компоненте применяется дискретное косинусное преобразование (ДКП) по формуле:

$$F(u, v) = \frac{c(u, v)}{4} * \sum_{x=0}^8 \sum_{y=0}^8 f(x, y) * \cos\left(\frac{2 * x + 1}{16} u \pi\right) * \cos\left(\frac{2 * y + 1}{16} v \pi\right)$$

$$c(u, v) = \begin{cases} \left(\frac{1}{2}\right), u = 0 \text{ и } v = 0 \\ 1, u \neq 0 \text{ или } v \neq 0 \end{cases}$$

Где u, v координаты новых матрицы 8x8, а x, y координаты имеющих. Таким преобразованием самые большие значения обычно находятся в левом верхнем углу, где находятся низкочастотные значения, а самые маленькие в правом нижнем, где соответственно высокочастотные. Например, вот такая матрица:

-415	-33	-58	35	58	-51	-15	-12
5	-34	49	18	27	1	-5	3
-46	14	80	-35	-50	19	7	-18
-53	21	34	-20	2	34	36	12
9	-2	9	-5	-32	-15	45	37
-8	15	-16	7	-8	11	4	7
19	-28	-2	-26	-2	7	-44	-21
18	25	-12	-44	35	48	-37	-3

3. Квантование

Алгоритм JPEG сжимает информацию с потерями. Так как высокочастотная информация плохо воспринимается человеческим глазом, то эти коэффициенты и являются основным пренебрежением. Для этого используют матрицы квантования 8x8, они хранятся в файле JPEG и могут быть различны для разных компонент. Например, в стандарте JPEG представлена вот такая матрица:

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

После квантования разделив каждый коэффициент наших матриц на соответствующие в матрице квантования получим интересный результат:

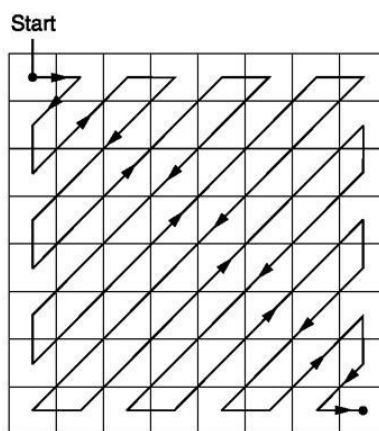
-26	-3	-6	2	2	-1	0	0
0	-3	4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Таким образом в нашей матрице потеряна высокочастотная информация и значащие коэффициенты скапливаются в левом верхнем углу.

Варианты кодирования квантованных матриц

1. Зигзаг преобразование

Все основные алгоритмы кодирования квантованных матриц начинаются с зигзаг преобразования, в том числе и в формате JPEG. Матрица преобразуется из двумерной таблицы в одномерный вектор путём обхода по диагоналям так чтобы в начале были значащие коэффициенты.



Таким образом мы получим одномерный вектор чисел в конце которого будет большинство нулей нашей матрицы.

$[-26, -3, 0, -3, -3, -6, 2, 4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, 0, \dots]$

Также для уменьшения количества нулей в последней диагонали имеющей значащие числа можно выбирать вариант обхода, например, в нашем случае если транспонировать матрицу изменив обход будет на один 0 меньше.

$[-26, 0, -3, -6, -3, -3, -4, 1, 4, 2, 2, 1, 5, 1, 1, 0, 0, 2, -1, 1, -1, 0, 0, -1, -1, 0, \dots]$

2. Обрезка нулей

Так как в нашем векторе чрезмерно много нулей в конце нужно как-то их обрезать, основных вариантов 2:

- 1) Вставка длины количества значащих элементов вектора в результирующую последовательность

25: $[-26, 0, -3, -6, -3, -3, -4, 1, 4, 2, 2, 1, 5, 1, 1, 0, 0, 2, -1, 1, -1, 0, 0, -1, -1]$

2) Вставка символа конца значащих элементов
[**-26**, 0, -3, -6, -3, -3, -4, 1, 4, 2, **2**, **1**, **5**, **1**, **1**, 0, 0, 2, -1, 1, -1, **0**, **0**, -1, -1, **END**]

Количество элементов в векторе равно 64 так как матрицы были 8x8, соответственно при декодировании их надо будет добавить. И элемент конца значащих **END** должен кодироваться меньше, чем 6 битами, потому что столько нужно для указания длины $2^6 = 64$.

3. Экспоненциальный код Голомба

Одним из самых простых способов закодировать числа в нашем векторе является экспоненциальный код Голомба. Каждое число будет состоять из трёх частей: (на примере числа -26)

- 1) Длина в битовом представлении модуля числа ($26_{10} = 11010_2$) из 1 заканчивающееся 0

$$(26_{10} = 11010_2) \Rightarrow 111110$$

- 2) Знак числа, где 0 это минус, 1 это плюс
- 3) Само число без ведущей единицы

$$(26_{10} = 11010_2) \Rightarrow 1010$$

Получим, что наши числа будут кодироваться примерно так (0 кодируется просто как 0):

$$-26 \Rightarrow 11111001010$$

$$1 \Rightarrow 101$$

$$-1 \Rightarrow 100$$

В итоге алгоритм кодирования будет выглядеть так:

- 1) Применяем к матрице зигзаг преобразование в вектор
- 2) Обрезаем с помощью указания длины значащих коэффициентов вектор
- 3) Каждое значение вектора кодируем экспоненциальным кодом Голомба

Плюсы алгоритма:

+ Простая реализация

- + Кодируются абсолютно любые данные, нет ограничений на размер чисел
- + Можно кодировать потоком без анализа самих таблиц

4. Кодирование Хаффмана

Кодирование Хаффмана анализирует частоту использования определённых символов и даёт им в соответствие соответствующие коды необходимые для их представления.

1) Возьмём последовательность символов

AAAAAADDDDDDDAAAKKKKKKKKKFFCCFFF 32 символа

2) Строим таблицу частот символов:

Символы	A	D	K	C	F
Количество	10	6	9	2	5

3) Стартовый шаг алгоритма: от частот символов строим листья дерева по правилу: берём два блока наименьшего количества, левому присваиваем 0, правому присваиваем 1, удаляем блок из просматриваемых и добавляем суммарный блок. Так пока не дойдём до корня, потом восстанавливаем коды.

Блоки	A	K	D	F — 0	C — 1
Количество	10	9	6	5	2
Блоки	A	K	D — 0	FC — 1	
Количество	10	9	6	5+2=7	
Блоки	A — 0	K — 1	DFC		
Количество	10	9	6+5+2=13		
Блоки	AK — 0	DFC — 1			
Количество	10+9 = 19	6+5+2=13			
Блоки	AKDFC				
Количество	10+9+6+5+2=32				

4) Восстанавливаем коды букв в порядке, идущем от корня:

A — 00, K — 01, D — 10, F — 110, C — 111

В нашем случае мы можем взять в качестве символов значения чисел в векторе, но нам придётся хранить таблицу Хаффмана под каждый конкретный файл. В стандарте JPEG решили что данные в матрицах (соответственно и в векторе) мы ограничим значениями от -32767 до 32767 и составим коды увеличивающиеся по мере увеличения значений:

Значения	Категория	Код
0	0	-
-1, 1	1	0, 1
-3, -2, 2, 3	2	00, 01, 10, 11
-7, -6, -5, -4, 4, 5, 6, 7	3	000, 001, 010, 011, 100, 101, 110, 111
⋮	⋮	⋮
-32767...16384, 16384,...,32767	16	...

Так каждое число кодируется в виде 4 бита на категорию и код в соответствующей категории.

5. Блоки нулей

В векторе получившемся после зигзаг преобразования и обрезки всё равно часто встречаются внутренние последовательности из нулей, например если какой-то коэффициент где-то в конце оказался достаточно большим и у нас есть числа в начале, дальше поток нулей, а потом ещё число/числа.

В формате JPEG решили, что такие последовательности тоже надо отлавливать. Так как выравнивание на байт предпочтительно, а на категорию отводится 4 бита, то ещё 4 бита отвели под кол-во нулей до этого числа, теперь у нас есть пара (количество нулей, категория):

$$(5, 3)010 \Rightarrow 5 \text{ нулей, } 3 \text{ категория, } 011 \Rightarrow [0, 0, 0, 0, 0, -4]$$

Блок (15, 0) соответствует 16 нулям. А блок (0, 0) не имеющий смысла, сделали концом значимых данных, то есть **END**.

Сами же пары, которых в общей сложности $2^8 = 256$ кодируются с помощью алгоритма Хаффмана и хранят свою таблицу в файле.

6. Алгоритм в формате JPEG

Последовательность действий в алгоритме JPEG выглядит так:

- 1) Преобразовать таблицу по зигзаг кодированию
- 2) Преобразование в пару ((нулей до, категория), значение в категории)

- 3) (нулей до, категория) + ((0, 0) = *END*) для всех векторов собираются частоты и кодируются по алгоритму Хаффмана, таблицы частот записываются, для восстановления значений
- 4) Каждый вектор записывается как код пары, значение в категории и так каждое значение до 64 элемента или до не значащей последовательности нулей которая кодируется маркером *END*.

Плюсы алгоритма:

- + Очень хорошо сжимает таблицы квантования, поэтому JPEG данные очень сложно сжать ещё больше

Минусы алгоритма:

- Требуется хранить информацию для декодирования в файле
- Нельзя декодировать на ходу, ведь нужно сгенерировать таблицы
- В таком описании ограничен размером категории, то есть чисел, однако расширяем при изменении размерности пар

Также в JPEG принято кодировать первый элемент вектора отдельной таблицей, так как он обычно больше остальных и является основным, его называют DC, остальные AC кодируются отдельной таблицей.

Алгоритм архивации JPEG файлов

На основе полученной информации можно сформировать алгоритм архивации большого количества JPEG файлов путём сжатия матриц квантования. Так как каждый отдельный файл сжимает матрицы по-своему, то он хранит свою отдельную таблицу преобразования кодов Хаффмана, однако имея большое количество файлов, мы имеем много повторений одних и тех же символов, имеющих возможно и разные коды в таблицах, однако встречающихся в разных файлах.

Минимальный вариант алгоритма:

- 1) Восстановление таблиц каждого файла, эквивалентно получению огромного JPEG файла
- 2) Преобразуем каждую в вектор по зигзаг преобразованию, в теории можно не восстанавливать до матриц
- 3) В векторе формируем пары (количество нулей, категория) значение в категории
- 4) Пары (количество нулей, категория) формируют единую таблицу Хаффмана для всех файлов, тем самым избавляясь от их повторов
- 5) Записываем всё как в JPEG алгоритме код пары, значение в категории

В алгоритме JPEG каждое цветовое пространство хранится отдельно, так как данные разных компонент цветового пространства значительно различны, так что для каждой компоненты принято хранить отдельную структуру Хаффмана. Для организации потокового преобразования, и чтобы алгоритм работал быстрее, можно сформировать статические таблицы, сформированные по большому объёму данных, тогда сами таблицы тоже не нужно будет хранить, они будут частью алгоритма.

Заключение

В работе рассмотрена структура формирования JPEG файла. Описана последовательность действий для получения квантованных матриц. Эти матрицы являются хорошим местом сжатия данных. Но обычно они воспринимаются как сжатие для конкретного файла. Однако имея большой объём файлов, одинаковая структура этих таблиц в каждом файле позволяет воспринимать набор файлов как большой файл и сжимать информацию ещё эффективнее, как будто мы положили все наши фото на большой лист и теперь мы имеем просто большое фото. Рассмотрены способы формирования алгоритмов сжатия матриц и представлены варианты этих алгоритмов. Даже используя стандартный алгоритм, мы можем получить сжатие за счёт повторяющихся таблиц Хаффмана. Так же имея алгоритм, который преобразует эти матрицы лучше, чем в JPEG получим ещё большее сжатие. Таким образом заглянув в структуру файла можно найти места для их обобщения и сформировать алгоритмы для их архивации.

Оглавление

Титульный лист	1
Введение	2
Алгоритм преобразования в JPEG	3
1. Цветовое пространство	3
2. Дискретное косинусное преобразование	3
3. Квантование	4
Варианты кодирования квантованных матриц	6
1. Зигзаг преобразование	6
2. Обрезка нулей	6
3. Экспоненциальный код Голомба	7
4. Кодирование Хаффмана	8
5. Блоки нулей	9
6. Алгоритм в формате JPEG	9
Алгоритм архивации JPEG файлов	11
Заключение	12