Computer Vision I
Ex. 2: Filtering Techniques

Dmitrij Schlesinger, Carsten Rother

WS2016/2017, 25/26.10.2016

COMPUTER VISION LAB DRESDEN

TECHNISCHE UNIVERSITÄT DRESDEN

Notations

$D \subseteq \mathbb{Z}^2$ – the domain (image grid), $(i, j) \in D$ is a pixel

Image is a mapping (function) $f : D \to C$ (color space)

$f(i, j)$ is the pixel color at the position (i, j)

$h(k, l)$ is the convolution mask (kernel, filter ...)

Note: it is also an "image", i.e. $h : D \to \mathbb{R}$

Linear filtering or **convolution** is an operator $f \mapsto g$:

$$g(i, j) = \sum_{k,l} f(i + k, j + l) \cdot h(k, l)$$

Filtering

Replace each pixel by a linear combination of its neighbours and itself.

2D convolution (discrete): $g = f * h$

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

*

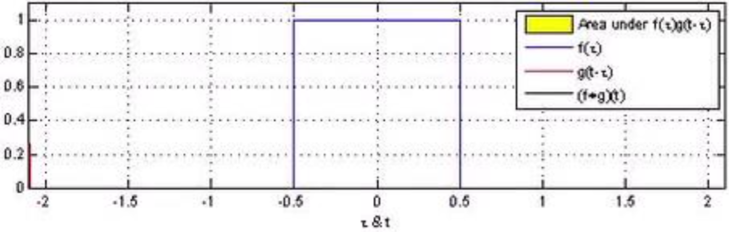
0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

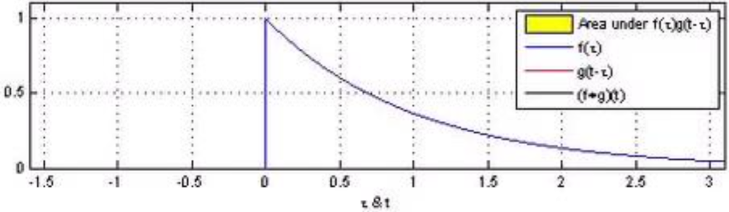
=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

Image $f(i,j)$ $*$ kernel $h(k,l)$ $=$ filtered image $g(i,j)$

Example – mean filter (1D)





Linear filtering (1D) – a naïve algorithm

One-dimensional signal for simplicity, the mean filter reads

$$g(i) = \frac{1}{2w+1} \cdot \sum_{i'=i-w}^{i+w} f(i')$$

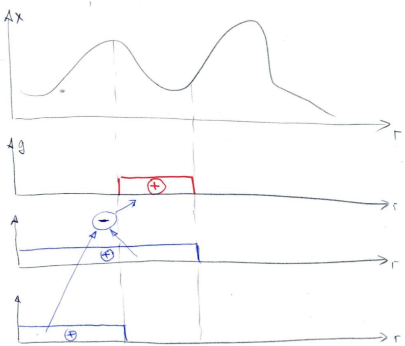
A naïve algorithm (according to the formula)

for $i = 0 \dots n$ **do**
 $sum = 0$
 for $i' = i - w \dots i + w$ **do**
  $sum = sum + f(i')$
 $g(i) = sum/(2w+1)$

Time complexity: $O(nw)$

Linear filtering (1D) – a better algorithm

A better algorithm – the idea:



If the "auxiliary" sums $\tilde{f}$ are pre-computed in advance, only one summation per value is needed for the output !!!

$$\sum_{i'=i-w}^{i+w} f(i') = \sum_{i'=0}^{i+w} f(i') - \sum_{i'=0}^{i-w-1} f(i') = \tilde{f}(i+w) - \tilde{f}(i-w-1)$$

Linear filtering (1D) – a better algorithm

It is indeed very simple to pre-compute $\tilde{f}(i)$ fast for all i

A better algorithm:

1. Compute $\tilde{f}(i)$ for all i

   for $i = 0 \ldots n$ **do**

   $$\tilde{f}(i) = \tilde{f}(i-1) + f(i)$$

2. Compute the output $g(i)$

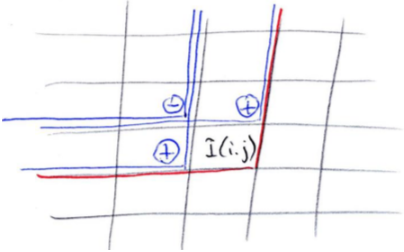
   for $i = 0 \ldots n$ **do**

   $$g(i) = \Big(\tilde{f}(i+w) - \tilde{f}(i-w-1)\Big)/(2w+1)$$

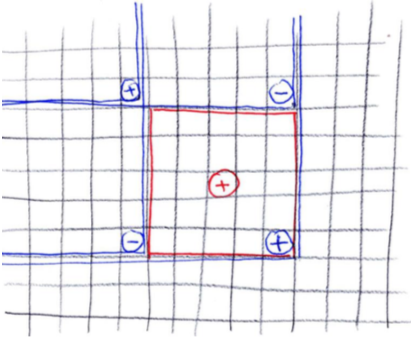
Time complexity: $O(n)$
i.e. it does not depend on the window size at all !!!

Generalization to the 2D-case:

1) Compute $\tilde{f}$



2) Compute g



Linear filtering (2D) – “Intergral Image”

Generalization to the 2D-case:

1. Compute $\tilde{f}(i,j)$ for all (i,j)

   for $(i,j)=(0,0)\dots(m,n)$ (row-wise) **do**

   $$\tilde{f}(i,j)=\tilde{f}(i{-}1,j)+\tilde{f}(i,j{-}1)- \\ -\tilde{f}(i{-}1,j{-}1)+f(i,j)$$

2. Compute the output $g(i,j)$ for all (i,j)

   for $(i,j)=(0,0)\dots(m,n)$ (row-wise) **do**

   $$g(i,j)=\Big(\tilde{f}(i{+}w,j{+}w)-\tilde{f}(i{+}w,j{-}w{-}1)- \\ -\tilde{f}(i{-}w{-}1,j{+}w)+\tilde{f}(i{-}w{-}1,j{-}w{-}1)\Big)/(2w+1)^2$$

Time complexity: $O(n)$
again, does not depend on the window size at all !!!

Convolutions

$$g = f * g \qquad g(i) = \sum_{i'=-\infty}^{\infty} f(i - i') \cdot h(i')$$

Properties:

- **Commutative**: $f * h = h * f$
- **Associative**: $(f * h^1) * h^2 = f * (h^1 * h^2)$
- **Distributive** with "+": $f * (h^1 + h^2) = f * h^1 + f * h^2$

- **Identical** convolution h^I has the property $f * h^I = f$, i.e. it does not change the input – delta-function.

- **Inverse** convolutions fulfill $h * h^{-1} = h^I$

Convolutions

Example for inverse convolutions ($i' = 0$ is marked bold):

$$\begin{aligned} h^{diff} &= [\ldots, 0, 0, 0, \mathbf{1}, -1, 0, \ldots] \text{ – differential operator} \\ h^{int} &= [\ldots, 0, 0, 0, \mathbf{1}, \quad 1, 1, \ldots] \text{ – integral operator} \end{aligned}$$

– easy to see that $h^{diff} * h^{int} = h^I$ holds.

The trick with the integral image is in fact

$$f * h = f * h^I * h = f * h^{int} * h^{diff} * h = (f * h^{int}) * (h^{diff} * h)$$

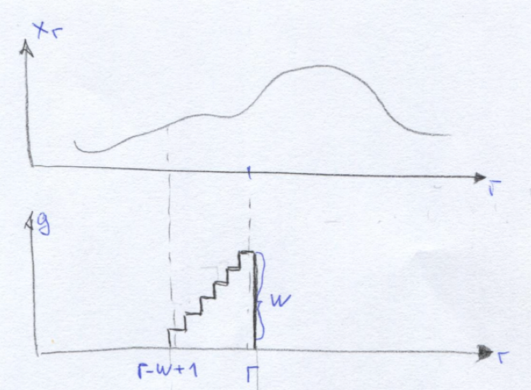
Consequences:

- $f * h^{int}$ is easy to compute (**linear** time complexity)
- $h * h^{diff}$ is **sparse** (a small number of non-zero elements)

A bit more complex example

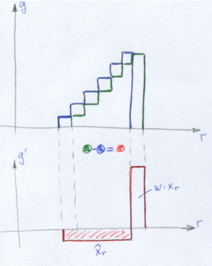
Convolve efficiently with the mask (1D)

$$g(i) = \sum_{i'=i-w+1}^{i} (w - i + i') \cdot f(i')$$



A bit more complex example

Consider, how $g(i+1)$ can be computed efficiently using $g(i)$



$$g(i{+}i) = \sum_{i'=i-w+2}^{i+1} (w{-}i{+}i'{-}1)\cdot f(i') =$$

$$= \sum_{i'=i-w+1}^{i} (w{-}i{+}i'{-}1)\cdot f(i') + w\cdot f(i{+}1) =$$

$$= g(i) - \sum_{i'=i-w+1}^{i} f(i') + w\cdot f(i{+}1) =$$

$$= g(i) - \hat{f}(i) + w\cdot f(i{+}1)$$

A bit more complex example

If $\hat{f}(i)$ is known (pre-computed in advance for all i), the next value could be computed in a **constant** time.

However, $\hat{f}(i)$ is nothing but a mean filter, i.e. it can be pre-computed in **linear** time !!!

The algorithm:

1. Compute the integral signal $\tilde{f}$
2. Compute the mean filter $\hat{f}$
3. Compute the output g from $\hat{f}$ and f

All steps have the linear complexity
$\rightarrow$ the overall time complexity is linear as well.

A bit more complex example

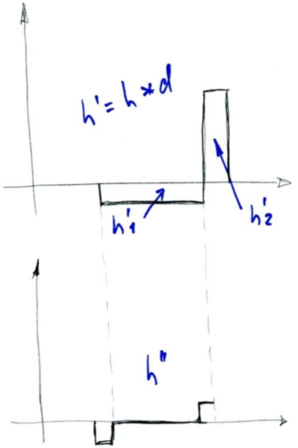
Explain it in terms of convolutions

$$g = f * h,$$

$$\begin{aligned} g * d &= f * (h * d) = f * h' = \\ &= f * (h'_1 + h'_2) = f * h'_1 + f * h'_2 \\ &= f * i * (d * h'_1) + f * h'_2 = \\ &= f * i * h'' + f * h'_2 \end{aligned}$$

$$\Downarrow$$

$$g = (f * i * h'' + f \cdot w) * i$$



d – differential operator
i – integral operator

Some interesting tasks

Implement convolution with (vertical) Sobel-kernel

-1	0	1
-2	0	2
-1	0	1

with only 3 additions per pixel and no multiplication.

Implement convolution with the exponential mask (for 1D)

$$g(i) = \tau \cdot \sum_{i'=-\infty}^{i} \exp(-\tau \cdot (i - i')) \cdot f(i')$$

with two multiplications and one addition per pixel.
Hint: simply consider, how $g(i+1)$ can be obtained from $g(i)$.

Some interesting tasks (possible assignments)

Let a convolution kernel be given.

1. Try to (automatically) find its representation using differentiation and integration in order to reduce the time complexity.
2. Try to approximate the mask by "the best possible" separable one (using SVD-decomposition of the kernel-matrix).
3. Decompose the kernel into a sum of separable filters. Study approximations.

Other task for linear filters: implement (efficiently)

$$
\begin{aligned}
g(i,j) &= \sum_{i',j'} \max\Big(0, C - \max(|i-i'|, |j-j'|)\Big) \\
g(i,j) &= \sum_{i',j'} \max\Big(0, C - (|i-i'| + |j-j'|)\Big) \\
&\;\;\ldots
\end{aligned}
$$

Morphological filters, fast minimum in 1D

$$g(i) = \min_{i'=i-w}^{i+w} f(i')$$

A naïve algorithm (according to the formula):
for each output value enumerate all inputs and take the minimal one. Time complexity: $O(nw)$

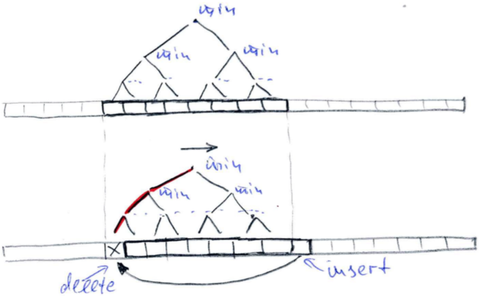
The idea:

1. Keep the **ordered** set of all values
2. For each output one elements should be inserted and one should me removed
3. Use data structure that allows to do it fast, i.e. $O(\log w)$

$\rightarrow$ the overall time complexity is $O(n \log w)$.

Fast minimum in 1D, 2D

Data structure for 8 elements in the set (example):



The number of operations for both insertion and removal is proportional to the tree depth, i.e. $O(\log w)$.

Min-filter in 2D is separable $\rightarrow$ the time complexity in 2D is $O(n \log w)$ as well !!!

Fast median

Let the set of values be ordered (as before) – so we can update it in $O(\log w)$ again. But how to pick the median (the middle element)? It requires $O(w^2)$:-((w is the window size).

Idea – keep **two** ordered sets: one for all elements that are smaller or equal to the actual median, the other for all elements that are greater or equal to the actual median. Keep the **maximum** value for the first set and the **minimum** value for the second one.

If the sets are of the same size, the actual maximum of the first set is the desired median.

Hint: a suitable container in C++ is "multiset".

Fast median

Consider e.g. the insertion of a new element:

1. Look, in which set the new element is to be inserted (compare it to the "$\max$"), insert it – it takes $O(\log w)$

2. If the sets have different sizes, **balance** them – transfer one element from the larger set to the other one. For example, remove the largest value from the first set and insert it into the second one (of course, update the corresponding $\max$ and $\min$). All operations take $O(\log w)$

$\rightarrow$ in 1D the overall time complexity is $O(n \log w)$. In 2D the overall time-complexity is $O(nw \log w)$, where w is the filter size (compare with $O(nw^2 \log w)$ for the naïve algorithm).

Assignments

1. Linear filters:
   1.1 Fast Guided filter (2P) – see the lecture
   1.2 Harris detector (up to 4P) – see the next lecture
   1.3 Approximations (see slide 17) (up to 4P)
   1.4 Some exotic filters (efficiently, see slide 17) (up to 4P)
   1.5 Something you like (?P)
2. Morphologic filters – $\min$, $\max$, for colors etc. (1-2P)
3. Fast median (with $O(nw \log w)$) (up to 3P)
4. Geometric median for RGB (up to 4P)
5. Evaluations are appreciated ($+$1P)

Delivery + Defense + Game rules

Defense: during exercise time slots (i.e. 1,2,8,9,15 Nov.), $\approx$ 5-10 min. per person $\rightarrow$ points

Delivery: per E-Mail an Dmytro.Shlezinger@tu-... or live during the defense, Sources (*.cpp, *.h, sufficiently commented !!!), input/output images; if applicable – *.pro, Makefile (compilation advises ...), comments, remarks, description, evaluation results (tables, diagrams etc.)

Game rules: There are four exercises. You can get 1P for the first one (past) and up to 4P for each of 2-4. Exercises are passed if:

1. There are 8P in total at the end of the semester **and**
2. There is at least 1P for each 2-4 exercises.

Additional remarks

- The programming language is C/C++
- Implement it by yourself, use e.g. OpenCV only to read/write (visualize) images
- Avoid platform-specific code
- Avoid GUI-s
- Avoid complex classes (data structures, templates etc.)
- Test your implementations *carefully* on *many* images