Visual Computing: Convolution and Filtering

Prof. Otmar Hilliges
Prof. Markus Gross
Last week Segmentation

Fast & Accurate ?
Graph Cuts
Boykov and Jolly (2001)

**Image**

Foreground (source)

Background (sink)

**Cut:** separating source and sink; **Energy:** collection of edges

**Min Cut:** Global minimal energy in polynomial time

Slides from Carsten Rother (MSR)
Visual Computing: Convolution and Filtering

Prof. Otmar Hilliges
Prof. Markus Gross
What is image filtering?

- Modify the pixels in an image based on some function of a local neighborhood of the pixels.
Linear Shift-Invariant Filtering

• About modifying pixels based on neighborhood. Local methods simplest.
• Linear means linear combination of neighbors. Linear methods simplest.
• Shift-invariant means doing the same for each pixel. Same for all is simplest.
• Useful to:
  – Low-level image processing operations
  – Smoothing and noise reduction.
  – Sharpen.
  – Detect or enhance features.
Linear Filtering

- $L$ is linear map/transform if

$$L[\alpha l_1 + \beta l_2] = \alpha L[l_1] + \beta L[l_2]$$
Linear Operations: Weighted Sum

- Output $I'$ of linear image operation is a weighted sum of each pixel in the input $I$

$$I'_j = \sum_{i=1}^{N} \alpha_{ij} I_i, \ j = 1...N$$

(note: $N=\text{wxh}$)
Linear Filtering

- Linear operations can be written:

\[
I'(x, y) = \sum_{(i, j) \in N(x, y)} K(x, y; i, j)I(x + i, x + j)
\]

- \(I\) is the input image; \(I'\) is the output of the operation.
- \(k\) is the *kernel* of the operation. \(N(m,n)\) is a neighbourhood of \((m,n)\).
Linear Filtering

- Linear operations can be written:

\[ I'(x, y) = \sum_{(i, j) \in N(x, y)} K(x, y; i, j) I(x + i, x + j) \]

- \( I \) is the input image; \( I' \) is the output of the operation.

- \( k \) is the kernel of the operation.

- \( N(m, n) \) is a neighbourhood of \((m, n)\).

Operations are “shift-invariant” if \( k \) does NOT depend on \((x, y)\): using same weights everywhere!
Correlation
(e.g. template matching)

\[
o(i,j) = c_{11} I(i-1,j-1) + c_{12} I(i-1,j) + c_{13} I(i-1,j+1) + \\
c_{21} I(i,j-1) + c_{22} I(i,j) + c_{23} I(i,j+1) + \\
c_{31} I(i+1,j-1) + c_{32} I(i+1,j) + c_{33} I(i+1,j+1)
\]
Correlation

• Linear operation of correlation:

\[ I' = K \circ I \]

\[ I'(x, y) = \sum_{(i, j) \in N(x, y)} K(i, j)I(x+i, y+j) \]

• Represent the linear weights as an image, \( K \)
**Convolution**
(e.g. point spread function)

\[ I'(x,y) = K(1,1)I(x-1,y-1) + K(0,1)I(x,y-1) + K(-1,1)I(x+1,y-1) + K(1,0)I(x-1,y) + K(0,0)I(x,y) + K(-1,0)I(x+1,y) + K(1,-1)I(x-1,y+1) + K(0,-1)I(x-1,y) + K(-1,-1)I(x+1,y+1) \]
Convolution

- Linear operation of convolution:

\[ I' = K \ast I \]

\[ I'(x, y) = \sum_{(i, j) \in N(x, y)} K(i, j)I(x - i, y - j) \]

- Represent the linear weights as an image, \( K \)
- Same as correlation, but with kernel reversed
Correlation

\[ I'(x, y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(i, j)I(x + i, y + j) \]

Convolution

\[ I'(x, y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(i, j)I(x - i, y - j) \]

\[ = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(-i, -j)I(x + i, y + j) \]

So if \( K(i,j) = K(-i, -j) \), then Correlation == Convolution
Linear Filtering
(warm-up)

Original

Slide credit: D.A. Forsyth
Linear Filtering (warm-up)

Original

Filtered (no change)

Slide credit: D.A. Forsyth
Linear Filtering

Original

(use convolution)
Linear Filtering

Original

(Use convolution)

Shifted left
By 1 pixel

Slide credit: D.A. Forsyth
Linear Filtering

Original

?
Linear Filtering

Original

1 1 1
1 1 1
1 1 1
Linear Filtering

Original

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[\frac{1}{9}\]
Linear Filtering

Original

Blur (with a box filter)

Slide credit: D.A. Forsyth
Linear Filtering

Original

(Note that filter sums to 1)
Linear Filtering

Original

Sharpening filter
- Accentuates differences with local average

Slide credit: D.A. Forsyth
Sharpening

before

after
Correlation
(e.g. Template-matching)

\[ I' = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(i, j)I(x+i, y+j) \]

(matlab default)

Convolution
(e.g. point spread function)

\[ I' = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(i, j)I(x-i, y-j) \]
Example

\[ K = \text{ones}(9,9); \]
\[ I2 = \text{conv2}(I,K); \]
Example

\[ K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \]
Yucky details

• What about near the edge?
  – the filter window falls off the edge of the image
  – need to extrapolate
  – methods:
    • clip filter (black)
    • wrap around
    • copy edge
    • reflect across edge
    • vary filter near edge
Separable Kernels

• Separable filters can be written

\[ K(m,n) = f(m)g(n) \]

• For a rectangular neighbourhood with size (2M+1)x(2N+1),

\[ I'(m,n) = f \ast (g \ast I(N(m,n))) \]

\[ I''(m,n) = \sum_{j=-N}^{N} g(j)I(m,n-j) \]

\[ I'(m,n) = \sum_{i=-M}^{M} f(i)I''(m-i,n) \]

computational advantage?
Smoothing Kernels
(Low-pass filters)

Mean filter:
\[
\frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

Weighted smoothing filters:

\[
\frac{1}{10} \begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

\[
\frac{1}{16} \begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1
\end{bmatrix}
\]
Gaussian Kernel

• Idea: Weight contributions of neighboring pixels by nearness

\[ G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \]

5 × 5, \( \sigma = 1 \)

• Constant factor at front makes volume sum to 1
Smoothing with a Gaussian

Slide credit: D.A. Forsyth
Smoothing with a box filter
Gaussian Smoothing Kernels

\[ g(x, y) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{(x^2 + y^2)}{2\sigma^2}\right] \]

\[ = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{x^2}{2\sigma^2}\right] \quad \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{y^2}{2\sigma^2}\right] \]

\[ = g(x) g(y) \]

Separable!
Gaussian Smoothing Kernels

- Amount of smoothing depends on $\sigma$ and window size.
- Width > 3$\sigma$

| $7\times7; \sigma = 1.$ | $7\times7; \sigma = 9.$ | $19\times19; \sigma = 1.$ | $19\times19; \sigma = 9.$ |
Gaussian Smoothing Kernel Top-5

1. Rotationally symmetric
2. Has a single lobe
   - Neighbor’s influence decreases monotonically
3. Still one lobe in frequency domain
   - No corruption from high frequencies
4. Simple relationship to $\sigma$
5. Easy to implement efficiently
Differential Filters

Prewitt operator:

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}
\]

Sobel operator:

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{bmatrix}
\]
High-pass filters

Laplacian operator:
\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\]

High-pass filter:
\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
High-pass filters

Laplacian

High pass
Differentiation and convolution

• Recall, for 2D function, $f(x,y)$:

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left( \frac{f(x + \varepsilon, y) - f(x, y)}{\varepsilon} \right)$$

• This is linear and shift invariant, so must be the result of a convolution.

• We could approximate this as

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

(which is obviously a convolution)

-1 1
Vertical gradients from finite differences
Filters are templates

- Filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products
- Filters look like the effects they are intended to find
- Filters find effects they look like
Image Sharpening

• Also known as Enhancement
• Increases the high frequency components to enhance edges.
• $I' = I + \alpha|k*I|$, where $k$ is a high-pass filter kernel and $\alpha$ is a scalar in $[0,1]$. 
Sharpening Example

original

\[ \alpha = 0.5 \]
Digression: Image Classification & Segmentation

- Dendrites: Impulses carried toward cell body
- Nucleus
- Axon: Impulses carried away from cell body
- Axon terminals
- Branches of axon
An engineer's view of a neuron

```
class Neuron:
    # ...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
        return firing_rate
```
Fully Connected Neural Networks
Hubel & Wiesel,
1959
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX
1962
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX
Receptive fields of single neurones in the cat's striate cortex.

Hierarchical Receptive Fields

Hubel & Weisel

topographical mapping

featural hierarchy

hyper-complex cells

complex cells

simple cells

high level

mid level

low level
S (simple) cells are tuned to specific stimuli and have typically small receptive fields.

C (complex) cells combine output from various S units to increase invariance and receptive field.

Many iterations of these operations allow for the construction of complex objects from low-level features.

[Adopted from Serre et al. PNAS 2007]
Neurocognitron

[Fukushima 1980]
LeNet

[LeCun, Bottou, Bengio, Haffner 1998.]
ImageNet
Convolution Layer

32x32x3 image

32 \text{ height}

32 \text{ width}

3 \text{ depth}
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image
5x5x3 filter $w$

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$w^T x + b$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

Consider a second, green filter.

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

![Diagram of ConvNet layers](Image)

- **CONV, ReLU**
- e.g. 6 5x5x3 filters
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

- **First Layer:**
  - Input: 3x3x3 
  - Convolutional Layer (CONV): 6 5x5x3 filters
  - ReLU

- **Second Layer:**
  - Input: 6x28x28
  - Convolutional Layer (CONV): 10 5x5x6 filters
  - ReLU

- **Third Layer:**
  - Input: 10x24x24
  - Convolutional Layer (CONV): 24
  - ReLU

- **Remaining Layers:**
  - Convolutional Layer (CONV), ReLU
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[From recent Yann LeCun slides]
Preview

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[From recent Yann LeCun slides]
Preview: Object detection
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation

Single object

Multiple objects

CAT

CAT

CAT, DOG, DUCK

CAT, DOG, DUCK
Classification + Localization: Task

**Classification:** C classes
- **Input:** Image
- **Output:** Class label
- **Evaluation metric:** Accuracy

**Localization:**
- **Input:** Image
- **Output:** Box in the image (x, y, w, h)
- **Evaluation metric:** Intersection over Union

**Classification + Localization:** Do both
Idea #1: Localization as Regression

**Input:** image

Only one object, simpler than detection

**Neural Net**

**Output:**

Box coordinates (4 numbers)

**Correct output:**

Box coordinates (4 numbers)

**Loss:**

L2 distance
Simple Recipe for Classification + Localization

**Step 1:** Train (or download) a classification model (AlexNet, VGG, GoogLeNet)
Step 2: Attach new fully-connected “regression head” to the network
Simple Recipe for Classification + Localization

**Step 3**: Train the regression head only with SGD and L2 loss
Simple Recipe for Classification + Localization

**Step 4**: At test time use both heads

![Image convolution and pooling](image)

- **Convolution and Pooling**
  - Final conv feature map

- **Fully-connected layers**
  - Class scores
  - Box coordinates
Per-class vs class agnostic regression

Assume classification over C classes:

Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)
Where to attach the regression head?

After conv layers: Overfeat, VGG

After last FC layer: DeepPose, R-CNN

Convolution and Pooling

Final conv feature map

Fully-connected layers

Class scores

Softmax loss
Thursday: Image Features