Мультимедиа

Компрессия

Компрессия

- При компрессии информации удаляется некоторый избыток, присутствующий в исходных данных.
- Избыточность - центральное понятие в теории сжатия информации. Любые данные с избыточной информацией можно сжать. Данные, в которых нет избыточности, сжать нельзя, точка.

Избыточность в тексте

- Компьютер способен сохранять и обрабатывать лишь двоичную информацию, состоящую из нулей и единиц => каждому символу текста необходимо сопоставить двоичный код.
- Современные компьютеры используют так называемые коды ASCII (хотя все больше компьютеров и приложений используют новые коды Unicode).

Избыточность в тексте

- ASCII представляет код фиксированной длины, где каждому символу присваивается 8-битовая последовательность (сам код занимает семь битов, а восьмой - проверочный, который изначально был задуман для повышения надежности кода).
- Код фиксированной длины представляется хорошим выбором, поскольку позволяет программам легко оперировать с символами различных текстов.
- Но код фиксированной длины является по существу избыточным.

Избыточность в тексте

- В случайном текстовом файле мы ожидаем, что каждый символ встречается приблизительно равное число раз. Однако файлы, используемые на практике, не являются случайными. Они содержат осмысленные тексты, и по опыту известно, что, например, в типичном английском тексте некоторые буквы, такие, как «E», «T» и «A», встречаются гораздо чаще, чем «Z» и «Q».
- Это объясняет, почему код ASCII является избыточным, а также указывает на пути устранения избыточности.

Избыточность в тексте

- Статистические методы компрессии используют статистические свойства сжимаемых данных и присваивают всем символам коды с переменной длиной.
- Под «статистическими свойствами» обычно понимается вероятность (в смысле частоты появления) каждого символа в потоке данных.

Избыточность в тексте

- ASCII избыточен прежде всего потому, что независимо от частоты использования присваивает каждому символу одно и то же число бит (восемь).
- Чтобы удалить такую избыточность, можно воспользоваться кодами переменной длины, в котором короткие коды присваиваются буквам, встречающимся чаще, а редко встречающимся буквам достаются более длинные коды.

Избыточность

- Степень сжатия зависит от избыточности взятого текста, а также от используемых кодов переменной длины. Текст, в котором одни символы встречаются очень часто, а другие - очень редко, имеет большую избыточность; он будет сжиматься хорошо, если коды переменной длины выбраны подходящим образом.
- Случайные данные невозможно сжать, так как в них нет избыточности

Избыточность

- Избыточность в оцифрованных изображениях (фотографии, рисунки, картинки, графики и т.п.)
- Цифровое изображение - это прямоугольная матрица окрашенных точек, называемых пикселами.
- Каждый пиксел представляется в компьютере с помощью цветового кода.

Избыточность

- Имеется два вида избыточности в цифровых изображениях.
- Первый вид похож на избыточность в текстовом файле. В каждом неслучайном изображении некоторые цвета могут преобладать, а другие встречаться редко. Такая избыточность может быть удалена с помощью кодов переменной длины, присваиваемых разным пикселам, точно также как и при сжатии текстовых файлов.

Избыточность

- Другой вид избыточности в изображении гораздо более важен, он является результатом корреляции пикселов. Когда наш взгляд перемещается по картинке, он обнаруживает в большинстве случаев, что соседние пикселы окрашены в близкие цвета. Можно сказать, что ближайшие пикселы изображения коррелируют между собой.

Избыточность

Избыточность

- Независимо от метода, которым сжимается изображение, эффективность его компрессии определяется прежде всего количеством избыточности, содержащимся в нем.
- Предельный случай - это однотонное изображение. Оно имеет максимальную избыточность, потому что соседние пикселы тождественны. Понятно, такое изображение не интересно с практической точки зрения, оно, если встречается, то крайне редко. Тем не менее, оно будет очень хорошо сжиматься любым методом компрессии.

Избыточность

- Другим экстремальным случаем является изображение с некоррелированными, то есть, случайными пикселами. В таком изображении соседние пикселы, как правило, весьма различаются по своему цвету, а избыточность этого изображения равна нулю. Его невозможно сжать никаким методом.

Избыточность

- Не существует методов и алгоритмов, способных эффективно сжимать ЛЮБЫЕ файлы, или даже существенную часть их. Для того, чтобы сжать файл, алгоритм компрессии должен сначала изучить его, найти в нем избыточность, а потом попытаться удалить ее.
- Поскольку избыточность зависит от типа данных (текст, графика, звук и т.д.), методы компрессии должны разрабатываться с учетом этого типа.
- Алгоритм будет лучше всего работать именно со своими данными. В этой области не существует универсальных методов и решений.

Основные понятия

- *Компрессор* или *кодер* - программа, которая сжимает «сырой» исходный файл и создает на выходе файл со сжатыми данными, в которых мало избыточности.
- *Декомпрессор* или *декодер* работает в обратном направлении. Отметим, что понятие кодера является весьма обидим и имеет много значений, но поскольку мы обсуждаем сжатие данных, то у нас слово кодер будет означать компрессор.
- Термин *кодек* иногда используется для объединения кодера и декодера.

Основные понятия

- Метод *неадаптивного* сжатия подразумевает неспособность алгоритма менять свои операции, параметры и настройки в зависимости от сжимаемых данных. Такой метод лучше всего сжимает однотипные данные.
- *Адаптивные* методы сначала тестируют «сырые» исходные данные, а затем подстраивают свои параметры и/или операции в соответствии с результатом проверки.

Основные понятия

- Методы компрессии могут быть также *локально адаптивными*, что означает способность алгоритма настраивать свои параметры исходя из локальных особенностей файла и менять их, перемещаясь от области к области входных данных.

Основные понятия

- Компрессия *без потерь/с потерей*: некоторые методы сжатия допускают потери. Они лучше работают, если часть информации будет опущена. Когда такой декодер восстанавливает данные, результат может не быть тождественен исходному файлу. Такие методы позволительны, когда сжимаемыми данными является графика, звук или видео. Если потери невелики, то бывает невозможно обнаружить разницу.
- Текстовые данные, например, текст компьютерной программы, могут стать совершенно непригодными, если потеряется или изменится хоть один бит информации. Такие файлы можно сжимать только методами без потери информации.

Основные понятия

- *Симметричное* сжатие - это когда и кодер, и декодер используют один и тот же базовый алгоритм, но используют его в противоположных направлениях. Такой метод имеет смысл, если постоянно сжимается и разжимается одно и то же число файлов.

Основные понятия

- При *асимметричном* методе или компрессор или декомпрессор должны проделать существенно большую работу. Таким методам тоже находится место под солнцем, они совсем не плохи. Метод, когда компрессия делается долго и тщательно с помощью сложнейшего алгоритма, а декомпрессия делается быстро и просто, вполне оправдан при работе с архивами, когда приходится часто извлекать данные. То же происходит и при создании и прослушивании аудиофайлов формата mp3. Обратный случай, это когда внешние файлы часто меняются и сохраняются в виде резервных копий. Поскольку вероятность извлечения резервных данных невелика, то декодер может работать существенно медленнее своего кодера.

Основные понятия

- *Производительность* сжатия
- Несколько величин используются для вычисления эффективности алгоритмов сжатия:
- Коэффициент сжатия;
- Фактор сжатия

Основные понятия

1. Коэффициент сжатия определяется по формуле

Коэффициент сжатия = размер выходного файла / размер входного файла

- Коэффициент 0.6 означает, что сжатые данные занимают 60% от исходного размера. Значения большие 1 говорят о том, что выходной файл больше входного (отрицательное сжатие).

Основные понятия

2) Величина, обратная коэффициенту сжатия, называется фактором сжатия:

- Фактор сжатия = размер входного файла / размер выходного файла
- В этом случае значения большие 1 означают сжатие, а меньшие 1 - расширение. Этот множитель представляется большинству людей более естественным показателем: чем больше фактор, тем лучше компрессия.

Основные понятия

- *Вероятностная модель*. Это понятие очень важно при разработке статистических методов сжатия данных. Зачастую, алгоритм компрессии состоит из двух частей, вероятностной модели и непосредственно компрессора. Перед тем как приступить к сжатию очередного объекта (бита, байта, пиксела и т.п.), активируется вероятностная модель и вычисляется вероятность этого объекта. Эта вероятность и сам объект сообщаются компрессору, который использует полученную информацию при сжатии.

Основные понятия

- *Алфавит* означает множество символов в сжимаемых данных. Алфавит может состоять из двух символов, 0 и 1, из 128 символов ASCII, из 256 байтов по 8 бит в каждом или из любых других символов.

Мультимедиа

Статистические методы для компрессии

Энтропия

- Основы теории информации были заложены Клодом Шеноном в 1948 году.
- Для наших целей сжатия данных нам необходимо освоить только одно теоретико-информационное понятие, а именно, *энтропию*.

Энтропия

- Под энтропией символа a, имеющего вероятность P, подразумевается количество информации, содержащейся в a, которая равна

  $H = -P*log_2 P.$

- Например, если вероятность P_a символа a равна 0.5, то его энтропия $-P_a \, log_2 \, P_a = 0.5$.

Энтропия

- Если символы некоторого алфавита с символами от a_i до a_n имеют вероятности от P_i до P_n, то энтропия всего алфавита равна сумме $\Sigma_n\ (—P_i\ log_2\ P_i)$.
- Если задана строка символов этого алфавита, то для нее энтропия определяется аналогично.

Энтропия

- С помощью понятия энтропии теория информации показывает, как вычислять вероятности строк символов алфавита, и **предсказывает её наилучшее сжатие**, то есть, наименьшее, в среднем, число бит, необходимое для представления этой строки символов.

Энтропия

- Продемонстрируем это на простом примере.
- Для последовательности символов ABCDE с вероятностями 0.5, 0.2, 0.1, 0.1 и 0.1, соответственно, вероятность строки «AAAAABBCDE» равна $P = 0.5^5 * 0.2^2 * 0.1^3 = 1.25 * 10^{-6}$.
- $log_2 P = —19.6096$.
- Наименьшее в среднем число требуемых бит для кодирования этой строки равно — $log_2 P$, то есть, 20.
- Кодер, достигающий этого сжатия называется *энтропийным кодером*.

Энтропия

Пример: Проанализируем энтропию алфавита, состоящего всего из двух символов a_1 и a_2 с вероятностями P_1 и P_2.

- Поскольку $P_1 + P_2 = 1$, то энтропия этого алфавита выражается числом $-P_1 \log_2 P_1 - (1 - P_1) \log_2(1 - P_1)$
- В таблице приведены различные значения величин P_1 и P_2 вместе с соответствующей энтропией.

Энтропия

P_1	P_2	Энтропия
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

Энтропия

- Когда $P_1 = P_2$, необходим по крайней мере один бит для кодирования каждого символа. Это означает, что энтропия достигла своего максимума, избыточность равна нулю, и данные невозможно сжать.
- Однако, если вероятности символов сильно отличаются, то минимальное число требуемых бит на символ снижается. Мы, скорее всего, не сможем непосредственно предъявить метод сжатия, который использует 0.08 бит на символ, но мы точно знаем, что при $P_1 = 0.99$ такой алгоритм теоретически возможен.

Коды переменной длины

- Первое правило построения кодов с переменной длиной вполне очевидно. Короткие коды следует присваивать часто встречающимся символам, а длинные - редко встречающимся. Однако есть другая проблема. Эти коды надо назначать так, чтобы их было возможно декодировать однозначно, а не двусмысленно.

Коды переменной длины

- Рассмотрим четыре символа a1, a2, a3 и a4. Если они появляются в последовательности данных с равной вероятностью (=0.25), то мы им просто присвоим четыре двухбитовых кода 00, 01, 10 и 11. Все вероятности равны, и поэтому коды переменной длины не сожмут эти данные. Для каждого символа с коротким кодом найдется символ с длинным кодом и среднее число битов на символ будет не меньше 2. Избыточность данных с равновероятными символами равна нулю, и строку таких символов невозможно сжать с помощью кодов переменной длины (или любым иным методом).

Коды переменной длины

- Предположим теперь, что эти четыре символа появляются с разными вероятностями P(a1) = 0.49, P(a2) = 0.25, P(a3)=0.25, P(a4)=0.01
- В этом случае имеется избыточность, которую можно удалить с помощью переменных кодов и сжать данные так, что потребуется меньше 2 бит на символ. На самом деле, теория информации говорит нам о том, что наименьшее число требуемых бит на символ в среднем равно 1.57, то есть, энтропии этого множества символов.

Коды переменной длины

- Закодируем символы следующим образом
- a1 1
- a2 01
- a3 010
- a4 001
- Вроде бы всё хорошо. Тот символ, который встречается чаще, закодирован меньшим числом бит. Среднее число бит на символ 1.77, что тоже неплохо.

Коды переменной длины

- Рассмотрим последовательность из 20 символов a1a3a2a1a3a3a4a2a1a1a2a2a1a1a3a1a1a2a3a1, в которой четыре символа появляются, примерно, с указанными частотами.
- Код 1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1, 37 битов (1.85 на символ в среднем).
- * эта строка весьма коротка, и для того чтобы получить результат, близкий к теоретическому, необходимо взять входной файл размером несколько тысяч символов.

- **В чём проблема?**

Коды переменной длины

- Если мы теперь попробуем декодировать эту двоичную последовательность, то немедленно обнаружим, что код совершенно не годен.
- Первый бит последовательности равен 1, поэтому первым символом может быть только a1, так как никакой другой код не начинается с 1.
- Следующий бит равен О, но коды для a2, a3 и a4 все начинаются с О, поэтому декодер должен читать следующий бит.
- Он равен 1, однако коды для a2 и a3 оба имеют в начале 01. Декодер не знает, как ему поступить. То ли декодировать строку как 1|010|01..., то есть, a1a3a2..., то ли как 1|01|001..., то есть a1a2a4... Причем заметим, что дальнейшие биты последовательности уже не помогут исправить положение. Код является двусмысленным.

Коды переменной длины

- Нужен другой код.
- a1 1
- a2 01
- a3 000
- a4 001
- Этот код обладает свойством префикса.

Коды переменной длины

- Свойство префикса можно сформулировать так: если некоторая последовательность битов выбрана в качестве кода какого-то символа, то ни один код другого символа не должен иметь в начале эту последовательность (не может быть префиксом). Раз строка «1» уже выбрана в качестве целого кода для a1, то ни один другой код не может начинаться с 1 (то есть, они все должны начинаться на 0). Раз строка «01» является кодом для a2, то другие коды не должны начинаться с 01. Вот почему коды для a3 и a4 должны начинаться с 00. Естественно, они будут «000» и «001».

Коды переменной длины

- Выбирая множество кодов переменной длины, необходимо соблюдать два принципа: (1) следует назначать более короткие коды чаще встречающимся символам, и (2) коды должны удовлетворять свойству префикса. Следуя эти принципам, можно построить короткие, однозначно декодируемые коды, но не обязательно наилучшие (то есть, самые короткие) коды.

Коды переменной длины

- В дополнение к этим принципам необходим алгоритм, который всегда порождает множество самых коротких кодов (которые в среднем имеют наименьшую длину).
- Исходными данными этого алгоритма должны быть частоты (или вероятности) символов алфавита. К счастью, такой простой алгоритм существует. Он был придуман Давидом Хаффманом и называется его именем.

Кодирование Хаффмана

- Кодирование Хаффмана является простым алгоритмом для построения кодов переменной длины, имеют их минимальную среднюю длину.
- Этот весьма популярный алгоритм служит основой многих компьютерных программ сжатия текстовой и графической информации. Некоторые из них используют непосредственно алгоритм Хаффмана, а другие берут его в качестве одной из ступеней многоуровневого процесса сжатия.

Кодирование Хаффмана

- Метод Хаффмана [Huffman] производит идеальное сжатие (то есть, сжимает данные до их энтропии), если вероятности символов точно равны отрицательным степеням числа 2. Алгоритм начинает строить кодовое дерево снизу вверх, затем скользит вниз по дереву, чтобы построить каждый индивидуальный код справа налево (от самого младшего бита к самому старшему).
- Наилучший код переменной длины можно иногда получить без этого алгоритма. (Например, арифметический кодер).

Алгоритм Хаффмана

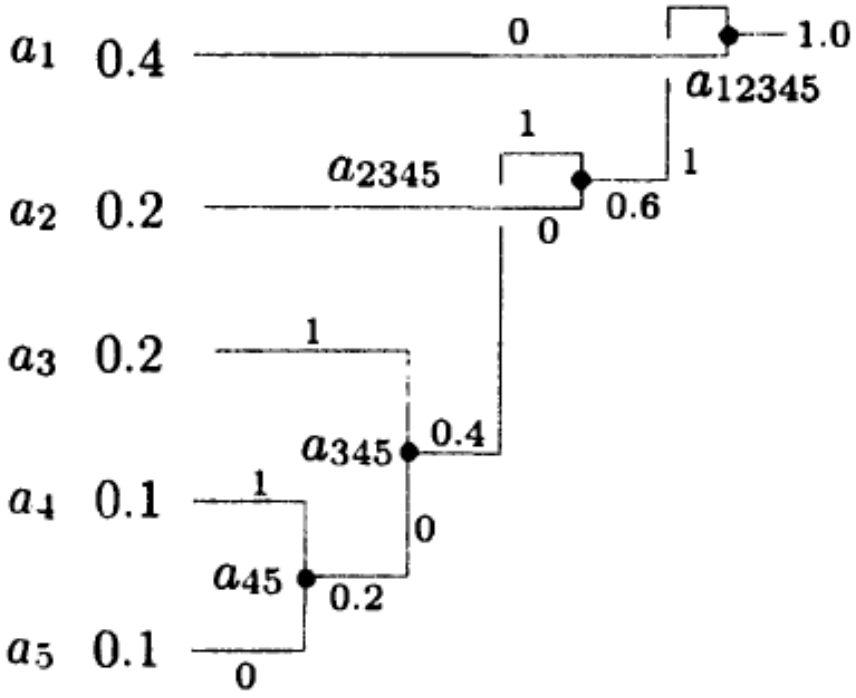
1. Составляем список символов алфавита в порядке убывания их вероятностей.
2. От корня строится дерево, листьями которого служат эти символы. Это делается по шагам – на каждом шаге выбираются два символа с наименьшими вероятностями, добавляются наверх частичного дерева, удаляются из списка и заменяются вспомогательным символом, представляющим эти два символа.
3. Вспомогательному символу приписывается вероятность, равная сумме вероятностей, выбранных на этом шаге символов.
4. Когда список сокращается до одного вспомогательного символа, представляющего весь алфавит, дерево объявляется построенным.
5. Завершается алгоритм спуском по дереву и построением кодов всех символов.

Алгоритм Хаффмана (пример)

- Рассмотрим 5 символов. Их вероятности P(a1) = 0.4, P(a2) = 0.2, P(a3)=0.2, P(a4)=0.1, P(a5)=0.1

1. a4 объединяется с a5, и оба заменяются комбинированным символом a45 с вероятностью 0.2;
2. осталось четыре символа, a1 с вероятностью 0.4, а также a2, a3 и a45 с вероятностями по 0.2. Произвольно выбираем a3 и a45, объединяем их и заменяем вспомогательным символом a345 с вероятностью 0.4;
3. теперь имеется три символа a1, a2 и a345 с вероятностями 0.4, 0.2 и 0.4, соответственно. Выбираем и объединяем символы a2 и a345 во вспомогательный символ a2345 с вероятностью 0.6;
4. наконец, объединяем два оставшихся символа a1 и a2345 и заменяем на a 12345 с вероятностью 1.

Алгоритм Хаффмана

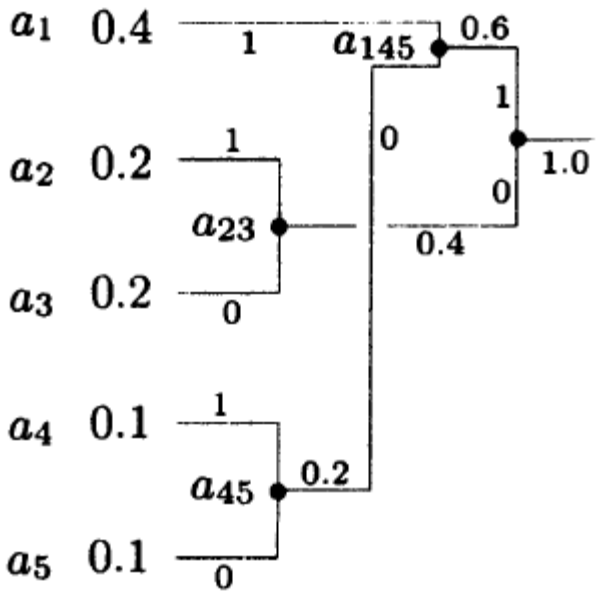


Алгоритм Хаффмана

- Дерево построено.
- Для назначения кодов мы произвольно приписываем бит 1 верхней ветке и бит 0 нижней ветке дерева для каждой пары. В результате получаем следующие коды: 0, 10, 111, 1101 и 1100. Распределение битов по краям – произвольное.

Алгоритм Хаффмана

- Кодов Хаффмана бывает много. Некоторые шаги алгоритма выбирались произвольным образом, поскольку было больше символов с минимальной вероятностью.
- (11, 01, 00, 101 и 100).



Мультимедиа

Основные принципы сжатия изображений

Сжатие изображений

- Квантовать - это, конечно, здорово, но работает это не слишком хорошо. К тому же никак не учитывается основной принцип сжатия изображений.
- Если случайно выбрать пиксел, то с большой вероятностью ближайшие к нему пикселы (neighborhood), будут иметь тот же или близкий к цвет.
- Сжатие изображений должно основываться на корреляции соседних пикселей. Такая корреляция называется пространственная избыточность.

Сжатие изображений

- Пример.
- Следующая последовательность чисел отражает интенсивность 24 смежных пикселов в одной строке некоторого непрерывно тонового изображения:
- 12,17,14,19, 21, 26, 23, 29,41,38,31,44,46, 57, 53,50,60, 58, 55, 54,52, 51,56,60.
- Здесь только два пиксела совпадают. Их среднее значение равно 40.3.
- Вычитание каждого предыдущего символа даст новую последовательность
- 12,5, -3,5, 2,5, -3, 6, 12, -3, -7,13,2,11, -4, -3,10, -2, -3, -1,-2, -1,5,4.
- Разность пикселов существенно меньше их абсолютных величин. Их среднее равно 2.58. Многие повторяются. Имеется только 15 различных значений.

Сжатие изображений

- Ещё один принцип сжатия изображений основан на том, что яркость близких пикселов тоже коррелирована. Два смежных пиксела могут иметь разные цвета. Один может быть близок к красному, а второй - к зеленому, однако, если первый был ярким, то его сосед, обычно, тоже будет ярким.
- Это свойство можно использовать для перевода представления пикселов RGB (red, green, blue) в виде трех других компонентов: яркости и двух хроматических компонентов, определяющих цвет. Одним таким форматом (или пространством цветов) служит YCbCr, где Y (компонента «светимости») отвечает за яркость пиксела, а Cb и Cr определяют его цвет.

Метрики ошибок

- Разработчикам методов сжатия изображений с частичной потерей информации необходимы стандартные метрики для измерения расхождения восстановленных изображений и исходных изображений.
- Чем ближе восстановленный образ к исходному, тем больше должна быть эта метрика (ее удобно называть «метрикой сходства»).

Метрики ошибок

- Эта метрика должна быть безразмерной и не слишком чувствительной к малым изменениям восстанавливаемого изображения.
- Общепринятой величиной, используемой для этих целей, служит *пиковое отношение сигнал/шум (PSNR) (peak signal to noise ratio).*

Метрики ошибок

- Обычно, величина PSNR варьируется в пределах от 20 до 40.
- Высокое значение PSNR означает определенную схожесть реконструированного и исходного изображений, но оно не дает гарантию того, что зрителю понравится восстановленный образ.

Метрики ошибок

- Обозначим через P_i *пикселы исходного изображения, а пикселы* восстановленного изображения пусть будут Q_i *(где* $1 < i < n$*). Дадим* сначала определение *среднеквадратической ошибке (MSE, mean* square error), которая равна

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(P_i - Q_i)^2.$$

Метрики ошибок

- Эта величина равна среднему квадратов ошибок (разностей пикселов) двух изображений. Число RMSE (корень среднеквадратической ошибки) определяется как квадратный корень числа MSE, а величина PSNR равна, по определению,

$$\mathrm{PSNR} = 20 \log_{10} \frac{\max_i |P_i|}{\mathrm{RMSE}}.$$

Метрики ошибок

- Чем больше схожесть между образами, тем меньше величина RMSE, а, значит, больше PSNR. Число PNSR безразмерно, поскольку единицами измерения и числителя, и знаменателя служат величины пикселов.
- *Использование* логарифмов сглаживает RMSE, делает эту величину менее чувствительной. Например, деление RMSE на 10 означает умножение PSNR на 2.

Метрики ошибок

- Отметим, что PSNR не имеет абсолютного значения. Бессмысленно говорить, что если PSNR равно, скажем, 25, то это хорошо. Величины PSNR используются только для сравнения производительности различных методов сжатия и для изучения влияния разных параметров на производительность того или иного алгоритма.

Интуитивные методы

- Подвыборка, возможно, является самым простым методом сжатия изображения. Простейший способ подвыборки - это просто отбросить некоторые пикселы.
- Кодер может, например, игнорировать каждую вторую строку и каждый второй столбец изображения и записывать оставшиеся пикселы в сжатый файл. Это составит 25% от исходного.
- Декодер вводит сжатые данные и использует каждый пиксел для создания четырех одинаковых пикселов реконструированного изображения. Это, конечно, приводит к потере многих деталей изображения. Такой метод редко приводит к удовлетворительным результатам.

Интуитивные методы

- Более приемлемые результаты можно получить, если записывать в сжатый файл среднее каждого блока из 4 пикселов исходного образа. При этом ни один пиксел не игнорируется, но метод по-прежнему крайне примитивен, поскольку хороший метод сжатия должен отбрасывать детали, незаметные глазу.

Интуитивные методы

- Термин «квантование» при использовании в сжатии данных означает округление вещественных чисел до целых или преобразование целых чисел в меньшие целые. Существует два вида квантования, скалярное и векторное.
- Скалярное квантование является интуитивным методом, при котором не всегда теряется только малозначимая информация.

Векторное квантование

- Изображение делится на равные блоки пикселов, которые называются *векторами,* а у кодера имеется список таких же блоков, называемый кодовой книгой.
- Каждый блок B изображения сравнивается со всеми блоками из кодовой книги и находится «ближайший» к B блок C. После чего в выходной файл записывается указатель на блок C. Если размер указателя меньше размера блока, то достигается сжатие.

Методы на основе преобразований

- Понятие преобразования широко применяется в математике. С его помощью решаются многие задачи в различных областях науки.
- Основная идея состоит в изменении математической величины (числа, вектора, функции или другого объекта) с целью придания ей другой формы, в которой она имеет, возможно, непривычный вид, но имеет полезные свойства. Преобразованная величина используется при решении задачи или при совершении некоторых вычислений, после чего к результату применяется обратное преобразование для возврата к исходной форме.

Методы на основе преобразований

- Простым иллюстративным примером могут служить арифметические операции над римскими числами. Древние римляне, по-видимому, знали, как оперировать с такими числами, однако, если требуется, скажем, перемножить два римских числа, то будет более удобно преобразовать их в современную (арабскую) форму, перемножить, а потом представить результат в виде римского числа.
- Вот простой пример:
- XCVI * XII -> 96 * 12 = 1152 -> MCLII.

Методы на основе преобразований

- Изображение можно сжать, преобразуя его пикселы (которые коррелированы) в представление, где они будут декоррелированными. Произойдет сжатие, если новые величины будут, в среднем. меньше исходных. Сжатие с потерей качества можно затем произвести с помощью квантования результата преобразования. Декодер читает сжатый файл и восстанавливает (точно или приближенно) исходные данные, применяя обратное преобразование.

Мультимедиа

JPEG

JPEG

- JPEG является очень изощренным методом сжатия изображений с потерей и без потери информации. Он применим как к цветным, так и к полутоновым изображениям.
- Этот алгоритм не очень хорошо сжимает двухуровневые черно-белые образы, но он прекрасно обрабатывает изображения с непрерывными тонами, в которых близкие пикселы обычно имеют схожие цвета.

JPEG

- Важным достоинством метода JPEG является большое количество настраиваемых параметров, которые пользователь может выбирать по своему усмотрению, в частности, он может регулировать процент теряемой информации, а, значит, и коэффициент сжатия, в широком диапазоне.

JPEG

- Обычно глаз не в состоянии заметить какого-либо ущерба даже при сжатии этим методом в 10 или 20 раз. Имеется две основные моды: с потерей (называемая также *базелиной) и без потерь информации (которая не слишком эффективна* и обычно дает фактор сжатия около 2). Большинство приложений прежде всего используют моду с потерей данных.

JPEG

- JPEG является прежде всего методом сжатия. Его нельзя рассматривать в качестве полноценного стандарта представления изображений. Поэтому в нем не задаются такие специфические параметры изображения как геометрический размер пиксела, пространство цветов или чередование битовых строк.

JPEG

- JPEG был разработан как метод сжатия непрерывно-тоновых образов. Основные цели метода JPEG состоят в следующем:

1. Высокий коэффициент сжатия, особенно для изображений, качество которых расценивается как хорошее или отличное.

2. Большое число параметров, позволяющих искушенному пользователю экспериментировать с настройками метода и добиваться необходимого баланса сжатие/качество.

JPEG

- 3. Хорошие результаты для любых типов непрерывно-тоновых изображений независимо от их разрешения, пространства цветов, размера пикселов или других свойств.
- 4. Достаточно изощренный метод сжатия, но не слишком сложный, позволяющий создавать соответствующие устройства и писать программы реализации метода для компьютеров большинства платформ.

JPEG

- 5. Несколько мод операций: (a) Последовательная мода: все цветные компоненты сжимаются в виде простого сканирования слева неправо и сверху вниз; (b) Прогрессирующая мода: изображение сжимается в виде нескольких блоков (называемых «сканами»), позволяющими делать декомпрессию и видеть сначала грубые, а потом все более четкие детали изображения; (c) Мода без потерь информации (нужная на случай, если пользователь желает сохранить пикселы без изменений; при этом приходится расплачиваться низкой степенью сжатия); и (d) Иерархическая мода, когда изображение сжимается со множеством разрешений, позволяющая создавать блоки низкого разрешения, которые можно наблюдать перед блоками высокого разрешения.

JPEG

- Сокращение JPEG произведено от Joint Photographic Experts Group (объединенная группа по фотографии). Проект JPEG был инициирован совместно комитетом CCITT и ISO (the International Standard Organization, международная организация по стандартам).
- Он начался в июне 1987 года, а первый черновой алгоритм JPEG был разработан в 1991 году. Стандарт JPEG доказал свою эффективность и стал широко применяться для сжатия изображений, особенно во всемирной паутине.

Основные шаги сжатия JPEG

1. Цветное изображение преобразуется из RGB в представление светимость/цветность.

- Этот шаг не является обязательным, но он очень важен, так как остальная часть алгоритма будет независимо работать с каждым цветным компонентом. Без преобразования пространства цветов из компонентов RGB нельзя удалить существенную часть информации, что не позволяет сделать сильное сжатие.

Основные шаги сжатия JPEG

2. Цветное изображение разбивается на крупные пикселы (этот шаг делается, если необходимо иерархическое сжатие; он всегда опускается для полутоновых изображений). Эта операция не делается для компоненты яркости.

- Укрупнение пикселов делается или в соотношении 2:1 по горизонтали и вертикали (так называемое укрупнение 2h2v или «4:1:1») или в пропорциях 2:1 по горизонтали и 1:1 по вертикали (укрупнение 2h1v или «4:2:2»).

Основные шаги сжатия JPEG

3. Пикселы каждой цветной компоненты собираются в блоки 8 x 8 , которые называются *единицами данных. Если число строк или столбцов* изображения не кратно 8, то самая нижняя строка и самый правый столбец повторяются нужное число раз.

Основные шаги сжатия JPEG

4. Затем применяется *дискретное косинус-преобразование (DCT*) к каждой единице данных, в результате чего получаются блоки 8 x 8 частот единиц данных. Они содержат среднее значение пикселов единиц данных и следующие поправки для высоких частот. Все это приготавливает данные для основного шага выбрасывания части информации.

На этом шаге происходит незначительное изменение информации из-за ограниченности точности машинных вычислений. Это означает, что даже без основного шага потери данных, происходит небольшое, крайне слабое искажение изображения.

Основные шаги сжатия JPEG

5. Каждая из 64 компонент частот единиц данных делится на специальное число, называемое *коэффициентами квантования (QC), которая* округляется до целого. Здесь информация невосполнимо теряется.

Большие коэффициенты QC вызывают большую потерю, поэтому высокочастотные компоненты, обычно, имеют большие QC. Все 64 коэффициента QC являются изменяемыми параметрами, которые, в принципе, пользователь может регулировать самостоятельно. Однако большинство приложений используют таблицу QC, рекомендуемую стандартом JPEG для компонентов светимости и цветности.

Основные шаги сжатия JPEG

6. Все 64 квантованных частотных коэффициента (теперь это целые числа) каждой единицы данных кодируются с помощью комбинации RLE и метода Хаффмана. Вместо кодирования Хаффмана может также применяться вариант арифметического кодирования, известный как кодер QM.

Основные шаги сжатия JPEG

7. На последнем шаге добавляется заголовок из использованных параметров JPEG и результат выводится в сжатый файл.

- Декодер JPEG совершает обратные действия. (Значит, JPEG является симметричным методом сжатия.)

Рекомендуемые QC

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

Светимость

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Цветность

Рекомендуемые QC

- Если квантование сделано правильно, то в блоке коэффициентов DCT останется всего несколько ненулевых коэффициентов, которые будут сконцентрированы в левом верхнем углу матрицы. Эти числа являются выходом алгоритма JPEG, но их следует еще сжать перед записью в выходной файл.

Рекомендуемые QC

- 64 числа выстраиваются одно за другим сканировании зигзагом. В начале стоят ненулевые числа, за которыми обычно следует длинный хвост из одних нулей. В файл выводятся только ненулевые числа (после надлежащего кодирования) за которыми следует специальный код EOB (end-of-block, конец блока). Нет необходимости записывать весь хвост нулей (можно также сказать, что EOB кодирует длинную серию нулей).

Кодирование

- Каждая матрица 8 x 8 квантованных коэффициентов DCT содержит коэффициент DC [в позиции (0,0) в левом верхнем углу], а также 63 коэффициента AC. Коэффициент DC равен среднему значению всех 64 пикселов исходной единицы. Наблюдения показывают, что при сжатии непрерывно-тоновых изображений, коэффициенты DC соседних единиц обычно являются коррелированными. Известно, что этот коэффициент равен сумме всех пикселов блока с некоторым общим множителем. Все это указывает на то, что коэффициенты DC близких блоков не должны сильно различаться.
- Поэтому JPEG записывает первый (закодированный) коэффициент DC, а затем кодирует разности коэффициентов DC последовательных блоков.

Кодирование

- Существуют рекомендованные коды Хаффмана для коэффициентов AC.
- (См. Д. Сэломон)

DCT

- Двумерное (матричное) DCT: Из опыта хорошо известно, что пикселы изображения имеют корреляцию по двум направлениям, а не только по одному (пикселы коррелируют со своими соседями слева, справа, а также сверху и снизу). Поэтому методы сжатия изображений используют двумерное DCT, которое задается формулой

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} p_{xy} \cos\left(\frac{(2y+1)j\pi}{2n}\right) \cos\left(\frac{(2x+1)i\pi}{2n}\right),$$

DCT

- Декодер восстанавливает сжатый блок данных (точно или приближенно), вычисляя обратное DCT (IDCT) по формуле

$$p_{xy} = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C_i C_j G_{ij} \cos\left(\frac{(2y+1)j\pi}{16}\right) \cos\left(\frac{(2x+1)i\pi}{16}\right),$$

$$\text{где } C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0, \\ 1, & f > 0. \end{cases}$$

Задание 3

- Своя реализация алгоритма сжатия изображения.
- За основу взять JPEG.
- Но не обязательно брать все его шаги.
- Оценить качество работы алгоритма по PSNR и энтропии.