Lecture:
Image manipulations (2)
Filters and kernels

Alireza Akhavan Pour

CLASS.VISION
Image Manipulations (Part 2)

• 1. Intro to Convolution
• 2. Blurring
• 3. Sharpening
• 4. Dilation, Erosion, Opening and Closing
• 5. Edge Detection & Image Gradients
Filter example #1: Moving Average

2D DS moving average over a $3 \times 3$ window of neighborhood

$$g[n, m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k, l]$$

$$= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n - k, m - l]$$
Filter example #1: Moving Average

\[ F[x, y] \]

\[ G[x, y] \]

\[(f * h)[m, n] = \sum_{k,l} f[k, l] h[m - k, n - l]\]
Filter example #1: Moving Average

\[
F[x, y] \quad G[x, y]
\]

\[
(f * h)[m, n] = \sum_{k,l} f[k, l] h[m-k, n-l]
\]
Filter example #1: Moving Average

\[
F[x, y] = (f * h)[m, n] = \sum_{k,l} f[k, l] h[m-k, n-l]
\]

\[
G[x, y]
\]
Filter example #1: Moving Average

\[ F[x, y] \]

\[ G[x, y] \]

\[
(f * h)[m, n] = \sum_{k,l} f[k, l] h[m - k, n - l]
\]
\[ F[x, y] \]

\[ G[x, y] \]

\[(f \ast h)[m,n] = f[k,l] h[m-k, n-l] \]
Filter example #1: Moving Average

\[ F[x, y] \]

\[ G[x, y] \]

\[
(f * h)[m, n] = \sum_{k,l} f[k, l] h[m-k, n-l]
\]

Source: S. Seitz
Filter example #1: Moving Average

In summary:

• This filter “Replaces” each pixel with an average of its neighborhood.

• Achieve smoothing effect (remove sharp features)

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

\[\frac{1}{9}\]
Filter example #1: Moving Average
2D convolution

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] \]

Assume we have a filter \((h[\cdot, \cdot])\) that is 3x3. and an image \((f[\cdot, \cdot])\) that is 7x7.
2D convolution

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] \]

Assume we have a filter \((h[\cdot, \cdot])\) that is 3x3. and an image \((f[\cdot, \cdot])\) that is 7x7.
2D convolution

\[ f[n, m] * h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] \]

Assume we have a filter \( h[] \) that is 3x3. and an image \( f[] \) that is 7x7.
2D convolution

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \cdot h[n - k, m - l] \]

Assume we have a filter \( h[\cdot, \cdot] \) that is 3x3, and an image \( f[\cdot, \cdot] \) that is 7x7.
2D convolution

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n-k, m-l] \]

Assume we have a filter \( h[\cdot, \cdot] \) that is 3x3. and an image \( f[\cdot, \cdot] \) that is 7x7.
2D convolution

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \ h[n - k, m - l] \]

Assume we have a filter(h[,] that is 3x3. and an image (f[,] that is 7x7.
2D convolution example

Input

Kernel

Output

D convolution example
2D convolution example

\[\begin{array}{ccc}
0 & 1 & 0 \\
-2 & 1 & 5 \\
4 & 0 & 3 \\
\end{array}\]

\[\begin{align*}
&= x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
&\quad + x[-1,0] \cdot h[1,0] + x[0,0] \cdot h[0,0] + x[1,0] \cdot h[-1,0] \\
&\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
&= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) = -13
\end{align*}\]

Slide credit: Song Ho Ahn
2D convolution example

\[
x[0,-1] \cdot h[1,1] + x[1,-1] \cdot h[0,1] + x[2,-1] \cdot h[-1,1] \\
+ x[0,0] \cdot h[1,0] + x[1,0] \cdot h[0,0] + x[2,0] \cdot h[-1,0] \\
+ x[0,1] \cdot h[1,-1] + x[1,1] \cdot h[0,-1] + x[2,1] \cdot h[-1,-1]
\]

\[
= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot (-1) + 5 \cdot (-2) + 6 \cdot (-1) = -20
\]
2D convolution example

\[
\begin{align*}
&= x[1,-1] \cdot h[1,1] + x[2,-1] \cdot h[0,1] + x[3,-1] \cdot h[-1,1] \\
&\quad + x[1,0] \cdot h[1,0] + x[2,0] \cdot h[0,0] + x[3,0] \cdot h[-1,0] \\
&\quad + x[1,1] \cdot h[1,-1] + x[2,1] \cdot h[0,-1] + x[3,1] \cdot h[-1,-1] \\
&= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 + 0 \cdot 0 + 5 \cdot (-1) + 6 \cdot (-2) + 0 \cdot (-1) = -17
\end{align*}
\]
\[
\begin{align*}
&= x[-1,0] \cdot h[1,1] + x[0,0] \cdot h[0,1] + x[1,0] \cdot h[-1,1] \\
&\quad + x[-1,1] \cdot h[1,0] + x[0,1] \cdot h[0,0] + x[1,1] \cdot h[-1,0] \\
&\quad + x[-1,2] \cdot h[1,-1] + x[0,2] \cdot h[0,-1] + x[1,2] \cdot h[-1,-1] \\
&= 0 \cdot 1 + 1 \cdot 2 + 2 \cdot 1 + 0 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 0 \cdot (-1) + 7 \cdot (-2) + 8 \cdot (-1) = -18
\end{align*}
\]
$$
\begin{align*}
&= x[0,0] \cdot h[1,1] + x[1,0] \cdot h[0,1] + x[2,0] \cdot h[-1,1] \\
&\quad + x[0,1] \cdot h[1,0] + x[1,1] \cdot h[0,0] + x[2,1] \cdot h[-1,0] \\
&\quad + x[0,2] \cdot h[1,-1] + x[1,2] \cdot h[0,-1] + x[2,2] \cdot h[-1,-1] \\
&= 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 1 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 + 7 \cdot (-1) + 8 \cdot (-2) + 9 \cdot (-1) = -24
\end{align*}
$$
2D convolution example

\[ \begin{align*}
&= x[1,0] \cdot h[1,1] + x[2,0] \cdot h[0,1] + x[3,0] \cdot h[-1,1] \\
&+ x[1,1] \cdot h[1,0] + x[2,1] \cdot h[0,0] + x[3,1] \cdot h[-1,0] \\
&+ x[1,2] \cdot h[1,-1] + x[2,2] \cdot h[0,-1] + x[3,2] \cdot h[-1,-1] \\
&= 2 \cdot 1 + 3 \cdot 2 + 0 \cdot 1 + 5 \cdot 0 + 6 \cdot 0 + 0 \cdot 0 + 8 \cdot (-1) + 9 \cdot (-2) + 0 \cdot (-1) = -18
\end{align*} \]
Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\* 

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[=\]

[?]
Convolution in 2D - examples

Original

* 0 0 0
   0 1 0
   0 0 0

Filtered (no change)

Courtesy of D Lowe
Convolution in 2D - examples

Original

\[
\begin{array}{ccc}
0 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[\ast\]

\[=\]

\[?\]
Convolution in 2D - examples

Original

Shifted right
By 1 pixel

*Courtesy of D Lowe
\[ \begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array} \]

\[ \times \frac{1}{9} \]

= ?
Convolution in 2D - examples

Original

Blur (with a box filter)

\[ \star \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \]
Original

(Note that filter sums to 1)

“details of the image”

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

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

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

= ?

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{array}
\] \[- \frac{1}{9} \]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
• What does blurring take away?

- original

smoothed (5x5)

= detail

- Let’s add it back:

original

detail

= sharpened
Convolution in 2D – Sharpening filter

Sharpening filter: Accentuates differences with local average

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

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

\[
\frac{1}{9}
\]
Image support and edge effect

• A computer will only convolve **finite support signals**.
• What happens at the edge?

- zero “padding”
- edge value replication
- mirror extension
- more (beyond the scope of this class)

-> Matlab conv2 uses zero-padding
Convolution vs. (Cross) Correlation

• A **convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function.
  – convolution is a filtering operation

• **Correlation** compares the *similarity of two sets of data*. Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best.
  – correlation is a measure of relatedness of two signals
What about **OpenCV**?
Convolution or (Cross) Correlation?

The function does actually compute correlation, not the convolution:

\[
dst(x,y) = \sum_{0 \leq x',y' < \text{kernel.cols}, \ 0 \leq y' < \text{kernel.rows}} \text{kernel}(x',y') \times \text{src}(x+x'-\text{anchor.x},y+y'-\text{anchor.y})
\]

That is, the kernel is not mirrored around the anchor point. If you need a real convolution, flip the kernel using \text{flip()}\ and set the new anchor to \( (\text{kernel.cols} - \text{anchor.x} - 1, \ \text{kernel.rows} - \text{anchor.y} - 1) \).

http://docs.opencv.org/modules/imgproc/doc/filtering.html#filter2d
<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Image result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 \end{bmatrix}$</td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Image Result" /></td>
</tr>
</tbody>
</table>

https://en.wikipedia.org/wiki/Kernel_(image_processing)
<table>
<thead>
<tr>
<th>Sharpen</th>
<th>$\begin{bmatrix} 0 &amp; -1 &amp; 0 \ -1 &amp; 5 &amp; -1 \ 0 &amp; -1 &amp; 0 \end{bmatrix}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Box blur</strong></td>
<td>$\frac{1}{9} \begin{bmatrix} 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

(normalized)
<table>
<thead>
<tr>
<th><strong>Gaussian blur 3 × 3</strong> (approximation)</th>
<th><img src="image" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \frac{1}{16} \begin{bmatrix} 1 &amp; 2 &amp; 1 \ 2 &amp; 4 &amp; 2 \ 1 &amp; 2 &amp; 1 \end{bmatrix} ]</td>
<td><img src="image" alt="Image" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Gaussian blur 5 × 5</strong> (approximation)</th>
<th><img src="image" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \frac{1}{256} \begin{bmatrix} 1 &amp; 4 &amp; 6 &amp; 4 &amp; 1 \ 4 &amp; 16 &amp; 24 &amp; 16 &amp; 4 \ 6 &amp; 24 &amp; 36 &amp; 24 &amp; 6 \ 4 &amp; 16 &amp; 24 &amp; 16 &amp; 4 \ 1 &amp; 4 &amp; 6 &amp; 4 &amp; 1 \end{bmatrix} ]</td>
<td><img src="image" alt="Image" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Unsharp masking 5 × 5</strong> (with no image mask)</th>
<th><img src="image" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \frac{-1}{256} \begin{bmatrix} 1 &amp; 4 &amp; 6 &amp; 4 &amp; 1 \ 4 &amp; 16 &amp; 24 &amp; 16 &amp; 4 \ 6 &amp; 24 &amp; -476 &amp; 24 &amp; 6 \ 4 &amp; 16 &amp; 24 &amp; 16 &amp; 4 \ 1 &amp; 4 &amp; 6 &amp; 4 &amp; 1 \end{bmatrix} ]</td>
<td><img src="image" alt="Image" /></td>
</tr>
</tbody>
</table>
Blurring is an operation where we average the pixels within a region (kernel).

$$Kernel_{\_} = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The above is a 5 x 5 kernel.

We multiply by 1/25 to normalize i.e. sum to 1, otherwise we’d be increasing intensity.

```python
cv2.filter2D(image, -1, kernel)
```

5 x 5 Kernel over our image
Other Types of Blurring

- **cv2.blur** – Averages values over a specified window
- **cv2.GaussianBlur** – Similar, but uses a Gaussian window (more emphasis or weighting on points around the center)
- **cv2.medianBlur** – Uses median of all elements in the window
- **cv2.bilateralFilter** – Blur while keeping edges sharp (slower). It also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The pixel difference function makes sure only those pixels with similar intensity to central pixel is considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.
Sharpening

Sharpening is the opposite of blurring, it strengthens or emphasizing edges in an image.

\[
\text{Kernel} = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 9 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]

Our kernel matrix sums to one, so there is no need to normalize (i.e. multiply by a factor to retain the same brightness of the original).
منابع

• http://vision.stanford.edu/teaching/cs131_fall1718/
• https://en.wikipedia.org/wiki/Kernel_(image_processing)
• https://www.udemy.com/master-computer-vision-with-opencv-in-python/
• http://docs.opencv.org/