

УДК 519.72

М.И.ШЛЕЗИНГЕР

БЫСТРАЯ РЕАЛИЗАЦИЯ ОДНОГО КЛАССА ЛИНЕЙНЫХ СВЕРТОК

Описана вычислительная схема линейной фильтрации изображений, обеспечивающая независимость объема вычислений от размера окошка, в котором выполняется фильтрация.

1. Основные определения и примеры быстрых фильтраций.

Пусть T - множество пар (i, j) целых чисел, т.е. двумерная целочисленная решетка; V – множество целых чисел. Изображение – это функция $T \rightarrow V$. Предположим, что изображения имеют ограниченное поле зрения. Это значит, что любое рассматриваемое изображение $v: T \rightarrow V$ таково, что для него существует такой прямоугольный участок $T_V = \{(i, j): i' \leq i \leq i'', j' \leq j \leq j''\}$, что для любой точки $(i, j) \notin T_V$ вне этого участка $v(i, j) = 0$. Количество точек на этом участке назовем размером изображения.

Пусть заданы два изображения: $y: T \rightarrow V$ и $x: T \rightarrow V$, первое из которых назовем маской. Линейной сверткой $y \times x$ изображения x с маской y , как известно [1], называется изображение z , (i, j) -я компонента которого определяется выражением $z(i, j) = \sum_{k,l} y(i-k, j-l) \cdot x(k, l)$. Линейные свертки изображений являются распространенным приемом при их обработке. Основная трудность, с которой сталкивается конструктор после того, как он уже пришел к выводу о целесообразности включения свертки в технологическую цепочку обработки, состоит в ее трудоемкости. В общем случае количество вычислений для получения значения результирующего изображения в одной точке пропорционально размеру маски.

В зависимости от того, как в известных работах учитывается эта трудность, в них можно выделить две основные группы. Работы первой группы завершаются скороговоркой о необходимости специализированной аппаратной реализации для свертки, т.е. доводка алгоритма обработки до практически пригодного вида как бы откладывается или перекладывается на кого-то другого. В работах второй группы рекомендуется ограничиваться лишь свертками малых размеров, т.е. использовать вместо требуемой обработки ее грубое приближение

В данной работе описан общий прием, позволяющий по крайней мере для определенного класса сверток строить алгоритмы, трудоемкость которых вообще не зависит от размера маски. Приведем несколько примеров таких алгоритмов.

Пример 1. Простейшая иллюстрация – это так называемое суммирование по скользящему интервалу, когда последовательность $\dots, x(-1), x(0), \dots, x(i) \dots$ должна быть преобразована в последовательность $\dots, y(-1), y(0), y(1) \dots$ по формуле $y(i) = \sum_{j=0}^{n-j} x(i-j)$, где n - ширина интервала.

Хотя здесь для вычисления одного (единственного) значения и требуется выполнить $(n-1)$ сложений, однако для преобразования всей последовательности длиной m требуется не $m \cdot (n-1)$, а $m \cdot 2$ сложений. Для этого следует реализовать вычисления не по исходной формуле, а по эквивалентному рекуррентному выражению

$y(i) = y(i-1) + x(i) - x(i-n)$, которое требует не $(n-1)$ сложений на каждую точку, а всего 2 сложения вне зависимости от ширины интервала.

Пример 2. Рассмотрим двумерное обобщение предыдущего примера, а именно, суммирование по скользящему окошку. Оно заключается в преобразовании изображения с компонентами $x(i, j)$ в изображение y , (i, j) -я компонента которого вычисляется по формуле $y(i, j) = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} x(i-k, j-l)$.

Непосредственное применение приема, использованного в предыдущем примере, дает схему вычислений по рабочей формуле

$y(i, j) = y(i, j - 1) + \sum_{k=0}^{m-1} x(i - k, j) - \sum_{k=0}^{m-1} x(i - k, j - n)$, которая обеспечивает определенный выигрыш, а именно, $2 \cdot (m - 1)$ сложений вместо $(m \cdot n - 1)$ сложений.

Определенные (хотя и не очень большие) усилия позволяют получить схемы, реализация которых имеет трудоемкость, не зависящую ни от m , ни от n . Для этого необходимо ввести в рассмотрение промежуточное изображение u , компоненты которого получаются из x в соответствии с выражением $u(i, j) = u(i - 1, j) + x(i, j) - x(i - m, j)$, преобразуемое в y по формуле $y(i, j) = y(i, j - 1) + u(i, j) - u(i, j - n)$. Реализация этой пары формул требует, как видно, 4 операции на каждую точку изображения вне зависимости от размера окошка.

Пример 3. Пусть входная последовательность $\dots, x(-1), x(0), x(1), \dots$ должна быть преобразована в последовательность $y(i)$ по формуле $y(i) = \sum_j w(i - j) \cdot x(i)$, где $w(i)$ – компоненты так называемой треугольной маски шириной $2n - 1$, равные $n - |i|$ при $|i| < n$, и равны 0 для всех прочих i . Требуется реализовать это преобразование так, чтобы количество вычислений не зависело от ширины маски n .

В первом примере требуемый алгоритм находился буквально на поверхности. Во втором примере алгоритм, хотя и не лежит на поверхности, однако при желании находится достаточно быстро. В последнем примере требуемый хороший алгоритм, конечно же, может быть найден, однако его поиск уже требует определенной программистской сообразительности. Далее будет описан общий прием, позволяющий для определенного класса масок конструировать вычислительные схемы линейных сверток даже при отсутствии программистских навыков и опыта.

2. Описание общего приема быстрых сверток

Пусть L и L^{-1} – две маски, обладающие тем свойством, что в изображении $x = L \times L^{-1}$ все компоненты $x(i, j)$ равны нулю, кроме компоненты $x(0, 0)$, которая равна 1. Это значит, что $L \times L^{-1} \times x = x$ для любого изображения x .

Общеизвестны два свойства сверток, а именно, $x \times y = y \times x$ для любой пары изображений, и $z \times (y \times x) = (z \times y) \times x$ для любой тройки изображений. Из этого следует, что свертка $w \times x$ любого изображения x с любой маской w дает тот же результат, что и свертка $(L \times w) \times (L^{-1} \times x)$ несколько преобразованного изображения $L^{-1} \times x$ с соответствующим образом преобразованной маской $L \times w$. При этом для целого класса сверток их реализация в виде $(L \times w) \times (L^{-1} \times x)$ оказывается значительно более эффективной, чем реализация в исходном виде $w \times x$.

Определим две пары масок вида L и L^{-1} : Δ_x, Δ_x^{-1} и Δ_y, Δ_y^{-1} . Маска Δ_x преобразует изображение x с компонентами $x(i, j)$ в изображение $y(i, j) = x(i, j) - x(i, j - 1)$. Маска Δ_y реализует преобразование $y(i, j) = x(i, j) - x(i - 1, j)$.

Преобразование Δ_x^{-1} определяется выражением $y(i, j) = \sum_{k=-\infty}^j x(i, k)$, а Δ_y^{-1} – выражением $y(i, j) = \sum_{k=-\infty}^i x(k, j)$. Очевидно, что преобразования Δ_x и Δ_y требуют по одной операции на каждую точку. Такова же трудоемкость преобразований Δ_x^{-1} и Δ_y^{-1} , так как они должны быть реализованы соответственно по выражениям $y(i, j) = y(i, j - 1) + x(i, j)$ и $y(i, j) = y(i - 1, j) + x(i, j)$.

Рассмотрим на примерах, как описанных выше, так и дополнительных, каким образом использование пар Δ_x, Δ_x^{-1} и Δ_y, Δ_y^{-1} позволяет ускорить вычисление сверток.

Пример 4. Преобразование, рассмотренное в первом примере, есть свертка $y = w \times x$ последовательности x с маской w , в которой компоненты

$w(i), 0 < i \leq n - 1$ равны 1, а остальные – равны нулю. Вычисление этой свертки в виде $(\Delta \times w) \times (\Delta^{-1} \times x)$ требует лишь две операции для каждого значения i , так как маска $\Delta \times w$ содержит лишь две ненулевые компоненты, а реализация преобразования

Δ^{-1} , как было показано выше, требует одной операции на клетку. В развернутом виде эти вычисления задаются парой формул: $u(i) = u(i - 1) + x(i)$, $y(i) = u(i) - u(i - n)$.

Пример 5. Преобразование в примере 2 есть свертка $y = w \times x$, в которой маска w имеет компоненты $w(i, j)$ равные 1, если $0 \leq i < m$ и $0 \leq j < n$ и равные 0 во всех прочих случаях. Преобразование этой маски с помощью маски $\Delta_x \times \Delta_y$ дает маску w' , содержащую лишь четыре ненулевые компоненты: $w(0, 0) = w(m, n) = 1$ и $w(0, n) = w(m, 0) = -1$. Свертка с такой маской требует лишь три операции сложения на каждую точку изображения. Маска w' , однако, должна применяться не к исходному изображению x , а изображению $\Delta_x^{-1} \times \Delta_y^{-1} \times x$, получение которого требует еще двух операций на каждую точку. Реализация суммирования по скользящему окошку размером $m \cdot n$ в виде $(\Delta_x \times \Delta_y \times w) \times (\Delta_x^{-1} \times \Delta_y^{-1} \times x)$ требует в итоге пять операций на каждую точку вне зависимости от величин m и n . В развернутом виде эта реализация задается тройкой формул

$$u(i, j) = u(i - 1, j) + x(i, j); \quad v(i, j) = v(i, j - 1) + u(i, j);$$

$$y(i, j) = v(i, j) + v(m, n) - v(i, j - n) - v(i - m, j)$$

Пример 6. Свертка $y = w \times x$ последовательности x с треугольной маской w , рассмотренная в примере 3, должна быть реализована в виде $(\Delta \times \Delta \times w) \times (\Delta^{-1} \times \Delta^{-1} \times x)$. Маска $\Delta \times \Delta \times w$ содержит лишь три ненулевых компоненты и должна применяться к последовательности $\Delta^{-1} \times \Delta^{-1} \times x$. Суммарные вычислительные затраты здесь составляют пять операций сложения или четыре операции сложения и одну операцию умножения на каждый элемент последовательности, что очевидно по рабочим формулам:

$$u(i) = u(i - 1) + x(i); \quad v(i) = v(i - 1) + u(i);$$

$$y(i) = v(i + n - 1) - 2 \cdot v(i - 1) + v(i - n - 1)$$

Пример 7. Рассмотрим маски, которые представляют собой многоугольники, составленные из горизонтальных и вертикальных линий, причем компоненты масок, соответствующие точкам внутри многоугольников, равны 1, а соответствующие точкам вне многоугольников – 0. Примеры таких масок приведены на рис.1 (первые маски в каждом столбце). Свертка изображения с такими масками позволяет выделять на изображении места, подозрительные на наличие в них стандартной конфигурации, задаваемой маской. По-прежнему требуется найти такую схему вычисления свертки, трудоемкость которой не зависит от размеров искомой конфигурации, взаимного расположения ее частей и т.п. Воспользуемся для этого тем же общим приемом.

Подвергнем маски преобразованию и получим изображения (средние в каждом столбце на рис.1), в которых количество ненулевых компонент не зависит от вертикальных размеров маски. Свертывая полученные изображения с маской Δ_x , получим изображения (нижние в каждом столбце), в которых количество ненулевых компонент не зависит от геометрических размеров исходных масок (в рассматриваемом примере – 12). Именно эти изображения должны использоваться в качестве масок. Однако применяться они должны не к исходным изображениям, а к изображениям, полученным из исходных с помощью преобразования Δ_x^{-1} и Δ_y^{-1} . Суммарная трудоемкость требуемого преобразования по полученной схеме составляет 13 операций сложения на точку, что, как правило, меньше, чем m - количество ненулевых компонент в исходных масках.

Рассмотренные примеры позволяют сформулировать в общем виде рекомендацию по конструированию быстродействующих вычислительных схем свертывания изображений. А именно, необходимо исходную маску преобразовывать с помощью дифференциальных операторов Δ_x и Δ_y столько раз, пока не получится маска, в которой количество ненулевых компонент не зависит от размера исходной маски, если это, конечно же, вообще возможно. В случае же, когда исходная маска является кусочно-полиномиальной функцией целочисленных переменных i и j , причем область определения каждого куска есть прямоугольный участок двумерной целочисленной решетки, то это всегда возможно. Это и определяет область безусловной применимости сформулированной рекомендации. В то же время описанный прием может быть полезен для приближенной реализации и тех сверток, которые не являются кусочно-полиномиальными. Покажем, как это происходит, на следующем примере.

Пример 8. Эффективность свертывания последовательности с маской, имеющей вид гауссовой кривой $e^{-\frac{|i|^2}{k^2}}$, показана в [2]. Вычисление таких сверток очень трудоемко, особенно при больших k . Многократное применение дифференциального оператора Δ к этой маске не упрощает ее в смысле количества ненулевых компонент. В то же время гауссова кривая достаточно точно может быть аппроксимирована (с точки зрения ее использования как маски) кусочно-линейно-квадратичной кривой, как показано на рис.2. Преобразование ее с помощью маски $\Delta \times \Delta \times \Delta$ приводит к маске, содержащей 6 ненулевых компонент независимо от параметра k , т.е. ширины исходной гауссовой кривой (см. рис.2).

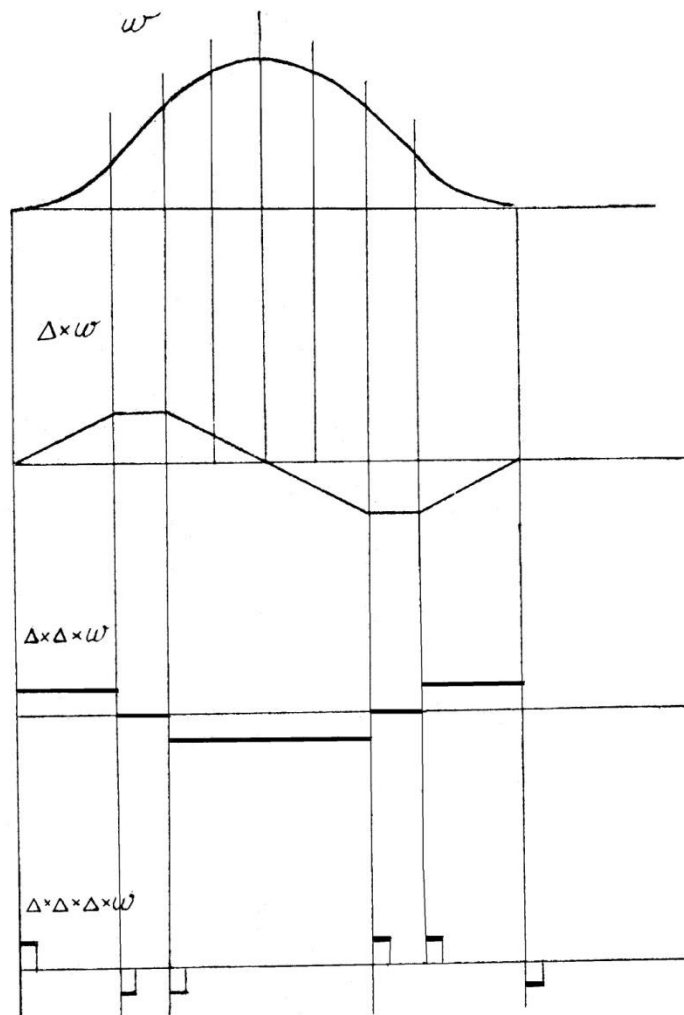


Рис. 2

3. О точности представления промежуточных данных

Пусть компоненты исходного изображения v принимают значения от 0 до $2^{n_1} - 1$, т.е. требуют для своего запоминания n_1 двоичных разрядов, компоненты маски w запоминаются в n_2 двоичных разрядах, а максимально возможный размер маски равен 2^{n_3} . В таком случае хранение выходных результатов, т.е. компонент изображения $w \times x$, требует M -разрядных двоичных ячеек памяти, $M = n_1 + n_2 + n_3$. Для вычисления же свертки непосредственно по формуле $y(i, j) = \sum_{k, l} w(i - k, j - l) \cdot x(k, l)$ необходимо M -разрядное арифметическое устройство, т.е. устройство, правильно вычисляющее величины $x + y$ и $x \cdot y$ при условии, что как операнды, так и результат помещаются в M -разрядную двоичную сетку.

Трудность (и как мы увидим далее, кажущаяся) в предлагаемой быстрой реализации свертки состоит в том, что предполагается вычисление промежуточных данных, которое не может быть выполнено на M -разрядном арифметическом устройстве, так как эти промежуточные данные могут принимать значения, превосходящие 2^M . Более того, невозможно указать, какова требуемая разрядность процессора, так как промежуточные данные при увеличении размеров изображения могут принимать сколь угодно большие значения.

Покажем, что эта трудность легко преодолима. Для любого целого числа x обозначим $Mod(x)$ такое целое число x' , что $0 < x' < 2^M$, и существует такое целое число k , такое, что $x = k \cdot 2^M + x'$. Иначе говоря, $Mod(x)$ - это M младших разрядов в двоичном представлении числа x . Для функции $Mod(x)$ справедливы два равенства: $Mod(x + y) = Mod(Mod(x) + Mod(y))$ и $Mod(x \cdot y) = Mod(Mod(x) \cdot Mod(y))$, которые выражают тот очевидный факт, что содержимое младших разрядов суммы и произведения зависит только от младших разрядов слагаемых и сомножителей. Из этих равенств следует, что если результатом работы какой-то программы является величина $Mod(x)$, полученная с помощью программы, вычисляющей лишь суммы и произведения, то любые исходные и промежуточные величины могут быть представлены не своими действительными значениями, а лишь своими младшими разрядами. Поскольку в нашем случае для любой выходной величины x предполагается, что $0 < x < 2^M$, то, $Mod(x) = x$, и справедливо следующее утверждение.

Если вычисление свертки $w \times x$ непосредственно по формуле

$y(i, j) = \sum_{k, l} w(i - k, j - l) \cdot x(k, l)$ может быть реализовано на арифметическом устройстве определенной разрядности, то предлагаемая в статье схема быстрой свертки также может быть реализована на устройстве такой же разрядности, несмотря на то, что в процессе вычислений по предлагаемой схеме могут быть получены промежуточные данные, выходящие за пределы разрядной сетки. Для получения конечных и правильных результатов достаточно заблокировать останов процессора по переполнению.

С п и с о к л и т е р а т у р ы

1. Прэтт У. Цифровая обработка изображений: Пер. с англ. – М. : Мир, 1982. – Кн.1. – 312 с.; Кн.2 – 480 с.
2. Шульга В.И. Представление контурного изображения отрезками линий с помощью бэл-аппроксимации // Математические и технические средства робототехники и распознавания образов. – Киев : Ин-т кибернетики АН УССР. 1981. – С. 31-42.