Compressed sensing
Эффективный программный формат сжатых изображений

Кокряшки Максим
max.kokryashkin@gmail.com

14 Ноября 2021

Аннотация

Существующие форматы для хранения изображений обычно реализуют одну из двух крайностей: либо в файле хранится максимально возможное количество информации (BMP, RAW, NEF), либо обеспечивается сильное сжатие со значительной потерей качества изображения (jpg, jpeg, png, web-ready png, gif). Эта статья описывает формат сжатого изображения, который обеспичивает значительную степень сжатия, сохраняя при этом приемлемое качество изображения.

1 Мотивация

В чем проблема JPG и подобных ему форматов

Все изображение делится на квадраты 8x8 а затем каждый квадрат представляется в виде суммы из базисных векторов дискретного косинусного преобразования (DCT). Выбор именно такого базиса обусловлен универсальностью, но не обязательно наиболее выгоден для того или иного изображения. Кроме того, с увеличением степени сжатия изображения быстро теряют детализацию. Базсиные вектора для JPEG изображены на рисунке **??**.

Ниже можно увидеть сравнительную таблицу для двух существующих форматов, один из которых сохраянет максимум информации, второй использует сжатие. Тестируемуе изображения приведены на рисунке 2.

Формат	Объем файла (байт)	Размер относительно оригинала
BMP	3275658	100%
JPEG	36074	1%

Как будет описано далее, улучшение выбора базиса нецелесообразно, поэтому предлагаемый алгоритм будет производить основное сжатие посредством PCA, чтобы избежать потерю детализации, после чего будет производиться сжатие с использовнием DCT.

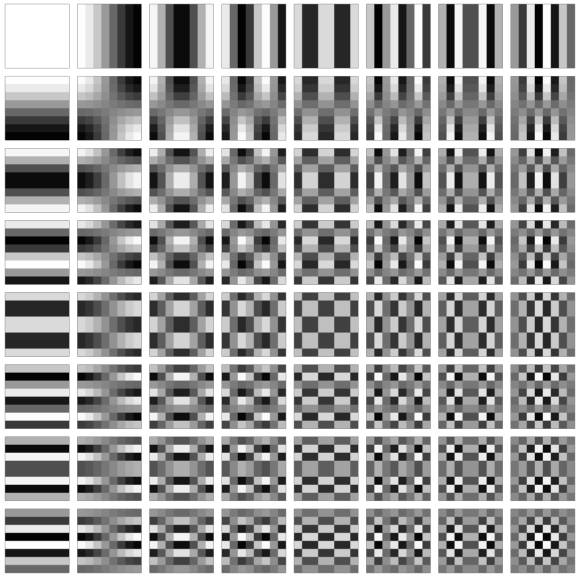


Рис. 1: Базисные вектора для JPEG

(a) Оригинал

(b) Сжатое

Рис. 2: Сравнение сжатого и оригинального изображений

2 Методы

Задача восстановления оригинального сигнала x по сжатому m описывается как

$$m = Fx + \eta, \tag{1}$$

где F — матрица сжатия.

Важной деталью при сжатии данных является «разреженность» сигнала. Сигнал x считается k-разреженным, если в векторе x есть не более k ненулевых элементов. Сам по себе сигнал может быть и не разреженным, но его можно привести к разреженному правильной заменой базиса. Восстанавливать разреженный сигнал намного проще, так как размер базиса, который необходимо хранить становится значительно меньше.

Отметим, что модель разреженности не является линейной, потому что базис, приводящий сигнал к $k-$ будет меняться от сигнала к сигналу. Сжимаемость сигнала оценивается минимальной потерей данных, которой можно достичь, приближая его неким элементом из множества k-разреженных векторов.

Аналогичный способ сжатия — использование малоранговых матриц. Общая идея здесь сохраняется, равно как и метрики качества сжатия.

Для оценки близости сигналов x_1 и x_2 классически используется свойство RIP (Restricted

Isometry Property, Свойство Ограниченной Изометрии), которое выглядит следующим образом:

$$(1-\delta)\|x_1-x_2\|_2^2 \le \|F(x_1-x_2)\|_2^2 \le (1+\delta)\|x_1-x_2\|_2^2.$$

В данном свойстве,под δ понимается некая малая константа, $\delta \in (0,1)$.

Это свойство гарантирует минимизацию ошибки, выражаемой как $\hat{x} = \underset{x}{\operatorname{argmin}}\|m-Fx\|_2^2$. Более того, если матрица F удовлетворяет условию RIP, то это означает, что большая часть алгоритмов Comressed Sensing смогут восстановить исходный сигнал используя ее.

С помощью RIP легко оценить размерность матрицы сжатия F. Если матрица имеет разменость $m \times n$ и удовлетворяет RIP порядка $2k$ (то есть свойство выполнено для всех k-разреженных векторов) с $\delta \in (0, \frac{1}{2}]$, то

$$m \ge Ck \log\left(\frac{n}{k}\right), \tag{2}$$

где $C = \frac{1}{2}\log(\sqrt{24}+1)$

Теперь, имея необходимое представление о теории, лежащей под сжатием, обсудим практическую сторону. Наиболее сложным вопросом является построение матрицы сжатия F, удовлетворяющей вышеописанным свойствам. Существует достаточно много детерминированных и недетерминированных способов построения матриц сжатия. Детерминированные матрицы дают больший коэффициент сжатия, но их построение может быть крайне сложной с точки зрения ресурсов вычислительной задачей. Недетерминированные матрицы строить проще, но они дают коэффициент сжатия чуть ниже, тем не менее, его можно привести к значениям сопоставимым с детерминированными.

С практической же точки зрения, детерминированные матрицы дают однозначность, что всегда хорошо для программистов, но их использование для создания формата сжатия изображений полностью нивелирует всю изначальную идею. С увеличением размера картинки будет расти размер матрицы сжатия, при этом вычисления будут занимать экспоненциально больше времени. В связи с этим использование детерминированных матриц для нашей задачи нецелесообразно. Недетерминированные матрицы строить можно намного быстрее, но они все еще занимают много пространства на диске.

Рассмотрим это на примере: пусть дано RGBA-изображение в разрешении 1920×1080, оно представляется в виде вектора размерности $n = 1920 \cdot 1080 \cdot 4 = 8294400$ и занимает оно, соответственно 7.91 MiB. Допустим, что мы хотим сжать изображение вдвое, тогда минимальный размер матрицы сжатия в байтах можно оценить по формуле 2 как:

$$n \cdot m \ge n \cdot Ck \log\left(\frac{n}{k}\right) = 8294400 \cdot 4147200 \cdot \log 2 \cdot C \approx 23843248021983 \cdot C \approx 21148960995499.$$

Это соответствует 1.92 TiB. То есть для сжатия изображения не самого высокого разрешения необходимо хранить матрицу сжатия объемом в почти два терабайта.

Попробуем иной подход: будем разбивать изображение на чанки размером 8×8, а матрицы строить уже для них. Вектор для такого чанка будет иметь размерность $n = 8 \cdot 8 \cdot 4 = 256$. Тогда вновь, нижняя оценка на размер матрицы сжатия в байтах получается:

$$n \cdot m \ge n \cdot Ck \log\left(\frac{n}{k}\right) = 256 \cdot 128 \cdot \log 2 \cdot C \approx 22714 \cdot C \approx 20156.$$

Изображение из примера будет состоять из 32400 чанков, то есть суммарный размер матриц будет равен 622 MiB. Это уже значительно меньше, но все еще превышает исходный рзамер изображения. Именно по этой причине форматы похожие на JPEG использует дискретное косинусное преобразование для всех изображений — его не надо хранить в самом

файле и даже вычислять матрицу нет необходимости, в силу некоторых существующих методологий.

Подведем итог. Использование индивидуальных матриц сжатия и индивидуальных базисов для сжатия сложного сигнала нецелесообразно из-за очень большого размера результирующего пакета данных и вычислительной сложности. В связи с этим надо использовать подход со стандартизированным базисом.

Предлагается воспользоваться следующим подходом: использовать дискретное косинусное преобразование, предварительно обработав изображение посредством PCA.

Для дискретного косинусного преобразовния все еще необходима матрица, но ее уже можно не пересылать, так как она стандартная. Проблемой все еще остается то, что вычисление матрицы для большого изображения все еще потребует много оперативной памяти. Решить эту проблему можно подсчетом матриц для отдельных чанков изображения, тогда они уместятся в оперативную память.

В случае с DCT, матрица сжатия для изображения X может быть вычислена следующим образом:

$$D_i = \text{idct}(I_i),$$

где idct — обратное DCT, а I_i — единичная матрица размера i. Тогда двумерное преобразование idct_2 можно задать следующим образом:

$$\text{idct2}(X) = \text{idct}(\text{idct}(X^T)^T) = D_m(D_n X^T)^T = D_m X D_n^T,$$

то есть здесь мы применяем $IDCT$ сначала к столбцам, а затем к строкам матрицы изображения. В итоге, приняв за $\oplus$ произведение Кронекера, получаем представление для матрицы сжатия, которая произведет DCT:

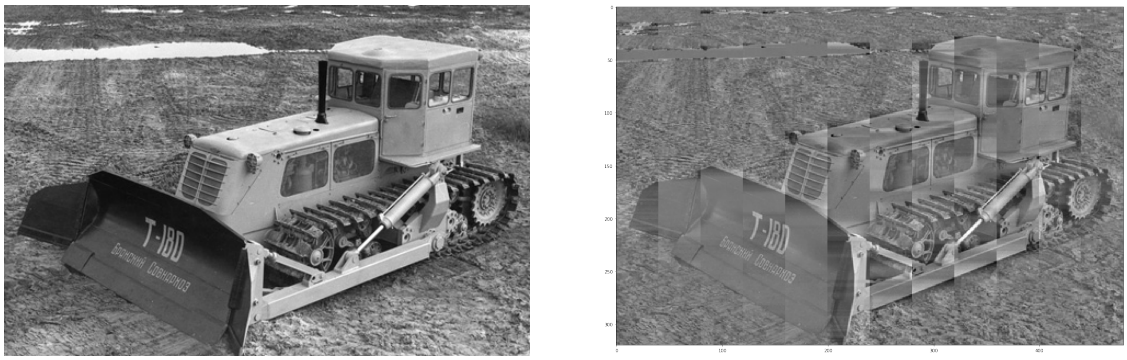
$$F = D_n \oplus D_m.$$

После этого мы имеем уже понятную нам выпуклую задачу вида (1), которая хорошо решается программными методами.

Теперь рассмотрим предобработку посредством PCA. Этот метод работает следующим образом: данные центрируются, затем производится сингулярное разложение входной матрицы X, то есть она приобретает вид $X = U\Sigma V^*$. Отсюда напрямую получаем собственные векторы и собственные значения. Из этих векторов оставляем n с самыми большими собственными значениями и проецируем исходное пространство на порождаемое ими подпространство.

В контексте предложенной задачи PCA рассматривается как один из способов упростить данные перед применением DCT. Снова возникает проблема с вычислением PCA на всем изображении в силу необходимости большого количества памяти для этих вычислений, растущего экспоненциально относительно размера входного изображения. При вычислениях на чанках проблема нивелируется, тем не менее, визуальный результат далек от воспринимаего человеком как «близкого к оригиналу». Пример почанкового PCA можно увидеть на рисунке 3.

Из этих результатов можно заключить, что хоть PCA и крайне эффективен для уменьшения размерности входных данных без значительных потерь информации, он также крайне неэффективен в контексте сжатия изображений, если мы хотим добиться визуальной схожести результата с оригиналом. Основной проблемой становится отчетливое визуально разделение чанков, на которых производилось преобразование. Кроме этого можно наблюдать серьезные проблемы с цветопередачей у сжатого изображения: в значительной степени заметна потеря цветовой насыщенности.



(a) Оригинал (b) PCA

Рис. 3: Сравнение per-chunk PCA и оригинального изображений

3 Результаты

В итоге был применен подход со сжатием исходного изображения посредством применения двумерного DCT к каждому из цветовых слоев. Этот метод позволяет производить эффективное сжатие с приемлемой скоростью вычисления, сохраняя при этом исходные цвета и визуальную схожесть. Артефактов после восстановления при это меньше, и они выражены не так сильно, как у JPEG-семейства.

Изображение на котором были проведены тесты и его же после сжатия в два раза и декомпрессии можно увидеть на рисунке 4.

(a) Оригинал

(b) Сжатое и восстановленное

Рис. 4: Сравнение сжатого и восстановленного изображения с исходным

В сжатом виде изображение имеет вид как на рисунке **??**.

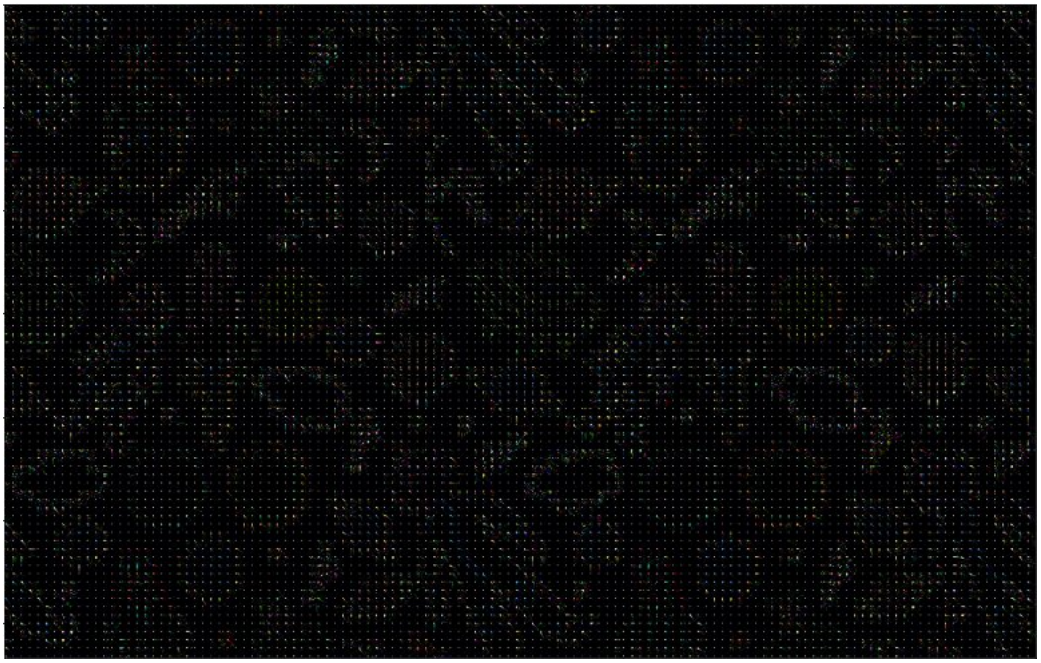


Рис. 5: Сжатый вид

Как можно видеть, при уменьшении занимаемого дискового пространства в два раза изображение восстанавливается близко к исходному практически без визуальных артифактов.

4 Ссылки

[1]: Google Colab notebook с программной имплементацией предложенного метода.

Список литературы

[1]: Deep Compressed Sensing
[2]: The Nyquist–Shannon Theorem: Understanding Sampled Systems
[3]: Introduction to compressed sensing
[4]: A Step-by-Step Explanation of Principal Component Analysis (PCA)
[5]: Image Compression Using Principle Component Analysis
[6]: PCA Hometask from DLSchool
[7]: DCT in Python