Limitations of Point Operations

- They don’t know where they are in an image
- They don’t know anything about their neighbors
- Most image features (edges, textures, etc) involve a spatial neighborhood of pixels
- If we want to enhance or manipulate these features, we need to go beyond point operations
What Point Operations Can’t Do

Blurring/Smoothing
What Point Operations Can’t Do

Sharpening
What Point Operations Can’t Do

Weird Stuff
Spatial Filters

**Definition**

A **spatial filter** is an image operation where each pixel value $I(u, v)$ is changed by a function of the intensities of pixels in a neighborhood of $(u, v)$. 

![Diagram](image.png)
Example: The Mean of a Neighborhood

Consider taking the mean in a $3 \times 3$ neighborhood:

$$I'(u, v) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u + i, v + j)$$
How a Linear Spatial Filter Works

$H$ is the filter “kernel” or “matrix”

For the neighborhood mean:  

$$H(i,j) = \frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 
\end{bmatrix}$$
General Filter Equation

Notice that the kernel $H$ is just a small image!

Let $H : R_H \rightarrow [0, K - 1]$

$$I'(u, v) = \sum_{(i,j) \in R_H} I(u + i, v + j) \cdot H(i, j)$$

This is known as a correlation of $I$ and $H$
What Does This Filter Do?

Identity function (leaves image alone)
What Does This Filter Do?

Mean (averages neighborhood)
What Does This Filter Do?

Shift left by one pixel
What Does This Filter Do?

Sharpen (identity minus mean filter)
Filter Normalization

- Notice that all of our filter examples sum up to one.
- Multiplying all entries in $H$ by a constant will cause the image to be multiplied by that constant.
- To keep the overall brightness constant, we need $H$ to sum to one.

\[
I'(u, v) = \sum_{i, j} I(u + i, v + j) \cdot (cH(i, j))
\]

\[
= c \sum_{i, j} I(u + i, v + j) \cdot H(i, j)
\]
Effect of Filter Size

Mean Filters:

Original    7 × 7    15 × 15    41 × 41
What To Do At The Boundary?
What To Do At The Boundary?

- Crop
What To Do At The Boundary?

- Crop
- Pad
What To Do At The Boundary?

- Crop
- Pad
- Extend
What To Do At The Boundary?

- Crop
- Pad
- Extend
- Wrap
Convolution

**Definition**

Convolution of an image $I$ by a kernel $H$ is given by

$$I'(u, v) = \sum_{(i,j) \in R_H} I(u - i, v - j) \cdot H(i, j)$$

This is denoted: $I' = I \ast H$

- Notice this is the same as correlation with $H$, but with negative signs on the $I$ indices
- Equivalent to vertical and horizontal flipping of $H$:

$$I'(u, v) = \sum_{(-i,-j) \in R_H} I(u + i, v + j) \cdot H(-i, -j)$$
Linear Operators

**Definition**

A **linear operator** $F$ on an image is a mapping from one image to another, $I' = F(I)$, that satisfies:

1. $F(cI) = cF(I)$,
2. $F(I_1 + I_2) = F(I_1) + F(I_2)$,

where $I, I_1, I_2$ are images, and $c$ is a constant.

Both correlation and convolution are linear operators.
Infinite Image Domains

Let’s define our image and kernel domains to be infinite:

$$\Omega = \mathbb{Z} \times \mathbb{Z}$$

Remember \( \mathbb{Z} = \{ \ldots, -2, -1, 0, 1, 2, \ldots \} \)

Now convolution is an infinite sum:

$$I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u - i, v - j) \cdot H(i, j)$$

This is denoted \( I' = I \ast H \).
The infinite image domain $\Omega = \mathbb{Z} \times \mathbb{Z}$ is just a trick to make the theory of convolution work out.

We can still imagine that the image is defined on a bounded (finite) domain, $[0, w] \times [0, h]$, and is set to zero outside of this.
Properties of Convolution

Commutativity:

\[ I \ast H = H \ast I \]

This means that we can think of the image as the kernel and the kernel as the image and get the same result.

In other words, we can leave the image fixed and slide the kernel or leave the kernel fixed and slide the image.
Properties of Convolution

Associativity:

\[(I * H_1) * H_2 = I * (H_1 * H_2)\]

This means that we can apply \(H_1\) to \(I\) followed by \(H_2\), or we can convolve the kernels \(H_2 * H_1\) and then apply the resulting kernel to \(I\).
Properties of Convolution

**Linearity:**

\[(a \cdot I) \ast H = a \cdot (I \ast H)\]

\[(I_1 + I_2) \ast H = (I_1 \ast H) + (I_2 \ast H)\]

This means that we can multiply an image by a constant before or after convolution, and we can add two images before or after convolution and get the same results.
Properties of Convolution

**Shift-Invariance:**
Let $S$ be the operator that shifts an image $I$:

$$S(I)(u, v) = I(u + a, v + b)$$

Then

$$S(I * H) = S(I) * H$$

This means that we can convolve $I$ and $H$ and then shift the result, or we can shift $I$ and then convolve it with $H$. 
Properties of Convolution

**Theorem:** The only shift-invariant, linear operators on images are convolutions.
If my image $I$ has size $M \times N$ and my kernel $H$ has size $(2R + 1) \times (2R + 1)$, then what is the complexity of convolution?

$$I'(u, v) = \sum_{i=-R}^{R} \sum_{j=-R}^{R} I(u - i, v - j) \cdot H(i, j)$$

**Answer:** $O(MN(2R + 1)(2R + 1)) = O(MNR^2)$. Or, if we consider the image size fixed, $O(R^2)$. 
Which is More Expensive?

The following both shift the image 10 pixels to the left:

1. Convolve with a $21 \times 21$ shift operator (all zeros with a 1 on the right edge)
2. Repeatedly convolve with a $3 \times 3$ shift operator 10 times

The first method requires $21^2 \cdot wh = 441 \cdot wh$.
The second method requires $(9 \cdot wh) \cdot 10 = 90 \cdot wh$. 
Separability

**Definition**

A kernel $H$ is called **separable** if it can be broken down into the convolution of two kernels:

$$H = H_1 \ast H_2$$

More generally, we might have:

$$H = H_1 \ast H_2 \ast \cdots \ast H_n$$

**Example:** The “shift by ten” kernel is 10 copies of the “shift by one” kernel convolved together.
Remember the associative property:

\[ I \ast (H_1 \ast H_2) = (I \ast H_1) \ast H_2 \]

If we can separate a kernel \( H \) into two smaller kernels \( H = H_1 \ast H_2 \), then it will often be cheaper to apply \( H_1 \) followed by \( H_2 \), rather than \( H \).
Separability in $x$ and $y$

Sometimes we can separate a kernel into “horizontal” and “vertical” components.

Consider the kernels

$$H_x = [1 \ 1 \ 1 \ 1 \ 1], \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Then

$$H = H_x \ast H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$
Complexity of $x/y$-Separable Kernels

What is the number of operations for the $3 \times 5$ kernel $H$?
**Answer:** $15wh$

What is the number of operations for $H_x$ followed by $H_y$?
**Answer:** $3wh + 5wh = 8wh$

What about the case of a $M \times M$ kernel?
**Answer:**
$O(M^2)$ – no separability ($M^2wh$ operations)
$O(M)$ – with separability ($2Mwh$ operations)
Some More Filters

Box | Gaussian | Laplace

(a) | (b) | (c)

0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0

0 1 2 1 0
1 3 5 3 1
2 5 9 5 2
1 3 5 3 1
0 1 2 1 0

0 0 -1 0 0
0 -1 -2 -1 0
-1 -2 16 -2 -1
0 -1 -2 -1 0
0 0 -1 0 0
A “Better” Blurring

- The mean (box) filter gives “blocky” blurring.
- We would prefer something radially symmetric.
- Also, blurring looks better if the weighting dies off gradually, rather than all of a sudden.
- The Gaussian is radially symmetric and dies off gradually.
The Gaussian

In 1D:

\[ g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{x^2}{2\sigma^2} \right) \]

In 2D:

\[ G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \]
A 2D Gaussian is just the product of 1D Gaussians:

\[
G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right)
= \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{x^2}{2\sigma^2} \right) \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{y^2}{2\sigma^2} \right)
= g_\sigma(x) \cdot g_\sigma(y)
\]
Separability of 2D Gaussian

As a result, convolution with a Gaussian is separable:

\[ I \ast G = I \ast G_x \ast G_y, \]

where \( G \) is the 2D discrete Gaussian kernel; \( G_x \) is the “horizontal” and \( G_y \) the “vertical” 1D discrete Gaussian kernels.
Gaussian Filtering

1. Pick a $\sigma$ and radius $R = 3\sigma$
2. Compute a 1D array (kernel) with Gaussian values
   $k = \left[ g_\sigma(-R) \ldots g_\sigma(R) \right]$
3. Normalize this array to sum to one
4. Convolve horizontally by $k$
5. Convolve vertically by $k$
Implementation Detail

- Spatial filters cannot be done “in place”
- Because neighbor values are needed, we can’t overwrite them
- Need to compute into a copy image
- Multiple convolutions (e.g., separable filters) need to go back and forth between two images