Бредихин Д. Ю.

Сжатие графики без потерь качества

Содержание

1. Основы теории сжатия графической информации 3
1.1. Графическая информация как частный случай мультимедийной информации . 3
1.2. Общая схема сжатия фотореалистичных изображений..... 4
2. Кодирование изображений..... 6
2.1. Предиктивное кодирование 6
2.2. Линейное предиктивное кодирование..... 9
2.3. Нелинейные предикторы 11
2.4. Адаптивное кодирование..... 14
3. Дополнительные этапы кодирования..... 17
3.1. Контекстно-зависимая обратная связь 17
3.2. Отображение символов 19
3.3. Вычисление контекстов кодирования 20
Список литературы 21

1. Основы теории сжатия графической информации

1.1. Графическая информация как частный случай мультимедийной информации

Мультимедийной информацией (ММИ) называют, как правило, звук, двумерные картинки, видео и трехмерные образы.

Данный вид информации имеет следующие особенности:

1. Источник ММИ - приборы, измеряющие параметры физического мира. С некоторой натяжкой любую информацию, получаемую цифровым устройством из физического мира через измерительные (оцифровывающие) устройства, можно назвать ММИ (например, сила тока, скорость движения, направление ветра). Данный процесс называют аналого-цифровым преобразованием (АЦП) или "оцифровкой". Другими словами, ММИ — это дискретизированные значения показателей физических процессов.

2. Фиксированная размерность. Для хранения результата каждого измерения используется фиксированное число бит, обычно от 4-х до 32-х. Этим свойством ММИ похожа на текстовую информацию и отличается от исполнимый код для процессоров семейства x86.

3. Линейность. Измерения проводятся в большинстве случаев через равные интервалы времени или через равные расстояния в пространстве. Значения измерений всегда записываются в блок ММИ последовательно.

4. Каналы. Три первых пункта описывают блок одноканальной ММИ. Как правило, блок ММИ содержит информацию о нескольких каналах ММИ (о нескольких показателях процесса). Например, в случае стерео звука это два канала - левый и правый. В случае изображения можно считать каналом либо каждую строку, либо каждый столбец цифрового изображения. Или же вовсе не трактовать столбцы или строки как разные каналы, а просто считать, что окрестность каждого байта - двумерная, а не одномерная. Каждый квант изображения (точка, пиксель) обычно состоит из нескольких компонент: например, RGB - яркость красного (red), зеленого (green), синего (blue). Эти компоненты тоже можно считать разными

каналами, и, как правило, именно это удобнее, так как их обычно 2, 3 или 4, а не столько, сколько строк или столбцов.

Первый из этих пунктов можно считать определением ММИ, таким образом, мультимедийная информация, частным случаем которой являются фотореалистичные изображения, - это информация, получаемая цифровым устройством из физического мира через измерительные оцифровывающие устройства [2].

1.2. Общая схема сжатия фотореалистичных изображений

Рассмотрим процесс сжатия графической информации в общем виде. Как уже говорилось выше, каждый пиксель изображения обычно состоит из нескольких компонент, например, RGB. В простейшем случае мы можем просто рассматривать такое изображение как набор из трёх изображений в градациях серого, каждое из которых соответствует одной из компонент. Большинство алгоритмов работает именно таким образом, так как использование зависимостей между разными компонентами пикселя не всегда оправдано – их наличие вовсе не обязательно для ММИ, и поэтому она может отсутствовать. Таким образом, мы можем привести сжатие многокомпонентного (цветного) изображения к сжатию изображения в градациях серого, поэтому в дальнейшем под сжатием изображений мы будем понимать сжатие фотореалистичного изображения в градациях серого.

Подавляющее большинство применяемых в настоящее время алгоритмов сжатия графической информации происходит по одной общей схеме, представленной на рисунке 1 и называемой «двухшаговая схема сжатия» [8] (очень часто также используются алгоритмы с отсутствующим первым шагом, которые, тем не менее, можно также представить данной схемой). На первом шаге, называемом «анализ изображения» выбирается набор параметров, который минимизирует длину сжатого изображения. Этот набор параметров, некоторым образом, в зависимости от конкретного алгоритма, используется на втором шаге, называемом «кодирование».

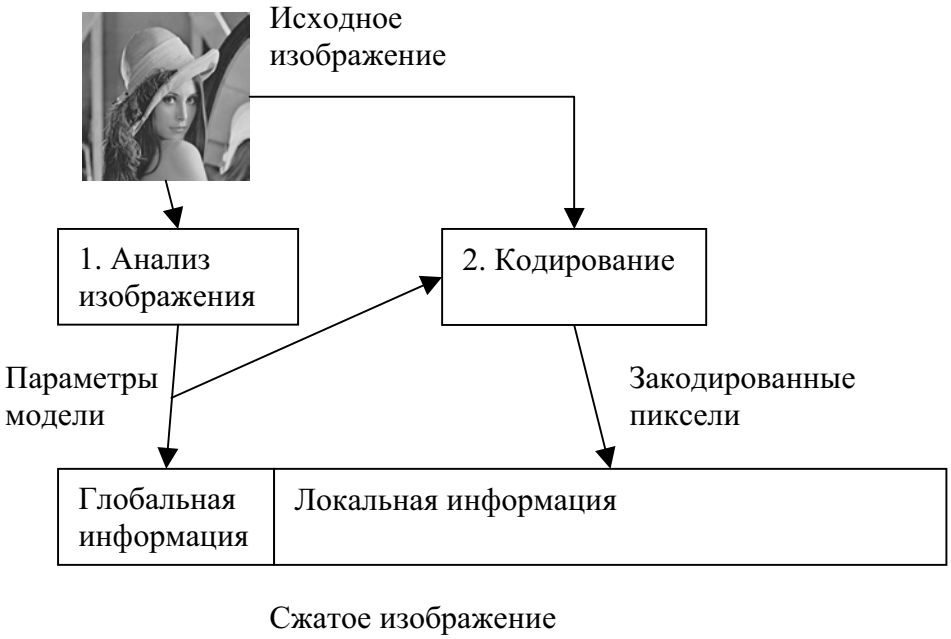

Рис. 1. Общая схема сжатия изображений

Анализ изображения специфичен для каждого из алгоритмов, и поэтому в данном пункте рассматриваться не будет. Рассмотрим более подробно второй шаг – кодирование.

2. Кодирование изображений

2.1. Предиктивное кодирование

В используемых в настоящее время алгоритмах кодирование также работает по общей схеме, называемой предиктивным кодированием (рис. 2).

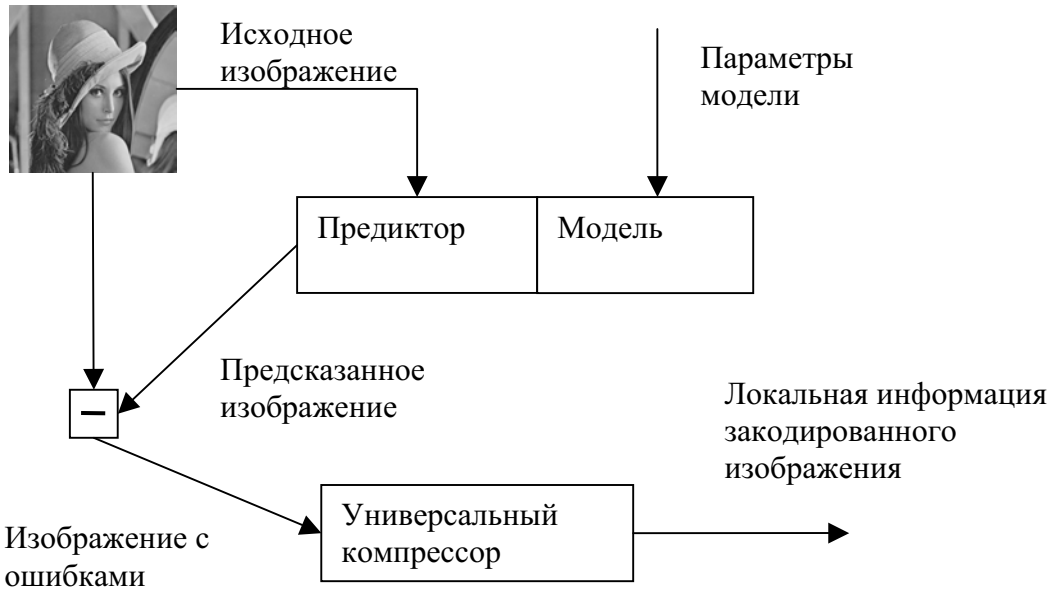

Рис. 2. Общая схема предиктивного кодирования

Предиктивное кодирование – это метод кодирования, при котором модель изображения используется для предсказания значения пикселя изображения на основании значений ряда известных пикселей (как правило, соседних).

Модель изображения в данном методе кодирования – это функция вида $V=f(x, y, \boldsymbol{n}, \boldsymbol{p})$, которая вычисляет (предсказывает) значение пикселя с координатами (x, y) в зависимости от вектора значений $\boldsymbol{n}$ соседних пикселей, значения которых уже известны, и вектора параметров изображения $\boldsymbol{p}$.

В общем случае процесс предиктивного кодирования происходит следующим образом:

1. Значения первого пикселя v_1 передаётся на вход универсального компрессора без изменений (или же сразу передаётся в локальную область закодированного изображения). Пиксель помечается как закодированный;

2. По некоторому алгоритму, называемому алгоритмом обхода плоскости, выбирается следующий кодируемый пиксель. Например, часто плоскость пикселей обходится слева-направо и сверху-вниз;

3. Используя модель изображения, вычисляется предсказанное значение текущего пикселя $p(i)=f(x_i, y_i, \boldsymbol{n}, \boldsymbol{p})$, где в вектор $\boldsymbol{n}$ включаются все пиксели, помеченные как закодированные;

4. Вычисляется ошибка предсказания $e(i)=v(i)-p(i)$,которая подаётся на вход универсального компрессора. Текущий пиксель помечается как закодированный;

5. Если изображение закодировано полностью, то процесс кодирования завершен, иначе переходим к шагу 2.

Из приведённого алгоритма видно, что описанная часть предиктивного кодирования не осуществляет непосредственно сжатие данных, а только некоторым образом изменяет их. В самом деле, непосредственным сжатием занимается последняя часть кодировщика, универсальный компрессор, который является обычным компрессором, способным сжимать любые данные. Описанный же алгоритм используется для того, чтобы повысить эффективность его работы. Это происходит следующим образом.

Минимальный ожидаемый объем данных на выходе универсального компрессора в большинстве случаев можно оценить следующей формулой [3] (для наиболее часто применяемых статистических компрессоров):

$$l = \sum P(i) \log_2 P(i), \qquad (2.1)$$

где $P(i)$ - относительная частота появления на входе значения i.

Пользуясь этой формулой, можно доказать, что длина кода будет уменьшаться при увеличении «неравенства» среди значений входного потока, т.е. увеличения частоты одних значений и уменьшения частоты других.

Рассмотрим гистограмму относительных частот появления значений для некоторого тестового изображения.

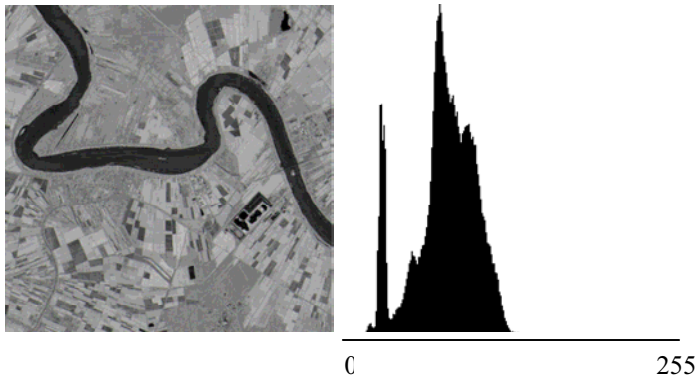

Рис. 3. Тестовое изображение и его гистограмма относительных частот

Рассмотрим теперь то же самое изображение, поданное на вход универсального компрессора после прохождения первых этапов некоторого предиктивного кодирования и аналогичную гистограмму для него.

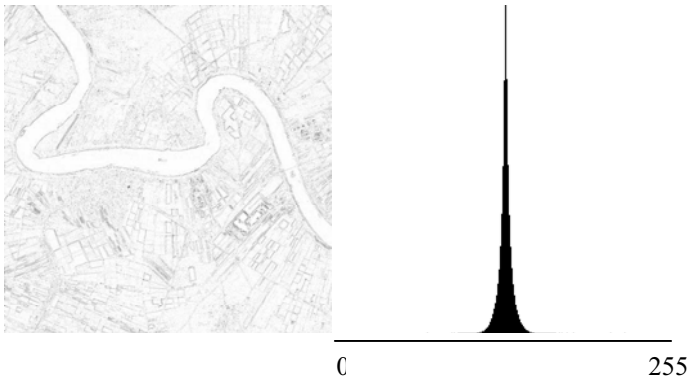

Рис. 4. Закодированное изображение и его гистограмма относительных частот

Можно заметить, что после кодирования распределение относительных частот значений пикселей стало более неравномерным, чем у исходного изображения, что, как было сказано выше, в большинстве случаев способствует уменьшению длины кода на выходе универсального компрессора.

Различные варианты алгоритмов предиктивного кодирования различаются между собой, прежде всего, видом используемого предиктора (модели), используемым универсальным компрессором, а также включением в схему дополнительных этапов, служащих для улучшения коэффициента сжатия.

Предикторы, используемые в кодировщиках, делятся по статичности модели на:

– неадаптивные – модели без параметров, использующие один и тот же предиктор для различных изображений;

– полуадаптивные – модели с параметрами, значения которых подбираются индивидуально для каждого обрабатываемого изображения, но не меняются в процессе кодирования;

– блочно-адаптивные – модели, параметры которых рассчитываются не для всего изображения целиком, а для отдельных его частей;

– адаптивные – параметры модели изменяются по ходу кодирования изображения.

Также предикторы делятся по виду модели:

– линейные, выражающие текущий пиксель через линейную комбинацию закодированных пикселей;

– нелинейные, использующие для этого выражения какую-либо нелинейную функцию;

Также можно выделить два подвида нелинейных предикторов. Это нейросетевые, использующие для предсказания обучающуюся нейронную сеть, и генетические, использующие для предсказания методы генетического программирования.

2.2. Линейное предиктивное кодирование

Линейное предиктивное кодирование (LP-кодирование, LPC) – это вид предиктивного кодирования, при котором значение текущего символа предсказывается через линейную комбинацию уже закодированных пикселей, т. е.

$$p(i) = \sum_{j=i-1}^{j=i-n} k_j v(j), \qquad (2.2)$$

где $p(i)$ - предсказываемое значение для *i*-го пикселя,

$v(j)$ - известное значение *j*-го пикселя,

k_j - некоторые коэффициенты, в общем случае не постоянные,

n - количество уже закодированных пикселей, используемых для предсказания.

Основная задача при LP-кодировании состоит в определении коэффициентов k_j, обеспечивающих как можно более точное предсказание [5].

Простейший вариант состоит в задании фиксированных коэффициентов k_j, одинаковых для каждого изображения. Полученный таким образом предиктор будет статическим.

Для расчета оптимальных значений коэффициентов предиктора можно заметить, что каждый элемент мультимедийной информации, как правило, отклоняется от значения соседних элементов по двум причинам: из-за «сильных» изменений, обусловленных характером самих данных – тренда, и «слабых» фоновых колебаний – шума [1]. Поэтому возможно два противоположных типа моделей:

- вклад шума невелик по сравнению с вкладом тренда;
- вклад тренда невелик по сравнению с вкладом шума.

В первом случае разумно предсказывать значение текущего элемента на основании сложившейся тенденции, во втором – как равное среднему арифметическому каких-либо предыдущих элементов.

Чаще всего при реализации LP-кодирования обход плоскости происходит слева-направо и сверху-вниз, поэтому в дальнейшем будем предполагать именно этот вариант обхода.

Рассмотрим наиболее известные из статических линейных предикторов. Обозначим пиксели как показано на рисунке 5.

		NN	NNE
	NW	N	NE
WW	W	p_i	

Рис. 5. Обозначение соседних пикселей

Простейшие из статических линейных предикторов предсказывают значение текущего пикселя как значение одного из соседних, либо не предсказывают совсем (значение пикселя сразу подаётся на вход универсального компрессора). Формулы для этих предикторов [7, с. 128]:

– *p(i)=0* – предсказание не используется;

– *p(i)=N* – значение текущего пикселя предсказывается равным значению верхнего пикселя;

– *p(i)=W*– значение текущего пикселя предсказывается равным значению левого пикселя;

– *p(i)=NW* – значение текущего пикселя предсказывается равным значению левого верхнего пикселя.

Все эти предикторы соответствуют шумовой модели. В современных вариантах алгоритма в чистом виде они не используются, так как обеспечивают довольно плохое предсказание. В настоящее время они используются как составные части адаптивных предикторов.

Следующая группа предикторов использует более сложные формулы, учитывающие значения нескольких соседних пикселей и трендовые или шумовые изменения между ними [7, с. 128]:

– *p(i)=N+W-NW* – считаем, что разница значений текущего пикселя и верхнего равна разнице между левым и левым верхним;

– *p(i)=N+(W-NW)/2* – считаем, что разница значений текущего пикселя и верхнего равна половине разницы между левым и левым верхним;

– *p(i)=W+(N-NW)/2* – считаем, что разница значений текущего пикселя и левого равна половине разницы между верхним и левым верхним;

– *p(i)= (N+W)/2* – значение текущего пикселя предсказывается равным среднему арифметическому значений левого и верхнего пикселя.

Первые три предиктора соответствуют трендовой модели, четвёртый – шумовой. Предикторы этой группы дают лучшее предсказание, однако, тоже чаще всего используются как составные части адаптивных предикторов.

2.3. Нелинейные предикторы

Нелинейные статические предикторы являются более сложными по сравнению с линейными, они уже используются при кодировании изображений самостоятельно, а не в составе белее сложных предикторов. Большинство из них

основано на анализе уже закодированных символов и выделении каких-либо особенностей изображения (выделении рёбер, градиентов и т.п.) в текущей области. Эти предикторы также можно задать в виде функций, однако принято более наглядное их описание – в виде алгоритма расчета p_i на псевдоязыке.

Предиктор Paeth. Разработка Алана В. Паефа [9]. Описывается следующим алгоритмом:

IF $|N-NW| \le |W-NW|$ AND $|N-NW| \le |N+W-2NW|$

$p(i) = W$

ELSE IF $|W-NW| \le |N+W-2NW|$

$p(i) = N$

ELSE

$p(i) = NW$

Предиктор DARK. Предложен фирмой Kodak, адаптируется к горизонтальным и вертикальным рёбрам [7, с. 130]. Алгоритм предиктора:

$\Delta_v = |W-NW|$ *//величина вертикального градиента*

$\Delta_h = |N-NW|$ *//величина горизонтального градиента*

// коэффициент, характеризующий преобладание вертикального градиента

// над горизонтальным

$$\alpha = \frac{\Delta_v}{\Delta_h + \Delta_v}$$

//чем больше данный коэффициент, тем с большим весом мы берем

//для предсказания левый пиксель и с меньшим - верхний

$p(i) = \alpha W + (1-\alpha)N$

Предиктор MED (median edge detection, определение середины ребра). Предложен HP Labs [6, с. 4]. Предиктор работает по следующему алгоритму:

IF $NW \ge \max(N,W)$ THEN

$p(i) = \min(N,W)$

ELSE IF $NW \le \min(N,W)$

$p(i) = \max(N,W)$

ELSE

$p(i) = N + W - NW$

Принцип работы предиктора состоит в адаптации к наличию локальных горизонтальных или вертикальных рёбер. Пиксель *N* используется для предсказания в том случае, если обнаружено вертикальное ребро. Пиксель *W* – если обнаружено горизонтальное ребро. И, наконец, если ребро не было обнаружено, то используется один из статических линейных предикторов.

Предиктор GAP (gradient-adjusted predictor, предиктор с настройкой на градиент). Предложен в алгоритме CALIC [7, с. 129]. Как видно из названия, этот предиктор адаптирует предсказанное значение в соответствии с локальным градиентом. Используется следующий алгоритм:

$d_h = |W - WW| + |N - NW| + |N - NE|$ *//величина горизонтального градиента*

$d_v = |W - NW| + |N - NN| + |NE - NNE|$ *//величина вертикального градиента*

IF (d_v-d_h>80) *//четкое горизонтальное ребро*

$p(i) = W$

ELSE IF (d_v-d_h<-80) *//четкое вертикальное ребро*

$p(i) = N$

ELSE {

$p(i) = (N + W)/2 + (NE - NW)/4$;

IF (d_v-d_h>32) *//горизонтальное ребро*

$p(i) = (p(i) + W)/2$

ELSE IF (d_v-d_h>8) *//слабовыраженное горизонтальное ребро*

$p(i) = (3p(i) + W)/4$

ELSE IF (d_v-d_h<-32) *//вертикальное ребро*

$p(i) = (p(i) + N)/2$

ELSE IF (d_v-d_h<-8) *//слабовыраженное вертикальное ребро*

$p(i) = (3p(i) + N)/4$

}

Данный алгоритм работает по следующему принципу. Если величина вертикального градиента больше горизонтального на некоторое пороговое значение, в данном случае 80, то мы считаем, что в рассматриваемом участке изображения находится четко выраженное горизонтальное ребро, и поэтому предсказываем значение текущего пикселя равным значению левого пикселя. Аналогичным

образом, если величина горизонтального градиента больше вертикального на 80, то мы предсказываем значение текущего пикселя равным значению верхнего пикселя.

В противном случае мы предсказываем значение текущего пикселя по формуле $p(i) = (N + W)/2 + (NE - NW)/4$ (обычный линейный предиктор), а затем, если разница градиентов достигла какого-либо другого порогового значения (32 или 8 в данном случае), то за предсказанное значение мы принимаем среднее взвешенное уже найденного значения и соответствующего пикселя с разными весами.

2.4. Адаптивное кодирование

Все описанные выше предикторы относились к классу статических, они применяли одни и те же преобразования над соседними пикселями для предсказания значения текущего, то есть коэффициенты и структура модели не изменялись в процессе кодирования. В связи с этим они не позволяли учесть особенности локальных областей изображения. Рассмотри теперь алгоритмы с адаптивными предикторами, параметры которых изменяются в зависимости от этих особенностей.

Алгоритмы, принадлежащие к этой группе можно разбить на два больших класса. К первому классу можно отнести предикторы, комбинирующие результаты предсказания статических предикторов с адаптивно изменяющимися весами, т.е. действующие по формуле

$$p(i) = \sum_{j=1}^{N} \alpha_j p_j(i), \qquad (2.3)$$

где $p(i)$ – значение текущего пикселя, предсказанное адаптивным предиктором,

$p_j(i)$ – значение текущего пикселя, предсказанное j-ым статическим предиктором,

α_j - некоторые переменные коэффициенты.

Основным методом для определения коэффициентов α_j является так называемый метод «штрафования» предикторов [4]. Суть метода состоит в

следующем. Пусть мы пытаемся предсказать *i*-ый пиксель. Тогда для каждого предиктора мы рассчитываем штрафной коэффициент:

$$G_j = \sum_{k=i-1}^{i-L} \left| v(k) - p_j(k) \right|, \tag{2.4}$$

где *L* – учитываемое количество уже просмотренных пикселей,

v(k) – значение *k*-го пикселя,

$p_j(k)$ - значение *k*-го пикселя, предсказанное *j*-ым предиктором.

После этого значения коэффициентов α_j рассчитываются по формуле

$$\alpha_j = \frac{1/G_j}{\sum_{i=1}^{N} 1/G_i}. \tag{2.5}$$

Ко второму классу относятся предикторы, в которых для предсказания используется формула обычного LP-кодирования

$$p(i) = \sum_{j=1}^{j=n} k_j v(i-j), \tag{2.6}$$

но, в отличие от него, динамически рассчитываются коэффициенты k_j.

Для динамического определения значений этих коэффициентов, уменьшающих ошибку предсказания, необходимо решить оптимизационную задачу. Наиболее популярный метод решения этой задачи – метод наименьших квадратов.

Суть этого метода состоит в том, что наилучшими значениями коэффициентов k_j мы считаем те, которые минимизируют сумму квадратов ошибок предсказания для *m* предыдущих пикселей. Величина *E,* которую необходимо минимизировать, рассчитывается по формуле:

$$E = \sum_{j=i-1}^{i-m} e^2(j) = \sum_{j=i-1}^{i-m} (p(j) - v(j))^2 = \sum_{j=i-1}^{i-m} (\sum_{t=1}^{n} k_t v(j-t) - v(j))^2 . \tag{2.7}$$

Для того чтобы эта величина была минимальной, необходимо, чтобы выполнялись следующие условия:

$$\frac{\partial E}{\partial k_t} = 0, t = 1..n . \tag{2.8}$$

Подставляя значение *E* в это условие, получим систему из *n* линейных уравнений с *n* неизвестными, решая которую мы и получим искомые значения коэффициентов k_t для каждого пикселя изображения.

Решение данной системы будет следующим:

$$K = Y \cdot X^{-1},$$

где $K = [k_t]$ - вектор искомых коэффициентов,

$X = \begin{pmatrix} v(i-2) & ... & v(i-n-1) \\ ... & ... & ... \\ v(i-m-1) & ... & v(i-m-n) \end{pmatrix}$ - матрица известных значений

$Y = [v(j)]$ - вектор предсказываемых значений.

Полученный по данной формуле вектор коэффициентов *K* будет содержать оптимальные по методу наименьших квадратов коэффициенты для данного пикселя.

3. Дополнительные этапы кодирования

Выше уже было сказано, что в различные варианты алгоритма предиктивного кодирования могут включаться дополнительные этапы, служащие для улучшения коэффициента сжатия.

Наиболее часто для этого используются такие методы, как контекстно-зависимая обратная связь, отображение символов и вычисление контекстов кодирования. Рассмотрим их более подробно.

3.1. Контекстно-зависимая обратная связь

Локальные градиенты, исходя из которых предсказывается значение текущего пикселя, не могут адекватно характеризовать некоторые сложные зависимости между предсказываемым пикселем и соседними пикселями. Такие сложные зависимости, как, например, шаблоны текстур, можно использовать, если после вычисления ошибки предиктора запомнить ее вместе с контекстом текущего пикселя и учитывать это значение в дальнейшем.

Впервые эффективное решение этой задачи было предложено в алгоритме CALIC и заключалось в следующем [7, с. 130-131]. Для каждого контекста C мы храним среднее значение всех ошибок предиктора $\bar{e}(C)$, и затем при получении очередного предсказанного значения *p(i)* для пикселя в контексте C мы прибавляем к нему это значение.

Этот алгоритм работает благодаря тому, что, благодаря несовершенству предиктора, в общем случае среднее значение ошибок предиктора $\bar{e}(C)$ на данном контексте C не равно нулю. Это утверждение не противоречит тому, что функция распределения ошибок предиктора без учёта контекста представляет собой распределение Лапласа с нулевым математическим ожиданием, так как на самом деле это распределение представляет собой сумму распределений ошибок предиктора на каждом контексте, которые, в свою очередь, обычно являются распределениями Лапласа с ненулевым математическим ожиданием. Этот факт можно увидеть на рисунке 6.

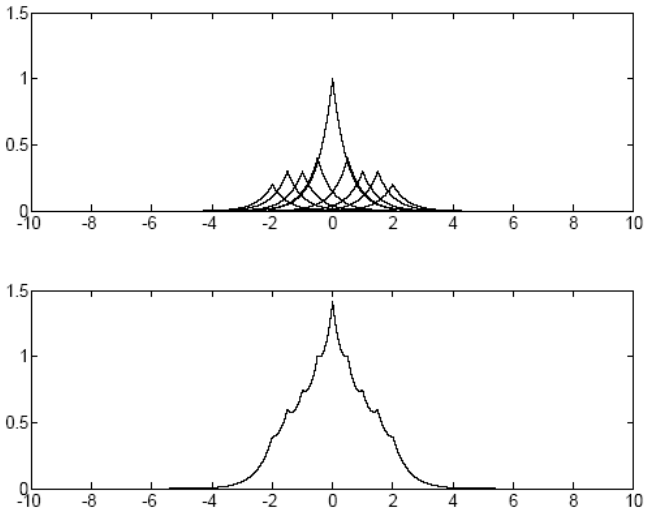

Рис. 6. Распределения ошибок предиктора с учётом контекста (сверху) и без учёта контекста (снизу)

Остаётся открытым вопрос о том, что же считать контекстом для данного пикселя. Если принимать за контекст значения нескольких соседних пикселей, то появится слишком большое число контекстов, причём большинство из них будет встречаться очень редко. Этот эффект известен под названием «проблема редких контекстов». В различных алгоритмах она решается различными способами. Для примера рассмотрим выбор контекстов в алгоритме CALIC.

В этом алгоритме контекст состоит из двух частей: текстурного контекста и контекста ошибок. Текстурный контекст формируется следующим образом. Строится вектор *C* следующего вида:

$$C(x_0,...,x_6,x_7) = (N,W,NW,NE,NN,WW,2N-NN,2W-WW).$$

Затем формируется двоичное число $B = \overline{b_7 b_6 ... b_0}$ по следующему правилу:

$$b_k = \begin{cases} 0, если\ x_k \geq p(i) \\ 1, если\ x_k < p(i) \end{cases}.$$

Это число B и называется текстурным контекстом. Очевидно, что оно представляет собой некоторый локальный «шаблон текстуры».

Контекста ошибок вычисляется, используя так называемое значение энергии ошибки, равное

$$\Delta = d_h + d_v + 2\left|p(i-1) - v(i-1)\right|, \qquad (3.1)$$

где $d_h = \left|W - WW\right| + \left|N - NW\right| + \left|N - NE\right|$,

$d_v = |W - NW| + |N - NN| + |NE - NNE|$.

Затем это значение квантуется до четырёх уровней, образуя контекст ошибок Q.

Общий контекст образуется комбинированием текстурного контекста и контекста ошибок. На первый взгляд кажется, что число контекстов, полученных таким образом, будет равно $4 \times 2^8 = 1024$. Однако, не все 2^8 значений текстурного контекста *B* возможны, а только 144. Таким образом, общее число контекстов в этом алгоритме получается равно $4 \times 144 = 576$.

3.2. Отображение символов

Отображение символов – это отображение ошибок предиктора из большого исходного набора значений (обычно от -255 до 255) в новый небольшой. В большинстве случаев такое преобразование позволяет улучшить степень сжатия благодаря тому, что значения ошибок распределены неравномерно [4, с. 15-16].

Так как очевидно, что такое преобразование не может быть обратимо, то ошибка предиктора *e(i)* преобразуется в три части: значение ошибки в новом алфавите $e_q(i)$ (квантованная ошибка), ошибка квантования *q(i)* и знаковый бит *s(i)*.

Преобразование задаётся схемой квантования, которая представляет собой набор интервалов $n_0, n_1, ..., n_k$, где *k*-размер нового алфавита. Само преобразование осуществляется по следующему алгоритму:

IF $n_k \le |e(i)| < n_{k+1}$

$e_q(i) = k$

$q(i) = |e(i)| - n_k$

$s(i) = sign(e(i))$

После окончания преобразования все три полученные части кодируются отдельно.

3.3. Вычисление контекстов кодирования

Вычисление контекстов кодирования заключается в том, что после прохождения всех основных и дополнительных этапов кодирования полученные данные подаются на вход универсального компрессора не одним общим потоком, а несколькими отдельными, с одинаковыми значениями контекста в каждом [7, с. 132-133].

Эксперимент показывает, что в данном случае предпочтительнее использовать меньшее число контекстов, чем в контекстно-зависимой обратной связи. Так, например, в алгоритме CALIC в качестве контекста кодирования используется описанный выше контекст ошибок, без текстурного контекста, но с квантованием до восьми, а не до четырёх, уровней.

Список литературы

1. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. – 384 с. Страница книги: www.compression.ru/book/
2. Ратушняк А. Сжатие мультимедийной информации. http://www.compression.ru/artest/mmi.txt
3. Bell T., Witten I., Clearly J. Modeling For Text Compression // ACM Computing Surveys, Vol.21, No.4, pp.557-591, Dec. 1989. www.compression.ru/download/articles/rev_univ/bell_1989_modeling/bell_1989_modeling_part1.html
4. Guang Deng, Hua Ye. Lossless image compression using adaptive predictor combination, symbol mapping and context filtering // Department of Electronic Engineering, La Trobe University, Bundoora.
5. Kenneth M. Dawson-Howe. Lossless Image Compression using a Simple Prediction Method // Department of Computer Science, Trinity College. www.compression.ru/download/articles/i_glless/dawson-howe_1996_spm_pdf.rar
6. Marcelo J. Weinberger, Gadiel Seroussi, Guillermo Sapiro. LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm // Hewlett-Packard Laboratories, Palo Alto, CA 94304. www.compression.ru/download/articles/loco/weinberger_1996dcc_loco_pdf.rar
7. Memon N., Wu X. Recent Developments in Context-Based Predictive Techniques for Lossless Image Compression // The Computer Journal, Vol. 40, No. 2/3, 1997. www.compression.ru/download/articles/i_lless/memon_xu_1997cj_context_pdf.rar
8. Meyer B., Tischer P. TMW – a New Method for Lossless Image Compression // Department of Computer Science, Monash University Clayton, Victoria. www.compression.ru/download/articles/i_glless/meyer_tischer_1997_tmw.pdf
9. PNG (Portable Network Graphics) Specification, Version 1.0 // W3C Recommendation