Сжатие растровых изображений нейронными сетями Цао Ена

Куликов Александр Иванович
Михальченко Николай Владимирович
ИВТ СО РАН
Новосибирск, Россия.

Аннотация

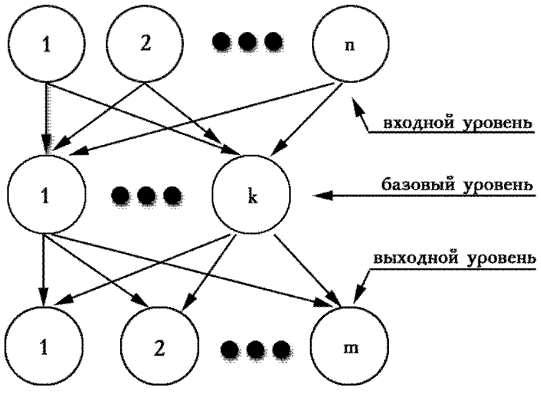
В докладе рассматриваются алгоритмы разработанные для сжатия растровых полноцветных изображений нейронными сетями специального вида - Цао Ена. Это асимметричный процесс с потерей качества. Процедура сжатия состоит из трех этапов:

- *обучение сети,*
- *подготовка для хранения,*
- *дополнительная упаковка с помощью алгоритма Хаффмана.*

Приводиться подробное описание алгоритмов разработанных для сжатия и восстановления изображений. Даются сравнительные характеристики сжатия этим алгоритмом и алгоритмом JPEG на разных классах изображений. Реализованное на основе разработанного алгоритма программное средство может использоваться не только для непосредственного сжатия изображений, но и для построения и анализа различных модификаций алгоритма

1. ВВЕДЕНИЕ

Сети Цао Ена разрабатывались для построения гладких замкнутых поверхностей в n-мерном пространстве по каркасу ключевых точек[1]. При помощи сети Цао Ена можно делать сжатие растровых изображений с потерей качества за счет выбора базовых точек, по которым восстанавливается изображение с помощью обученной нейронной сети. Для сжатия строиться сеть преобразующая координаты изображения в цвет(RGB).



Для использования сети Цао Ена, координаты изображения преобразуются в координаты на гиперсфере. Наиболее оптимально, преобразование на 4-ех мерную гиперсферу.

Это преобразование записывается так:

(3) $\alpha = 2\pi * x / \textit{ширина\_изображения}$

(4) $\beta = 2\pi * y / \textit{высота\_изображения}$

(5) $x_1 = \sin(\alpha)$, $x_2 = \cos(\alpha)$,
$x_3 = \sin(\beta)$, $x_4 = \cos(\beta)$

где (x_1, x_2, x_3, x_4) точка на гиперсфере.

Добавление каждой точки на базовый уровень делается по следующему правилу: координаты на гиперсфере заносятся как весовые значения для нейрона базового уровня, а значения цвета для нейронов выходного уровня. Этим самым, мы получаем не совсем стандартную нейронную сеть, поскольку стандартная работает с числами в пределах [0, 1], либо [-1, 1]. Но такой подход избавляет нас от необходимости писать специальный интерпретатор, что позволяет несколько выиграть время.

В данной работе, были исследованы два альтернативных варианта нейронов на последнем слое нейронной сети.

Назовем *максимизирующим нейроном*, нейрон, выбирающий максимум из всех входных сигналов, затем умножающий его на вес связи, по которой был выбран максимум.
Назовем *нейроном среднего взвешенного*, нейрон, вычисляющий среднее взвешенное скалярного произведения входного сигнала на веса связей, по которым пришел сигнал.

2. АЛГОРИТМ ВОССТАНОВЛЕНИЯ ТОЧКИ.

Сжатие и распаковка использует восстановление точки по базовым, поэтому опишем это в первую очередь. Восстановление точки – это фактически упрощенный цикл работы сети Цао Ена. На вход подаются координаты (x,y), а на выходе получаем цвет.

Для обработки всей сети Цао Ена (4,k,3) в модельном варианте описанном выше, где каждый нейрон функционирует, обрабатывается и связан со всеми нейронами. Поэтому, следует сделать k+7 обходов нейронов и k*7 связей, итого k*8+7 операций. На самом деле этот цикл сети можно свести к одному или двум

циклам из k итераций на основании следующего алгоритма:

Название: Простой алгоритм восстановления точки:
На входе: координаты точки: (x,y)
На выходе: цвет точки (color)
Промежуточные переменные: максимальное значение нейрона

Шаг 1: Преобразование координат на плоскости в сферические
Шаг 2: Присваиваем максимальному значению нейрона 0
Шаг 3: Цикл по всем базовым точкам
    Шаг 3.1: Вычисляем значение на выходе для текущей базовой точки
    Шаг 3.2: Если значение больше максимального, то присваиваем это значение максимальному значению нейрона
Шаг 4: Вычисляем цвет восстанавливаемой точки, как произведение максимального значения нейрона на цвет ассоциированный с этим нейроном.

Примечание: существует так же интерполирующий алгоритм восстановления точки, где вместо поиска максимума, рассчитывается среднее взвешенное по всем базовым точкам. Этот алгоритм использует среднее взвешенное на последнем уровне сети.

3. СЖАТИЕ ИЗОБРАЖЕНИЙ.

Весь алгоритм делится на 4 основных этапа. Для каждого этапа разработано несколько вариантов алгоритмов.

Основные этапы:
"*Построение сети*" – это подготовка данных для обучения, преобразование и занесение всех точек изображения в качестве базовых точек.
"*Обучении Сети*" – содержит несколько вариантов алгоритмов, исключающих все точки, без которых можно восстановить изображение с заданной точностью.
"*Преобразование сети*…" – содержит несколько вариантов алгоритмов, преобразующих сеть в более экономичный (внутренний) формат.
"*Сжатие*" - дополнительно сжимает данные без потерь. На данный момент реализован только алгоритм Хаффмана.[4]

Основными параметрами алгоритма сжатия изображения являются:
- допуск при обучении
- число рабочих строк
- число точек в рабочей окрестности
- чувствительность по x
- чувствительность по y

Допуск– задает порог допуска для обучения, такой что изображение будет восстановлено в рамках указанного допуска. Допуск задается в интервале от 0 до 255.
Число рабочих строк и число *точек в рабочей окрестности* определяет рабочую область для нейронной сети, это значит, что сеть строится и функционирует только для этой области. Более подробно о конкретном использовании рабочих областей описано в алгоритмах обучения.
Чувствительность по x и по y определяет крутизну сигнала от базовой точки.

4. АЛГОРИТМЫ ОБУЧЕНИЯ.

Простой предсказывающий алгоритм:

Простой алгоритм сжатия использует:
n – число рабочих строк
m – точек в рабочей окрестности
простой алгоритм восстановления точки (см. выше)
статическую рабочую область – рабочая область n на m точек.
Буфер строк – содержит n строк изображения, используется для увеличения скорости чтения и изменения информации

Точки сети проверяются последовательно слева на право сверху вниз. По ходу проверки в буфер строк загружаются последние n используемых строк, а рабочая область служит локальной сетью Цао Ена для изображения. По сути, статическая рабочая область имеет постоянные размеры. Одним из достоинств алгоритма является возможность обрабатывать данные потоком – это значит, что данные могут поступать непрерывным потоком и достаточно быстро обрабатываться. Алгоритм основывается на возможности сети предсказать цвет следующей точки в потоке при известных предыдущих. Таким образом, для каждой точки проверяется можно ли ее восстановить на основании уже имеющихся точек в рабочей области. Если не удается восстановить точку с заданным допуском, заносим ее в рабочую область или в глобальную сеть для всего изображения.

Алгоритм:
На входе: нейронная сеть, содержащая все точки изображения
На выходе: нейронная сеть, содержащая базовые точки для изображения
Промежуточные переменные: буфер строк, рабочая область, рабочая точка

Шаг 1: Цикл по всем строкам изображения
Шаг 1.1: Загружаем строку в буфер строк
Шаг 1.2: Перебираем все точки в последней строке буфера
Шаг 1.2.1: Заносим столбец из буфера строк в рабочую область

Шаг 1.2.2: Удаляем рабочую точку из рабочей области
Шаг 1.2.3: Пытаемся восстановить точку по рабочей области, если восстановление не удается, то рабочая точка заносится в сеть обратно
Шаг 1.2.4. Если рабочая область содержит m столбцов, то удаляем первый столбец из рабочей области.
Шаг 1.3: Если загружено n строк то удаляем первую строку из буфера строк.

Все тесты реализации алгоритма проводились на компьютере с процессором Intel Pentium II – 350, где сжатие изображения 540x370, с допуском 10 составляет 6 секунд, а время восстановления 2 секунды. При этом *коэффициент сжатия,* который определяется в нашем случае как кол-во оставшейся информации в %, составил 31%.

Динамический интерполирующий алгоритм
Явным преимуществом динамического алгоритма является замена статических рабочих областей на динамические. Главным условием построения динамической области является обязательное наличие точек на правой и левой границах рабочей строки и отсутствие точек между граничными. Причем количество рабочих строк в данной реализации алгоритма остается постоянным, хотя теоретически и оно может быть динамическим. Такая область гарантирует нам, что все точки между граничными будут интерполироваться без вмешательства точек этого же ряда при не большой чувствительности. Это значит, что интерполяция будет проходить по ближайшим точкам. Безусловно, вмешательство точек верхних строк неизбежно при статическом количестве рабочих строк, но так как вклад каждой верхней строки менее значителен, чем строк в рабочей строке, вмешательство это становиться менее ощутимым и далеко не всегда негативным.

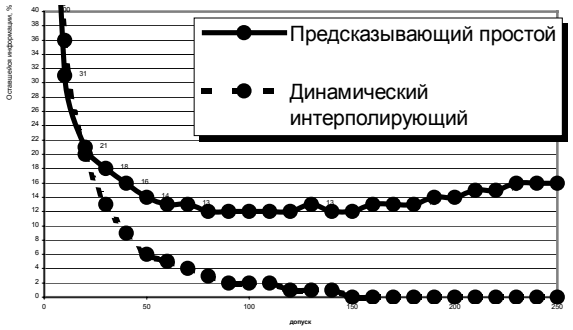
Рабочая точка находиться не в крайней левой позиции, как в предыдущих алгоритмах, а на точку левее от этой позиции. Это обусловлено появлением граничных точек.
Из условий накладываемых на рабочую область, получается, что с каждым удалением рабочей точки мы расширяем рабочую область, в противном случае рабочая область “схлопывается”. Под “*схлопыванием” рабочей области*, будем понимать удаление всех точек слева на право до получения области размером 3 на n. При обучении область никогда не может быть меньше чем 3 на n, т.к. между двумя граничными точками должна быть рабочая точка. Поэтому одной из особенностей результата работы алгоритма является обязательное присутствие правой и левой границ изображения, что учитывается при преобразовании изображения в формат для хранения.

Алгоритм:

На входе: нейронная сеть, содержащая все точки изображения
На выходе: нейронная сеть, содержащая базовые точки для изображения
Промежуточные переменные: буфер строк, рабочая область, рабочая точка

Шаг 1: Цикл по всем строкам изображения
Шаг 1.1: Загружаем строку в буфер строк
Шаг 1.2: Создаем рабочую область 2xn
Шаг 1.3: Цикл по всем точкам последней строки буфера строк
Шаг 1.3.1: Удаляем текущую точку из рабочей области и расширяем рабочую область
Шаг 1.3.2: Проверяем возможность восстановления всех точек в последней строке рабочей области
Шаг 1.3.3: Если не удалось восстановить, то восстанавливаем рабочую точку обратно и проводим схлопывание рабочей области
Шаг 1.3.4: Если загружено m столбцов в рабочую область, то удаляем первый столбец из нее
Шаг 1.4: Если загружено n строк в буфер строк, то удаляем первую строку из буфера

Сжатие/восстановление изображения 540x370, с допуском 10, проходило 22 секунд / 2 секунды. Коэффициент сжатия в этом случае составил 36%. Ниже приведен график, сравнивающий эти два алгоритма:



Как видно из графика, при больших допусках динамический алгоритм дает лучшие результаты сжатия, конечно при этом он проявляет свои эффекты, при больших коэффициентах сжатия.

5. АЛГОРИТМЫ ПРЕОБРАЗОВАНИЯ СЕТИ ДЛЯ ХРАНЕНИЯ.

До этого раздела была рассмотрена первая и основная часть сжатия изображения, но преобразования в формат для хранения и последующее сжатие его (о чем речь пойдет в следующей главе) не менее важные вопросы, поскольку без решения этих проблем применение алгоритма не имело бы никакого смысла.

Данные сети представляют собой пятерку чисел (x,y,R,G,B), причем x,y – Integer, а R,G,B – byte. То есть, для хранения нужно 2*2+3 = 7 байт на каждую точку, что даже при 10% сжатии даст только 23% реального сжатия. Это конечно возможный выход, но не самый удовлетворительный, потому было предложено использовать несколько форматов хранения данных, следующие два формата оказались наиболее хорошими и на настоящий момент реализованы:

Тегированный формат.

Идея состоит в представлении непрерывных участков данных некоторыми кусочками – тегами. Каждый тег содержится в линии. Тег содержит следующую информацию – (x, count, bits) , где x, count – однобайтовые числа, а bits – это тройка (R,G,B). Как видно из определений x и count не могут принимать значений больших 255. Поэтому, когда возникает кусок непрерывных данных длиной более 255 точек, он кодируется двумя тегами. А в большие изображения для представления данных в области x>255 разбиваются на фреймы или полоски. Каждая тег-линия содержит только информацию о количестве тегов в линии, ее номер вычисляется автоматически по возрастанию.

Таким образом, весь формат представляется иерархией: Фрейм -> тег-линия -> тег -> точки.

Реализовав такую иерархию, мы сокращаем количество хранимых данных.

RCE (Run count encoding)

Этот алгоритм несколько похож на известный всем алгоритм RLE, но с тем отличием, что RLE сжимает повторяющиеся последовательности, а RCE исключает из данных пустоты, то есть данные на выходе RCE-преобразования выглядят как совокупность точек и кол-ва пропусков.

Алгоритм одномерный, поэтому перед его использованием или во время его использования данные преобразуются в одномерный поток.

Алгоритм кодирования. Входной поток сканируется и вычисляется количество подряд идущих полезных точек и количество пропусков точек подряд. Все это затем записывается в формате изображенном на рисунке. Это преобразование в отличии от предыдущего никогда не может выдать на выходе больше данных чем хранилось в сети. Так как на каждую точку в сети приходится 4 байта служебной информации, а здесь не более одного, что гарантирует усиление сжатия.

Алгоритм распаковки создает пустую сеть и, по мере поступления данных, то записывает их в сеть, то пропускает нужное количество.

Улучшение компактности для алгоритма RCE.

Понятно, что для хранения длины значимых и выкинутых точек требуется некоторое число бит, которое для каждого вхождения такой группы точек не одинаково. В первой реализации этого алгоритма использовалась статическая длина слова (под *словом* в данном случае будем считать то кол-во бит, что отводится на хранение длины последовательности значимых точек или кол-ва выкинутых), которая либо высчитывалась как максимальная обеспечивающая хранение данных, либо заранее устанавливается. Позже, алгоритм был усовершенствован, введением динамической длины слова. Для этого, изначально определяется некоторая не нулевая текущая длина кодирования и вводится тег с нулевой длиной, как тег смены длины слова. Алгоритм работает после отработки простого RCE, вторым проходом. Пробегая по сформированным тегам из пропусков и значимых точек, вычисляется требуемая для хранения длина слова. Если длина слова больше текущей, вставляется тег смены длины, меняется текущая длина слова и продолжается просмотр. Если же длина меньше текущей, просматривается n точек вперед до того, пока не встретится длина слова, большая, чем у просматриваемой точки и вычисляется выигрыш от смены кодирования и цена смены по формуле

Выигрыш: ***(current_len – current_point_len)*n ,***
где current_len – текущая длина слова,
current_point_len – длина слова для текущей точки

Цена смены: ***current_len + 5***
и если выигрыш от смены больше чем цена, то производится смена, иначе продолжается сканирование.

6. АЛГОРИТМЫ ВОССТАНОВЛЕНИЯ ИЗОБРАЖЕНИЯ.

Алгоритм восстановления является обратным к алгоритму сжатия, поэтом все этапы проводятся в обратном порядке. Он делиться на 3 этапа:
Распаковка
преобразование формата в сеть
восстановление изображения по сети

Алгоритмы преобразования формата данных в сеть были уже описаны выше как обратные преобразования RCE и тегированного формата. Распаковка – это применение алгоритма Хаффмана, в нашем случае.

Для работы алгоритмов восстановления требуются следующие параметры:

- рабочих строк
- точек в рабочей окрестности (для предсказывающих алгоритмов)
- чувствительность по X и по Y

Алгоритмы восстановления изображений являются обратными для тех алгоритмов, которыми проводилось сжатия.

Простой предсказывающий алгоритм восстановления.
В этом алгоритме применяется статическая рабочая область. Статическая область строится так же, как и в одноименных алгоритмах обучения, поэтому для восстановления изображения нужно пройти по всем точкам получаемого изображения и восстановить их с соответствующей рабочей областью.

Динамический интерполирующий алгоритм восстановления.
Как и в одноименном алгоритме обучения, так и тут применена динамическая рабочая область и интерполирующая функция восстановления точки. При восстановлении изображения мы обходим все точки изображения, так что бы динамическая область в рабочей строке содержала только текущую точку и границы, если точка отсутствует в сети. Если же точка есть в сети, то она восстанавливается как есть, то есть просто копируется в изображение.

Время восстановления в среднем для обоих алгоритмов составляет 0.1 от времени сжатия.

7. ДРУГИЕ ВОЗМОЖНОСТИ.

Помимо своего основного предназначения – сжатия растровых изображений, алгоритм CNN можно применять и для восстановления переданных с искажениями изображений. Такая возможность может позволить использовать этот алгоритм для односторонней передачи изображений – это значит, что данные не будут пересылаться отправителем повторно, а получатель вовсе не будет посылать никаких сообщений, он будет только принимать. Это позволит использовать для передачи изображений односторонний канал и увеличить скорость передачи за счет отсутствия дополнительных пересылок. Правда надо заметить, что подобный алгоритм может применяться только там, где не нужна 100% достоверность изображения. Таким примером могут быть FAX-сообщения или подобные. Причем надо заметить, что даже при очень большой зашумленности канала, так что из информации удастся получить только 30%, восстановление будет вполне распознаваемо глазом. Для реализации такой возможности нужно будет только учесть, что протокол передачи изображений должен будет выявлять неверно переданные точки. Это возможно с применением ряда уже широко распространенных алгоритмов типа кода Рида-Саламона или по контрольной сумме. Надо заметить, что для восстановления будет применяться алгоритм схожий с алгоритмом восстановления изображения по сети и потому имеющий схожие временные характеристики. Потому можно сказать, что восстановление будет проходить

8. СРАВНЕНИЯ С JPEG.

Сжатие проводилось с:
- допуском: 30
- рабочих строк в сети: 2
- рабочих точек в окрестности: 5
- чувствительность по x : 1000
- чувствительность по y : 1000

Изображение	CNN	JPEG	BMP
облако [1536x1024]	377Kb 206Kb *	186Kb	4609Kb
черно-белое (полноцветное фото, портрет) [624x479]	72Kb 38Kb*	62Kb	876Kb
цветное фото (портрет) [254x384]	61Kb 42Kb*	32Kb	287Kb
Текстура [385x527]	247Kb 171Kb *	247Kb	595Kb
изображение с высокой частотой на черном фоне [407x480]	149Kb 155Kb *	43Kb	574Kb
Фотография здания [617x696]	316Kb 224Kb *	64Kb	1259Kb
Деловая графика [818x845]	321Kb 208Kb *	84Kb	2027Kb
Светлое изображение [280x185]	17Kb 12Kb*	12Kb	152Kb

Примечание: * - был использован алгоритм Хаффмана, для усиления коэффициента сжатия

9. РЕАЛИЗАЦИЯ.

Данный алгоритм был реализован для компьютеров серии Pentium, под операционную систему Windows 95, 98, NT/2000, на языке программирования Visual C++ 6.0. Вычислительное ядро написано на ANSI C, что позволяет его переносить под UNIX. Для тестирования и исследования алгоритма был создана программа, имеющая удобный графический интерфейс.
В программу вошли: 3 алгоритма обучения, 3 алгоритма усиления сжатия и дополнительное сжатие при помощи алгоритма Хаффмана.

Программа позволяет:

- считывать изображения в формате BMP и CNN
- производить сжатие и распаковку
- просматривать и сравнивать результаты визуально
- сохранять базовые точки для изображения
- настраивать параметры обучения
- загружать в сеть испорченные изображения и восстанавливать их

10. ЗАКЛЮЧЕНИЕ.

На данный момент проводятся работы по более детальному изучению и доработки алгоритма CNN, изучение и исследование схожих алгоритмов на основе нейронных сетей, анализ классов изображений. Хотелось бы отметить перспективы развития данной области.

Представляется перспективной:

- проверка возможности перехода с RGB на HUE
- проверка использования алгоритма Хаффмана для каждой цветовой плоскости по отдельности, используя особенности статических областей
- увеличение сжатия за счет хранения только разности статических областей
- разработка алгоритмов для сжатия видео
- разработка алгоритмов для сжатия трехмерных сцен и срезов.
- Разработка алгоритмов для гладкого масштабирования при помощи сети Цао Ена.
- Учтя результаты работы с статической областью, можно построить более эффективный нежели RCE алгоритм, основанный на повторяемости статических областей, в местах с малой частотой. Предполагается, что это позволит увеличить допуск от 1.5 до 3х раз.
- Тестирование возможностей распараллеливания вычисления при сжатии
- Перенос под UNIX

11. REFERENCES

[1] Программирование №4, 1992, стр. 4-16

[2] А.Н. Горбань Обучение нейронных сетей. СП ПараГраф Москва. 1990.

[3] Нейронные сети: обучение без учителя. http://www.neuropower.de/rus/books/001/htm/gl3_4.htm

[4] Сжатие по алгоритму Хаффмана http://algolist.vpro.ru/compress/standart/huffman.html

[5] Алгоритмы сжатия изображений http://www.sf.amc.ru/~dv/fractal/algcomp1.htm

About the author

Куликов Александр Иванович – н.с. ИВТ СО РАН.
Михальченко Николай – студент НГУ.

E-mail: kulikov@ict.nsc.ru, gst@pisem.net.