Lecture 3: Convolution

Mark Hasegawa-Johnson

ECE 401: Signal and Image Analysis

Outline

Convolution

- A **convolution** is exactly the same thing as a **weighted local average**. We give it a special name, because we will use it very often. It's defined as:

$$y[n] = \sum_m g[m]f[n-m] = \sum_m g[n-m]f[m]$$

- We use the symbol $*$ to mean "convolution:"

$$y[n] = g[n] * f[n] = \sum_m g[m]f[n-m] = \sum_m g[n-m]f[m]$$

Convolution

$$\boxed{y[n] = g[n] * f[n] = \sum_m g[m]f[n-m] = \sum_m g[n-m]f[m]}$$

Here is the pseudocode for convolution:

1. For every output n:
   1. Reverse $g[m]$ in time, to create $g[-m]$.
   2. Shift it to the right by n samples, to create $g[n-m]$.
   3. For every m:
      1. Multiply $f[m]g[n-m]$.
   4. Add them up to create $y[n] = \sum_m g[n-m]f[m]$ for this particular n.
2. Concatenate those samples together, in sequence, to make the signal y.

Outline

LSI Systems and Convolution

We care about linearity and shift-invariance because of the following remarkable result:

LSI Systems and Convolution

Let $\mathcal{H}$ be any system,

$$x[n] \xrightarrow{H} y[n]$$

If $\mathcal{H}$ is linear and shift-invariant, then whatever processes it performs can be equivalently replaced by a convolution:

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

Impulse Response

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

The weights $h[m]$ are called the "impulse response" of the system. We can measure them, in the real world, by putting the following signal into the system:

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & \text{otherwise} \end{cases}$$

and measuring the response:

$$\delta[n] \xrightarrow{H} h[n]$$

Convolution: Proof

1. $h[n]$ is the impulse response.

$$\delta[n] \xrightarrow{H} h[n]$$

2. The system is **shift-invariant**, therefore

$$\delta[n-m] \xrightarrow{H} h[n-m]$$

3. The system is **linear**, therefore **scaling the input by a constant** results in **scaling the output by the same constant**:

$$x[m]\delta[n-m] \xrightarrow{H} x[m]h[n-m]$$

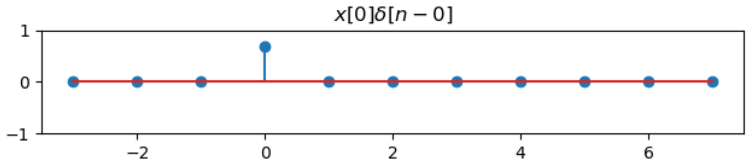
4. The system is **linear**, therefore **adding input signals** results in **adding the output signals**:

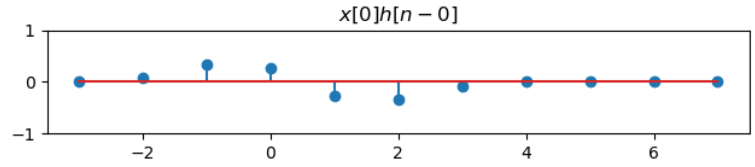
$$\sum_{m=-\infty}^{\infty} x[m]\delta[n-m] \xrightarrow{H} \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

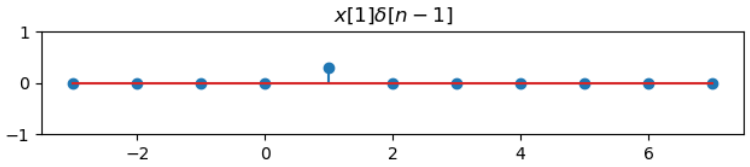
Convolution: Proof (in Words)

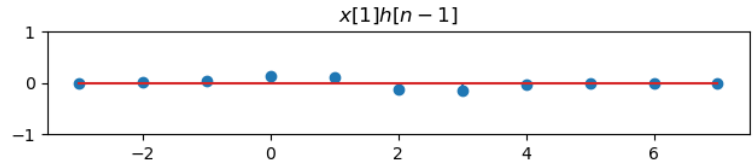
- The input signal, $x[n]$, is just a bunch of samples.
- Each one of those samples is a scaled impulse, so each one of them produces a scaled impulse response at the output.
- Convolution = add together those scaled impulse responses.

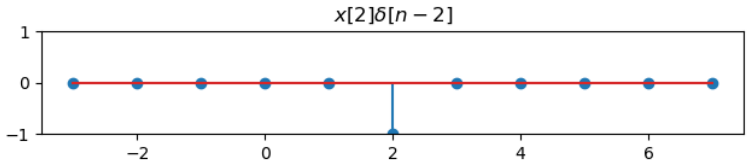
Convolution: Proof (in Pictures)

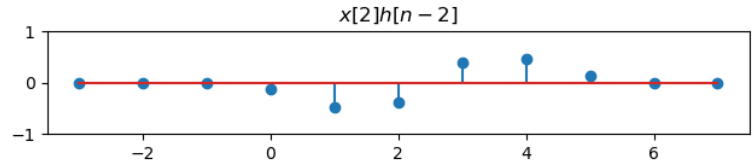


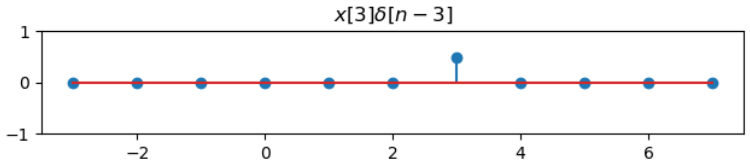


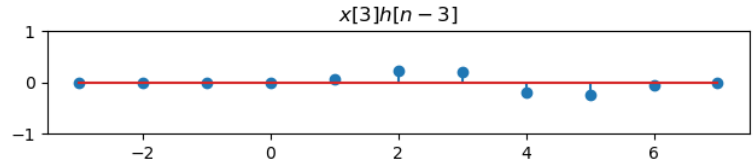


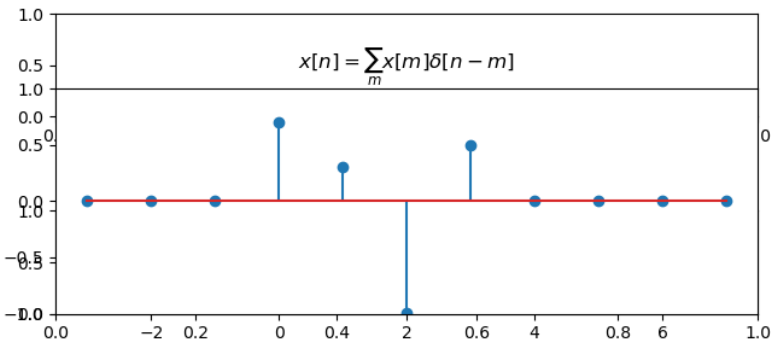


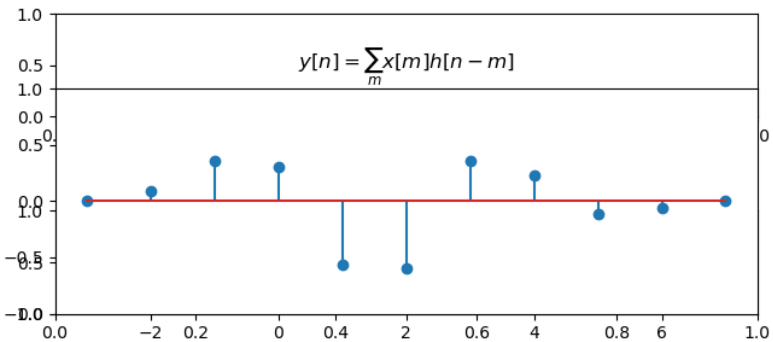












Outline

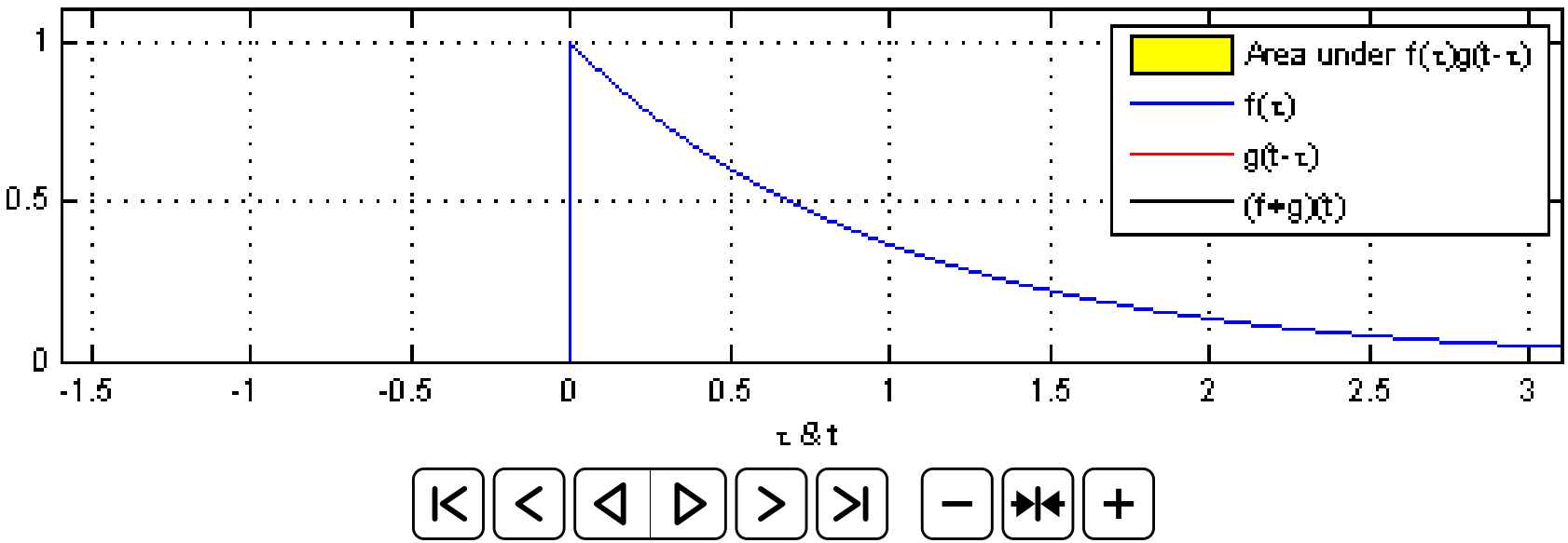
Convolution: how should you implement it?

- When writing code: use the numpy function, `np.convolve`. In general, if numpy has a function that solves your problem, you are *always* permitted to use it.
- When solving problems with pencil and paper: use *graphical convolution*.

Graphical Convolution

1. Choose one of the two functions whose breakpoints are easier to shift (i.e., it breaks at easy values like $n = 0$), and call that $f[n]$. Call the other function $g[n]$.
2. Plot $g[m]$ as a function of m.
3. Underneath, plot $f[n - m]$ as a function of m for some particular n.
4. Under that, plot $g[m]f[n - m]$ for the same particular n.
5. Use your plot as a guide to help you write the equation $\sum_m g[m]f[n - m]$ in a solvable form. Solve it to find $y[n]$.
6. If this gives you enough information to find $y[n]$ for every other n, then do so. If there's some other n that's not yet obvious to you, then repeat above process for the other n.

Graphical Convolution: A Video from Wikipedia



by Brian Amberg, CC-SA 3.0,

https://commons.wikimedia.org/wiki/File:Convolution_of_spiky_function_with_box2.gif

Quiz

Do the quiz! Go to the course webpage, and try the quiz.

Outline

Differencing is convolution, too

Suppose we want to compute the local difference:

$$y[n] = f[n] - f[n-1]$$

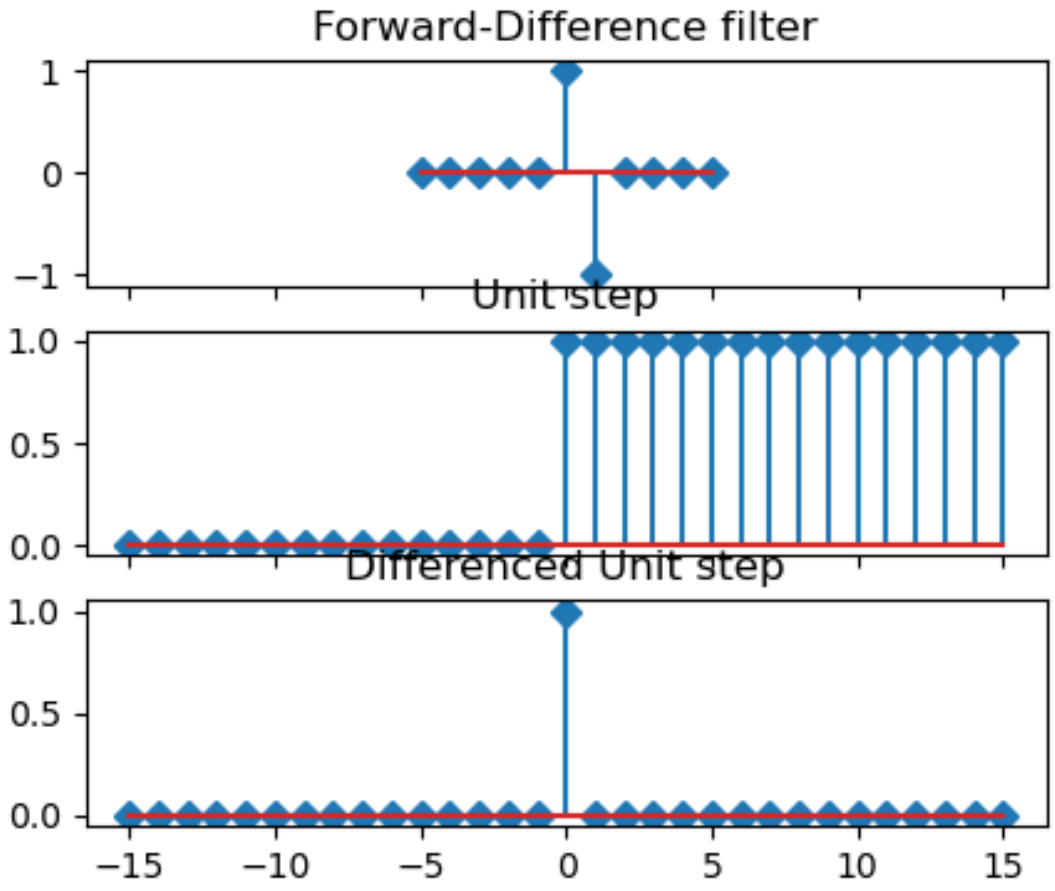
We can do that using a convolution!

$$y[n] = \sum_m f[n-m]h[m]$$

where

$$h[m] = \begin{cases} 1 & m = 0 \\ -1 & m = 1 \\ 0 & \text{otherwise} \end{cases}$$

Differencing as convolution



Outline

Weighted differencing as convolution

- The formula $y[n] = f[n] - f[n-1]$ is kind of noisy. Any noise in $f[n]$ or $f[n-1]$ means noise in the output.
- We can make it less noisy by
  1. First, compute a weighted average:

$$y[n] = \sum_m f[m]g[n-m]$$

  2. Then, compute a local difference:

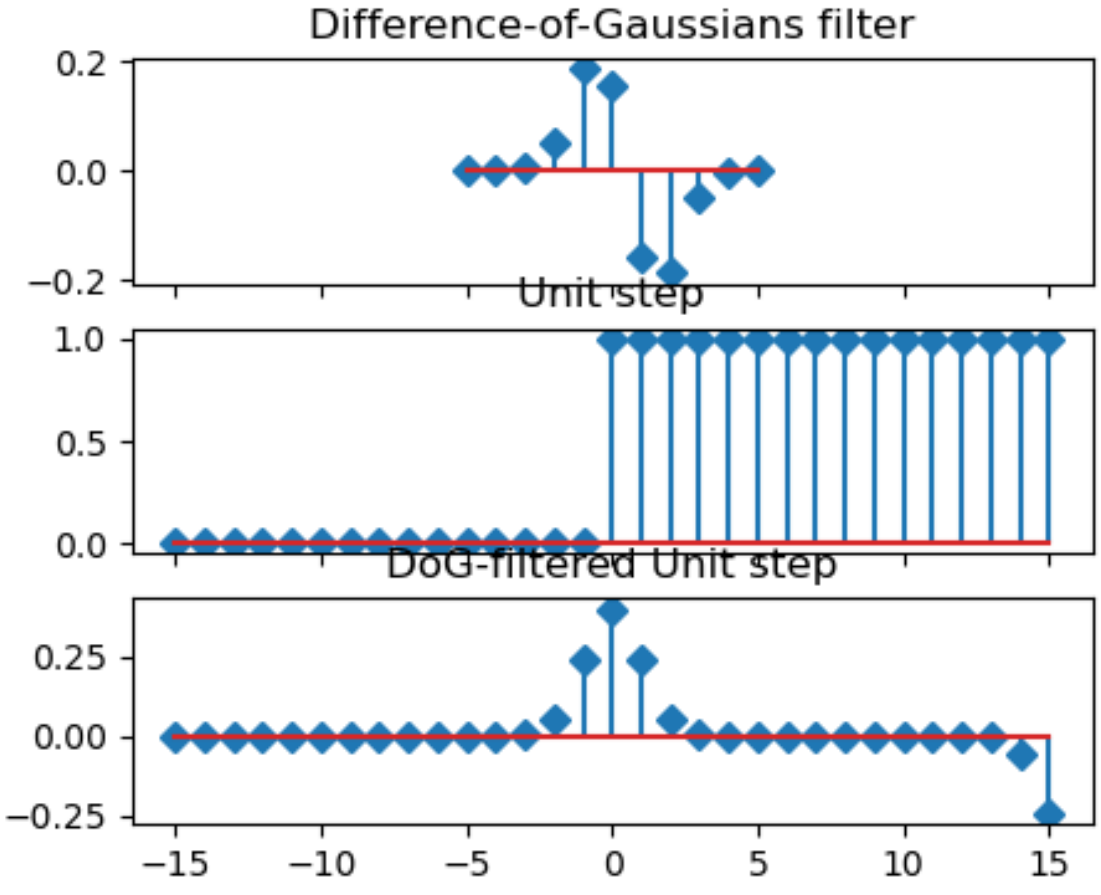
$$z[n] = y[n] - y[n-1] = \sum_m f[m]\left(g[n-m] - g[n-1-m]\right)$$

This is exactly the same thing as convolving with

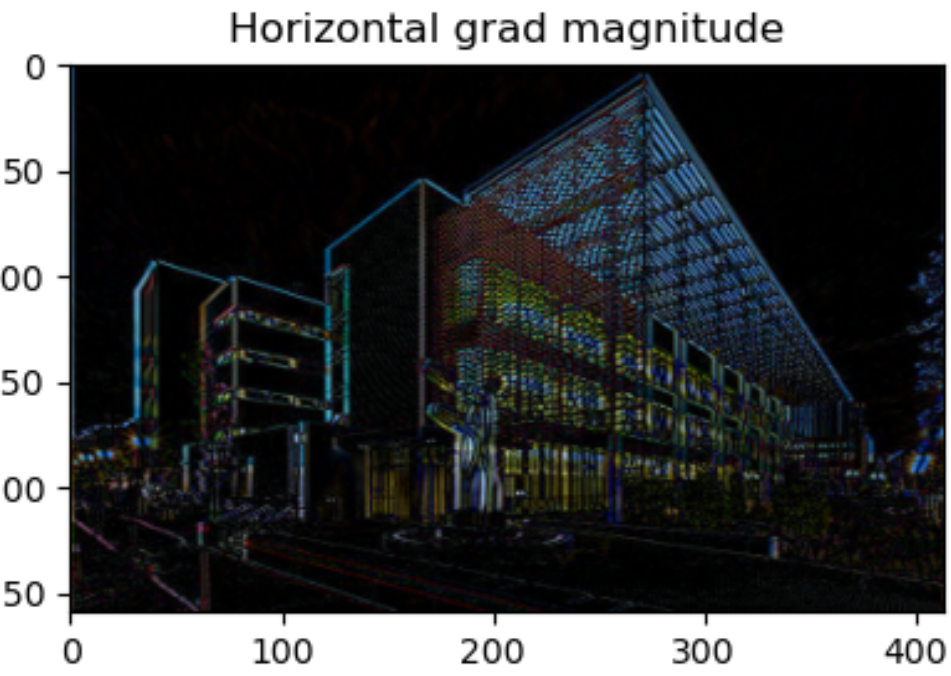
$$h[n] = g[n] - g[n-1]$$

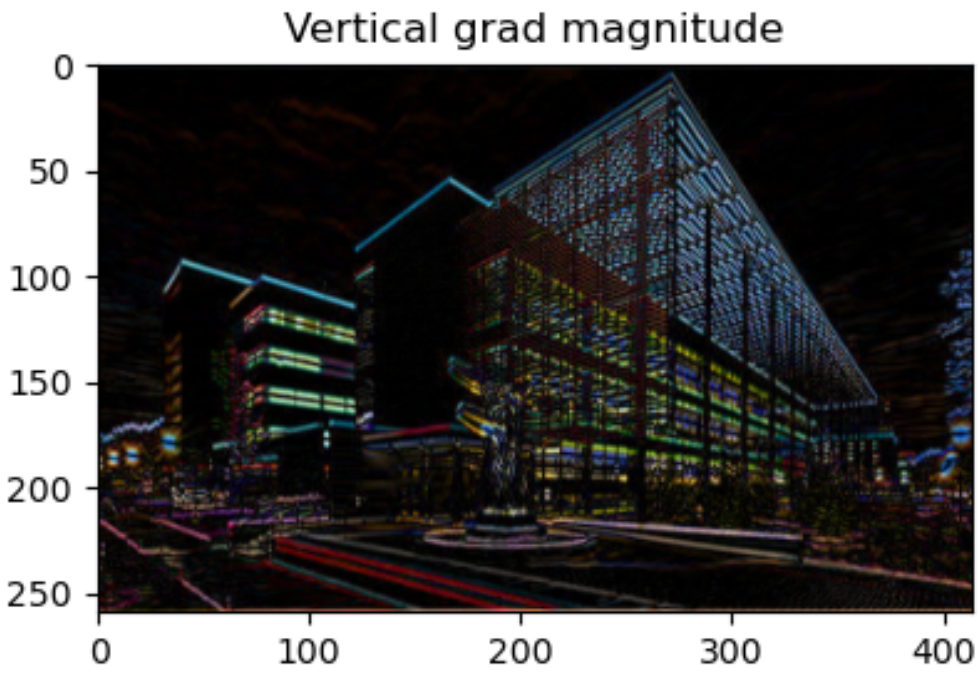
A difference-of-Gaussians filter

The top row is a "difference of Gaussians" filter, $h[n] = g[n] - g[n-1]$, where $g[n]$ is a Gaussian. The middle row is $f[n]$, the last row is the output $z[n]$.



Difference-of-Gaussians filtering in both rows and columns





Outline

Image gradient

- Suppose we have an image $f[i,j,k]$. The 2D image gradient is defined to be

$$\vec{G}[i,j,k] = \left(\frac{df}{di}\right)\hat{i} + \left(\frac{df}{dj}\right)\hat{j}$$

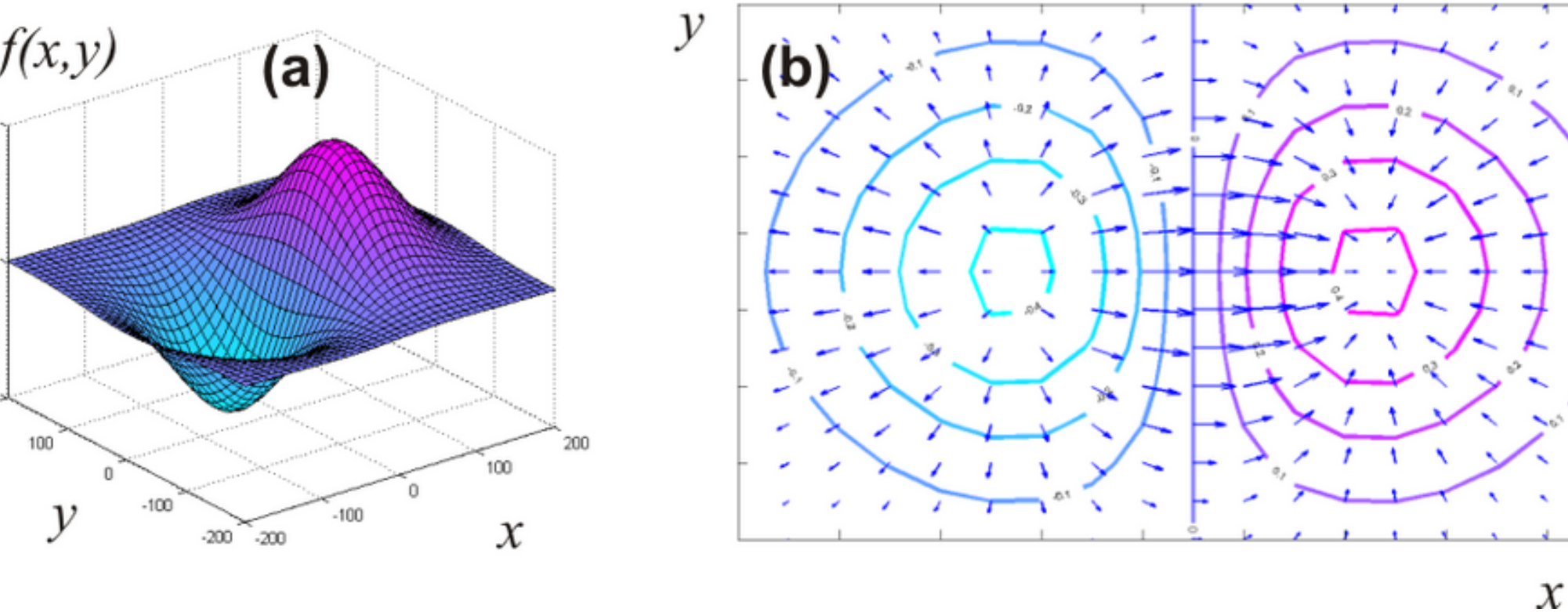
  where $\hat{i}$ is a unit vector in the i direction, $\hat{j}$ is a unit vector in the j direction.

- We can approximate these using the difference-of-Gaussians filter, $h_{dog}[n]$:

$$\frac{df}{di} \approx G_i = h_{dog}[i] * f[i,j,k]$$
$$\frac{df}{dj} \approx G_j = h_{dog}[j] * f[i,j,k]$$

The gradient is a vector

The image gradient, at any given pixel, is a vector. It points in the direction of increasing intensity (this image shows "dark" = greater intensity).



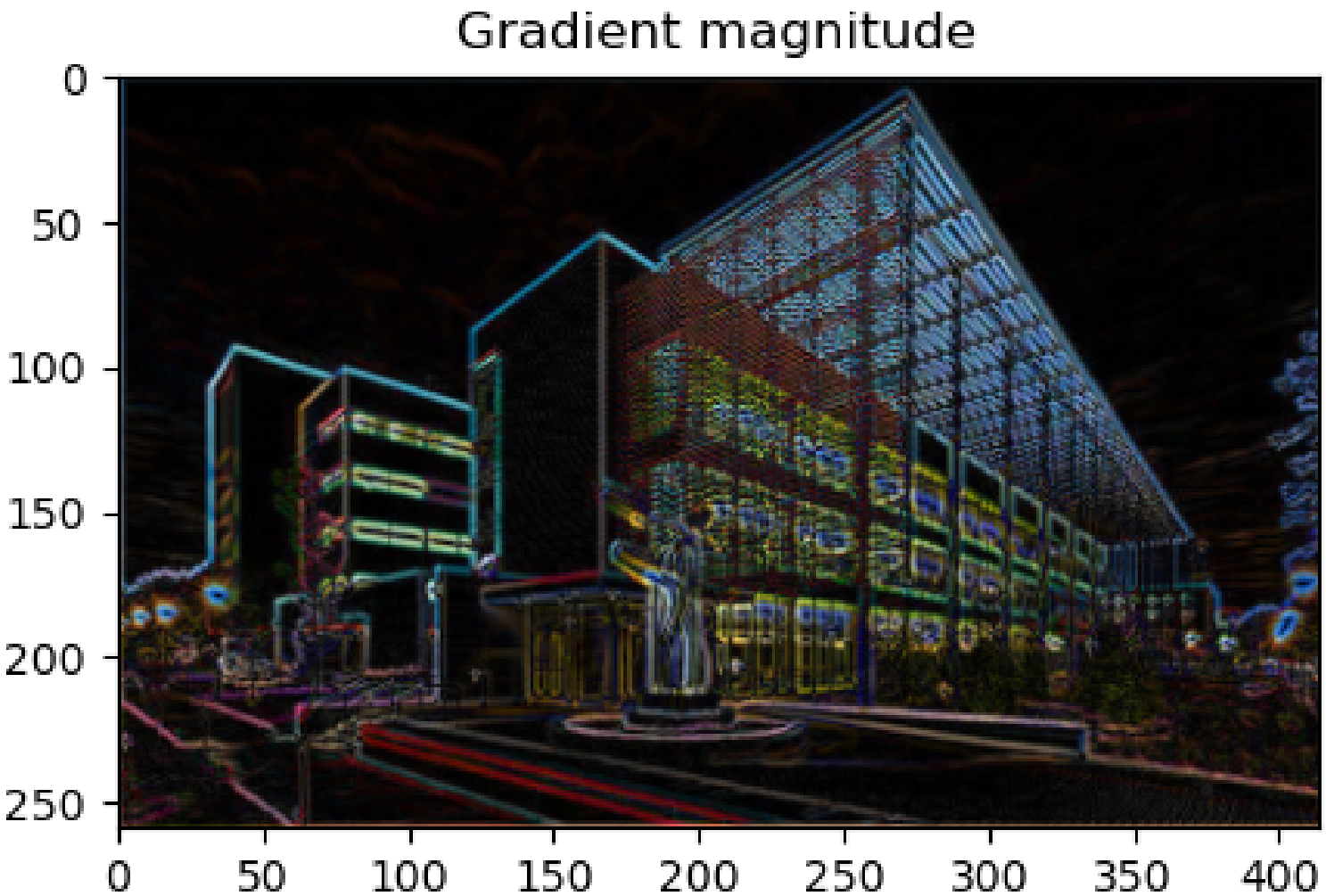
By CWeiske, CC-SA 2.5, https://commons.wikimedia.org/wiki/File:Gradient2.svg

Magnitude of the image gradient

- The image gradient, at any given pixel, is a vector.
- It points in the direction in which intensity is increasing.
- The magnitude of the vector tells you how fast intensity is changing.

$$\|\vec{G}\| = \sqrt{G_i^2 + G_j^2}$$

Magnitude of the gradient = edge detector



Outline

Summary

If a system is **linear and shift-invariant** (LSI), then it can be implemented using convolution:

$$y[n] = h[n] * x[n] = \sum_m h[m]x[n-m] = \sum_m h[n-m]x[m]$$

where $h[n]$ is the impulse response:

$$\delta[n] \xrightarrow{\mathcal{H}} h[n]$$