Representing Images as Functions

Image

In the continuous setting, an image I is a *function*

$$I : \Omega \to \mathbb{R}^n$$

Domain of the image

Image *domain* Ω is a rectangular subset of $\mathbb{R}^d$, $d \geq 1$
$\Omega \subset \mathbb{R}^1$: signal (1D)
$\Omega \subset \mathbb{R}^2$: image (2D)
$\Omega \subset \mathbb{R}^3$: volume (3D)

Range of the image

Image values lie in $\mathbb{R}^n$, $n \geq 1$, where n is the number of components:
$\mathbb{R}^1$: scalar valued image (grayscale)
$\mathbb{R}^2$: e.g. 2D-vector field
$\mathbb{R}^3$: e.g. RGB image, HSV values, 3D-vector field
$\mathbb{R}^4$: e.g. matrix valued images

Differential Operators on Images

We assume the image domain to be two-dimensional: $\Omega \to \mathbb{R}^2$.
Since images are functions, we can apply *differential operators* to them (assuming they are sufficiently smooth):

Partial derivative w.r.t. x of a scalar image $I : \Omega \to \mathbb{R}$

$$\partial_x I : \Omega \to \mathbb{R}, \quad (\partial_x I)(x, y) = \lim_{h \to 0} \frac{I(x + h, y) - I(x, y)}{h}$$

Partial derivative w.r.t. y of a scalar image $I : \Omega \to \mathbb{R}$

$$\partial_y I : \Omega \to \mathbb{R}, \quad (\partial_y I)(x, y) = \lim_{h \to 0} \frac{I(x, y + h) - I(x, y)}{h}$$

Derivatives for vector-valued images are defined component-wise

Differential Operators on Images

Gradient of a scalar image $I : \Omega \to \mathbb{R}$
The gradient combines all partial derivatives into a vector:

$$\nabla I : \Omega \to \mathbb{R}^2, \quad (\nabla I)(x,y) = \begin{pmatrix} (\partial_x I)(x,y) \\ (\partial_y I)(x,y) \end{pmatrix}$$

Divergence of a 2D-vector field $v : \Omega \to \mathbb{R}^2$
This one needs a vector field as input. The result is a scalar function:

$$\operatorname{div} v : \Omega \to \mathbb{R}, \quad (\operatorname{div} v)(x,y) = (\partial_x v_1)(x,y) + (\partial_y v_2)(x,y)$$

Differential Operators on Images

Gradient magnitude of a scalar image
Pointwise absolute value of ∇I:

$$|\nabla I| : \Omega \to \mathbb{R}, \quad (|\nabla I|)(x,y) = \sqrt{(\partial_x I)(x,y)^2 + (\partial_y I)(x,y)^2}$$

One often uses this as an edge detector, where big values of $|\nabla I|(x,y)$ indicate an edge at (x,y).

Differential Operators on Images

Laplacian of a scalar image $I : \Omega \to \mathbb{R}$
The gradient $\nabla I : \Omega \to \mathbb{R}^2$ is a 2D-vector field, and divergence div operates on 2D-vector fields. Thus, we can concatenate these two operators. The result is the *Laplacian*:

$$\Delta I : \Omega \to \mathbb{R}, \quad \Delta I := \operatorname{div}(\nabla I) = \operatorname{div}\begin{pmatrix}\partial_x I \\ \partial_y I\end{pmatrix}$$

$$(\Delta I)(x,y) = (\partial_{xx} I)(x,y) + (\partial_{yy} I)(x,y)$$

The laplacian is useful in *physical models*. For example, if the value $I(x,y)$ specifies the temperature at each point (x,y), then ΔI turns out to be the rate of local temperature decrease: $(\partial_t I)(x,y) = a(\Delta I)(x,y)$ for some $a > 0$.

Convolution

Convolution computes a weighted of image values.

Convolution

Given a *kernel* $K : \mathbb{R}^2 \to \mathbb{R}$ and an image $I : \Omega \to \mathbb{R}^n$:

$$K * I : \Omega \to \mathbb{R}^n, \quad (K * I)(x, y) = \int_{\mathbb{R}^2} K(a, b)\, I(a - x, b - y)\, da\, db$$

Definition at boundary of image domain
Special handling at (x, y) near the boundary of Ω is needed, as the argument $(a - x, b - y)$ of I may be outside of Ω.
Typical choices: Set to zero or Clamping.

2D-Gaussian kernel

$$K(a, b) = G_\sigma(a, b) := \frac{1}{2\pi\sigma^2}\, e^{-\frac{a^2+b^2}{2\sigma^2}}$$

with a standard deviation $\sigma > 0$.

Convolution: Properties

Assuming the image has been prolonged also beyond Ω to whole $\mathbb{R}^2$.

- Commutativity:
$$K * I = I * K$$
- Associativity:
$$K_1 * (K_2 * I) = (K_1 * K_2) * I$$
- Bilinearity:
$$(\alpha_1 K_1 + \alpha_2 K_2) * I = \alpha_1 (K_1 * I) + \alpha_2 (K_2 * I)$$
$$K * (\beta_1 I_1 + \beta_2 I_2) = \beta_1 (K * I_1) + \beta_2 (K * I_2)$$
for $\alpha_2, \alpha_2 \in \mathbb{R}$, $\beta_2, \beta_2 \in \mathbb{R}$.
- Differential operators:
$$\partial_x (K * I) = (\partial_x K) * I = K * (\partial_x I)$$
$$\partial_y (K * I) = (\partial_y K) * I = K * (\partial_y I)$$

Discretization of Scalar Images

For the numeral implementations, the image domain $\Omega \subset \mathbb{R}^2$ is *discretized* into a two-dimensional grid of $M \times N$ pixels.

Scalar valued images $I : \Omega \to \mathbb{R}$

We have one real number at each pixel: $I(x, y)$, $0 \leq x \leq M-1$, $0 \leq y \leq N-1$. This can be represented as matrices $\mathbb{R}^{M \times N}$.

Linearized storage

For computing purposes, the 2D-grid is linearized into a one-dimensional array $a \in \mathbb{R}^{MN}$ of size MN. Usual convention is row-major storage:

$$
\begin{aligned}
a = \Big(& I(0,0),\ I(1,0),\ I(2,0),\ \ldots,\ I(M-1,0), \\
& I(0,1),\ I(1,1),\ I(2,1),\ \ldots,\ I(M-1,1), \quad \ldots, \\
& I(0,N-1),\ I(1,N-1),\ I(2,N-1),\ \ldots,\ I(M-1,N-1) \Big).
\end{aligned}
$$

Linearized access to image values

Pixel (x, y) corresponds to the linearized index is $i = x + M \cdot y$:

$$I(x, y) = a[x + M \cdot y]$$

Discretization of Vector-Valued Images

2D-vector fields $v : \Omega \to \mathbb{R}^2$
We have *two* real numbers at each pixel: $v(x, y) = (v_1(x, y), v_2(x, y))$.
This can be represented two matrices $\mathbb{R}^{M \times N}$.

Linearized storage, component-wise
There are two common options for linearizing v. One is to first store v_1, and then v_2:

$$a = \Big(v_1(0, 0), \ldots, v_1(M-1, N-1), v_2(0, 0), \ldots, v_2(M-1, N-1)\Big).$$

Linearized (component-wise) access to image values
Values $v_1(x, y), v_2(x, y)$ are at locations $(x + M \cdot y) + k \cdot MN$, $k = 0, 1$.

Discretization of Vector-Valued Images

2D-vector fields $v : \Omega \to \mathbb{R}^2$
We have *two* real numbers at each pixel: $v(x, y) = (v_1(x, y), v_2(x, y))$.
This can be represented two matrices $\mathbb{R}^{M \times N}$.

Linearized storage, interlieved
Interlieved storage stores v_1 and v_2 pixel-wise:

$$a = \Big(v_1(0,0), v_2(0,0), v_1(1,0), v_2(1,0), \dots, v_1(M-1,0), v_2(M-1,0) \\ \dots, v_1(0,N-1), v_2(0,N-1), \dots, v_2(M-1,N-1)\Big).$$

Linearized (interlieved) access to image values
Values $v_1(x, y), v_2(x, y)$ are at locations $2(x + M \cdot y) + k,\ k = 0, 1$.

Discretization of Differential Operators

Gradient

Forward differences:

$$(\nabla I)(x,y) = \begin{pmatrix} (\partial_x^+ I)(x,y) \\ (\partial_y^+ I)(x,y) \end{pmatrix}$$

Forward differences (with Neumann boundary conditions)

$$(\partial_x^+ I)(x,y) = \begin{cases} I(x+1,y) - I(x,y) & \text{if } x+1 < M \\ 0 & \text{else} \end{cases}$$

$$(\partial_y^+ I)(x,y) = \begin{cases} I(x,y+1) - I(x,y) & \text{if } y+1 < N \\ 0 & \text{else} \end{cases}$$

This assumes I to have slope 0 at the boundary: $\partial_{\text{normal}_\Omega} I = 0$.

Discretization of Differential Operators

Divergence
Backward differences:

$$(\mathrm{div}\, v)(x,y) = (\partial_x^- v_1)(x,y) + (\partial_y^- v_2)(x,y)$$

Backward differences (with Dirichlet boundary conditions)

$$(\partial_x^- I)(x,y) = \left\{\begin{matrix} I(x,y) & \text{if } x+1 < M \\ 0 & \text{else} \end{matrix}\right\} - \left\{\begin{matrix} I(x-1,y) & \text{if } x > 0 \\ 0 & \text{else} \end{matrix}\right\}$$

$$(\partial_y^- I)(x,y) = \left\{\begin{matrix} I(x,y) & \text{if } y+1 < N \\ 0 & \text{else} \end{matrix}\right\} - \left\{\begin{matrix} I(x,y-1) & \text{if } y > 0 \\ 0 & \text{else} \end{matrix}\right\}$$

This assumes I to have zero values at the boundary.

Discretization of the Convolution

Convolution

$$(K * I)(x, y) = \int_{\mathbb{R}^2} K(a, b)\, I(a - x, b - y)\, da\, db$$

Discretized convolution
The convolution is discretized as a finite weighted sum in each pixel:

$$(K * I)(x, y) = \sum_{(a,b)\in S_K} K(a, b) \cdot I(x - a, y - b)$$

where S_K is the support of K, i.e. the positions (a, b) with $K(a, b) \neq 0$.

Discretization of the Convolution

Convolution

$$(K * I)(x, y) = \int_{\mathbb{R}^2} K(a, b)\, I(a - x, b - y)\, da\, db$$

Discretized kernel
In computer vision one often deals with the convolution with a small-support kernel K. An examples is the *Gaussian kernel*.

Windowing
The support is assumed to be in a small *window* of size $(2r_x + 1) \times (2r_y + 1)$ with radii $r_x \geq 1$, $r_y \geq 1$, which is symmetric w.r.t. zero:

$$(K * I)(x, y) = \sum_{a=-r_x}^{r_x} \sum_{b=-r_y}^{r_y} K(a, b)\, I(a - x, b - y)\, da\, db$$

For storing the discretized kernel K, the same row-major approach is used.