CSC 433/533 Computer Graphics

Anti-Aliasing and
Signal Processing
Sampling, Smoothing and Convolutions

Recall: Images are Functions

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, $I: R \rightarrow V$
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)
- The range is:
  - A set of possible values.
  - …in the space of color values we're encoding

Concept for the Day: Pixels are Samples of Image Functions

Image Samples

- Each pixel is a sample of what?
  - One interpretation: a pixel represents the intensity of light at a single (infinitely small point in space)
- The sample is displayed in such a way as to spread the point out across some spatial area (drawing a square of color)

Continuous vs. Discrete

- Key Idea: An image represents data in either (both?) of
  - Continuous domain: where light intensity is defined at every (infinitesimally small) point in some projection
  - Discrete domain, where intensity is defined only at a discretely sampled set of points.
  - This seem like a philosophical discussions without clear practical applications. Surprisingly, it has very concrete algorithmic applications.

Converting Between Image Domains

- When an image is acquired, an image is **sampled** from some continuous domain to a discrete domain.
- **Reconstruction** converts digital back to continuous.
- The reconstructed image can then be **resampled** and **quantized** back to the discrete domain.

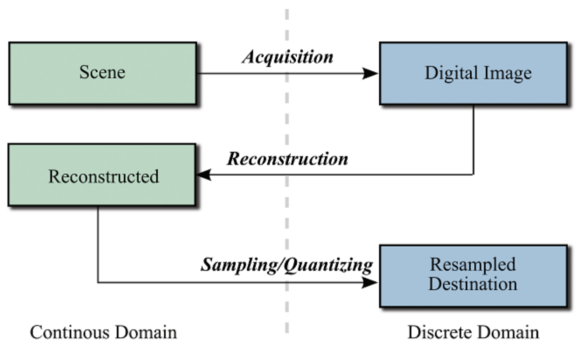
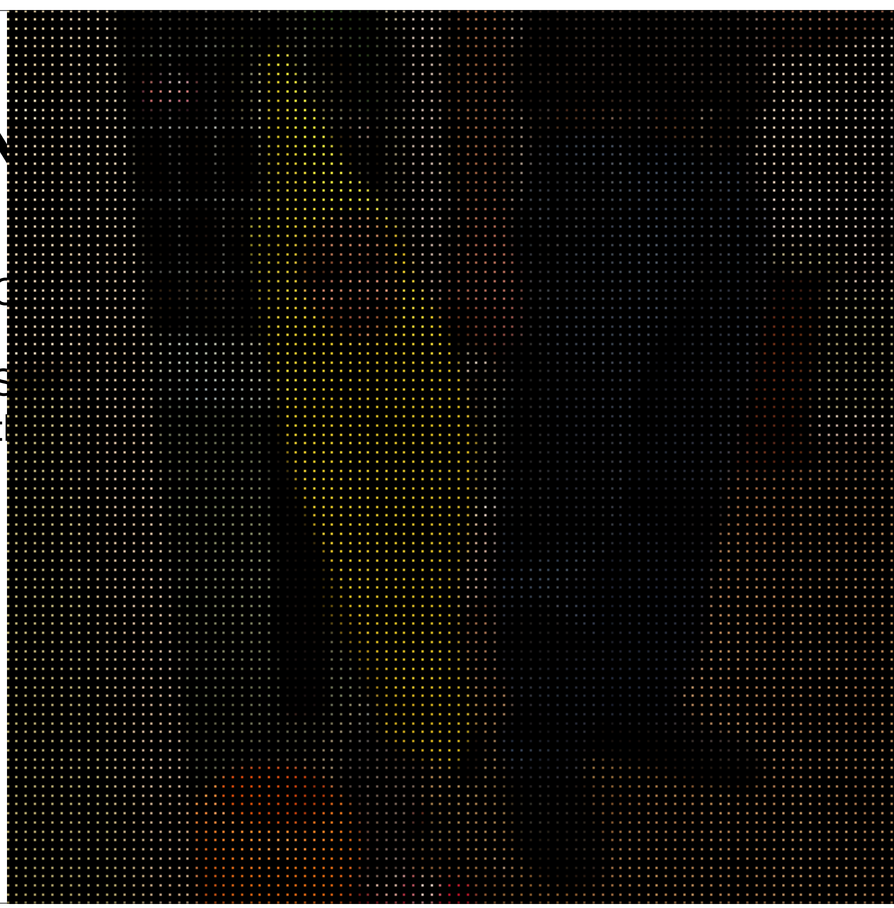


Figure 7.7. Resampling.

Naive Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```



N g

- C
- S
  t

What's the Problem?

- The output image has gaps!
- Why: we skip a many of the pixels in the output.
- Why don't we fix this by changing the code to at least put some color at each pixel of the output?

Naive Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```

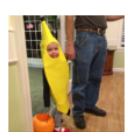
“Inverse” Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let index = (row/k)*W + (col/k);
      let index2 = row*k*W + col;
      output[index2] = input[index];
   }
}
```

Inverse Image Rescaling

400x400 image

Not great, but could become worse



100x100 image

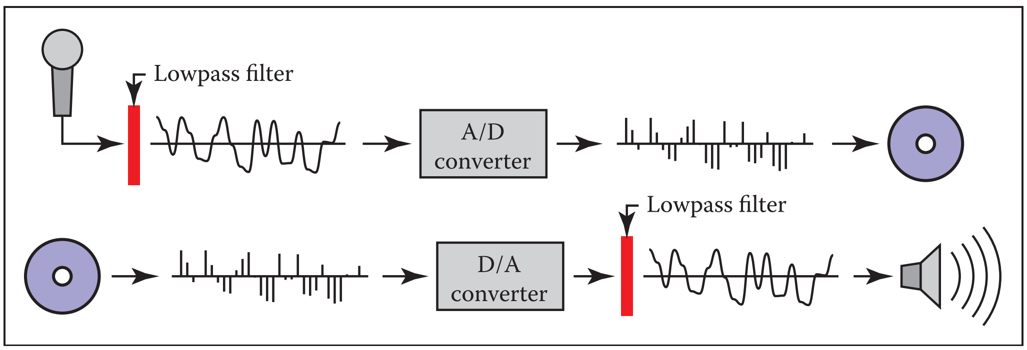
What’s the Problem?

- The output image is too “blocky”
- Why: because our image reconstruction rounds the index to the nearest integer pixel coordinates
  - Rounding to the “nearest” is why this type of interpolation is called **nearest neighbor interpolation**

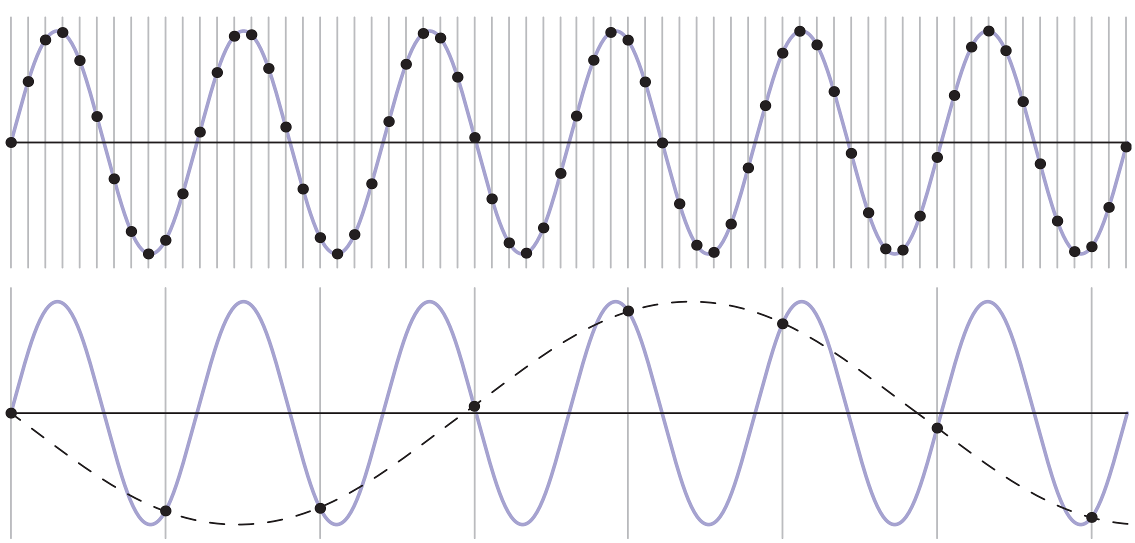
Sampling Artifacts / Aliasing

Motivation: Digital Audio

- Acquisition of images takes a continuous object and converts this signal to something digital
- Two types of artifacts:
  - **Undersampling** artifacts: on acquisition side
  - **Reconstruction** artifacts: when the samples are interpreted

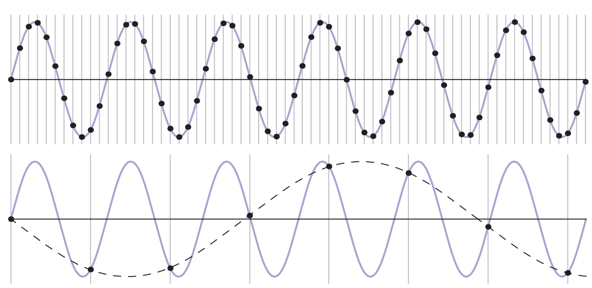


Undersampling Artifacts



Shannon-Nyquist Theorem

(not needed for the exam)

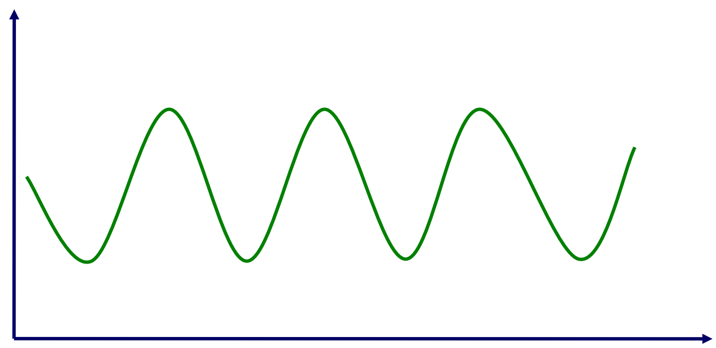


- The sampling frequency must be **<u>double</u>** the highest frequency of the content.
- If there are any higher frequencies in the data, or the sampling rate is too low, **aliasing**, happens
  - Named this because the discrete signal “pretends” to be something lower frequency

S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

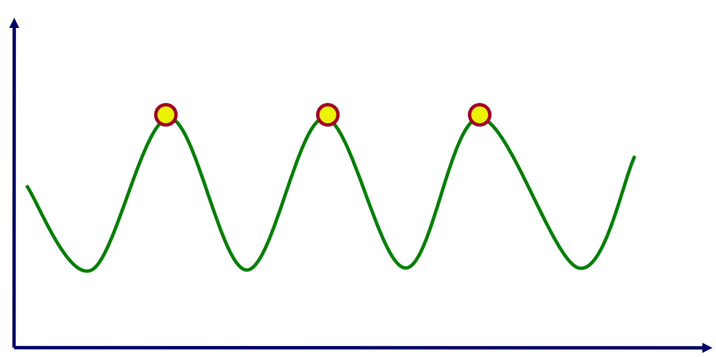
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

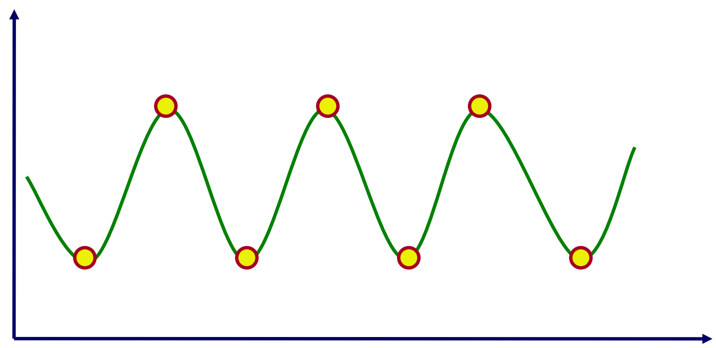
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

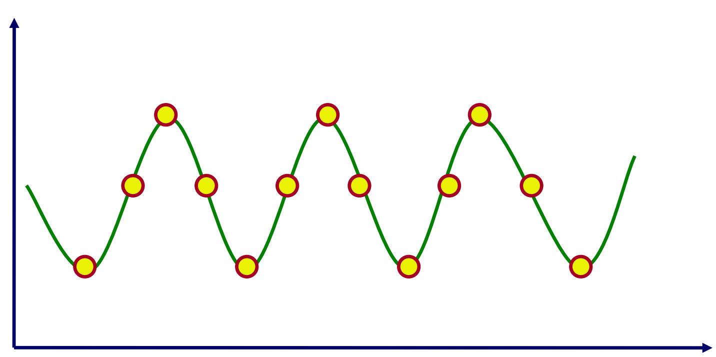
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

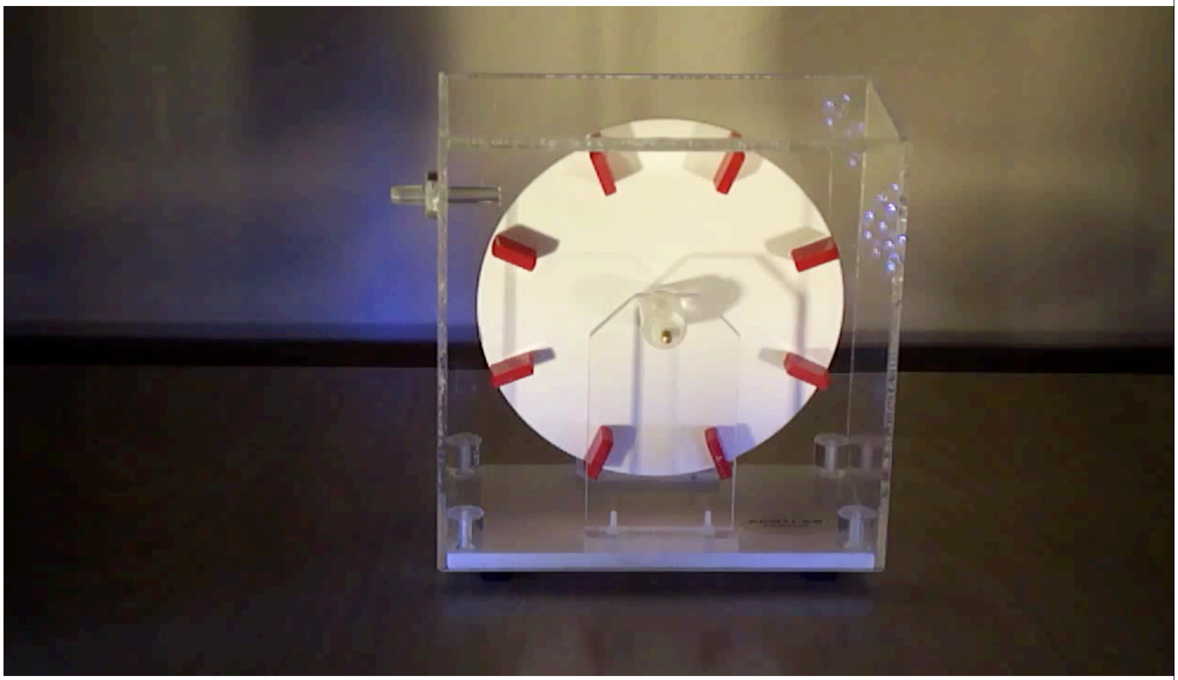


S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?





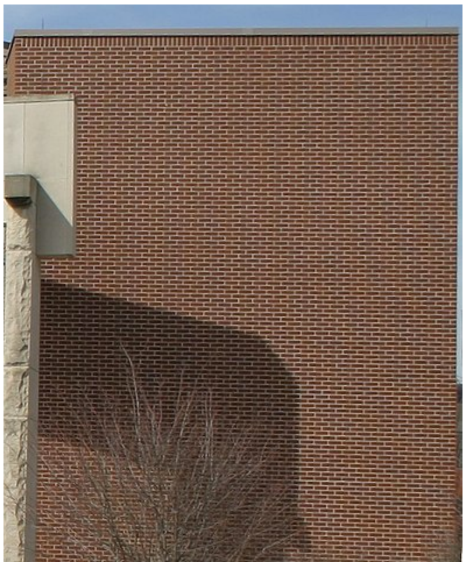
http://youtu.be/0k2lhYk6Lfs?rel=0

Aliasing in images

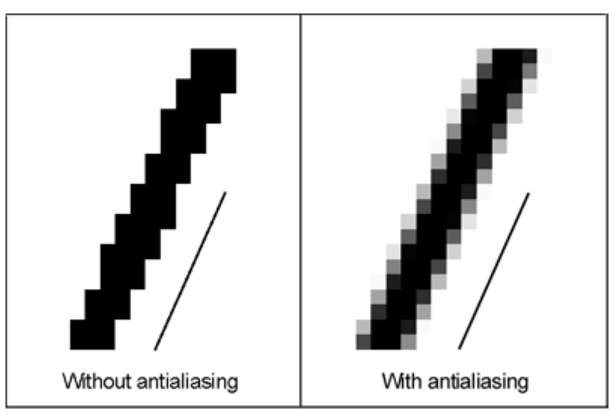
Two outcomes of under-sampling

1) Moire Pattern
2) Rasterization

Moire Patterns



Aliasing for edges

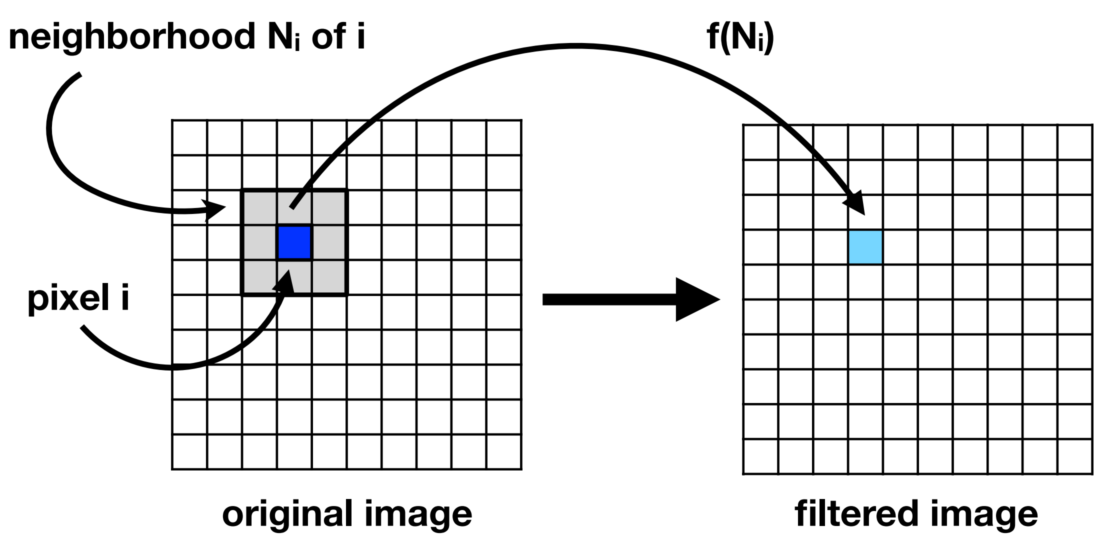


Each pixel is effected by nearby pixels
For example, even though the input image image is black/white,
We allow grey values for output pixels.

Convolution

Each pixel is effected by nearby pixels
For example, even though the image is black/white,
We allow grey values

Neighborhood Filtering (Schematic)



An Example: Mean Filtering

- Mean filters sum all of the pixels in a local neighborhood N_i and divide by the total number, computing the average pixel.
- Said another way, we replace each pixel as a linear combination of its neighbors (with equal weights!)
- To find the new color of a pixel *j,* we will look at N_j , defined as the (say) 3×3 neighborhood, and set

$$f(N_j) = \frac{1}{|N_j|} \sum_{p_k \in N_j} C_k$$

- Where the N_i is a square, we call these **box** filters
- Think about it as a weighted average:

$$f(N_i) = \sum_{\textbf{pixel j in the rergion N i}} w_j C_j$$

- The weights $w_1 \dots w_k$ are convex combination. Meaning that they are all positive, and $w_1 + w_2 + \dots w_k = 1$. For example, $w_1 = w_2 = w_3 = \frac{1}{3}$

Box Filtering

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

$$1/9 * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

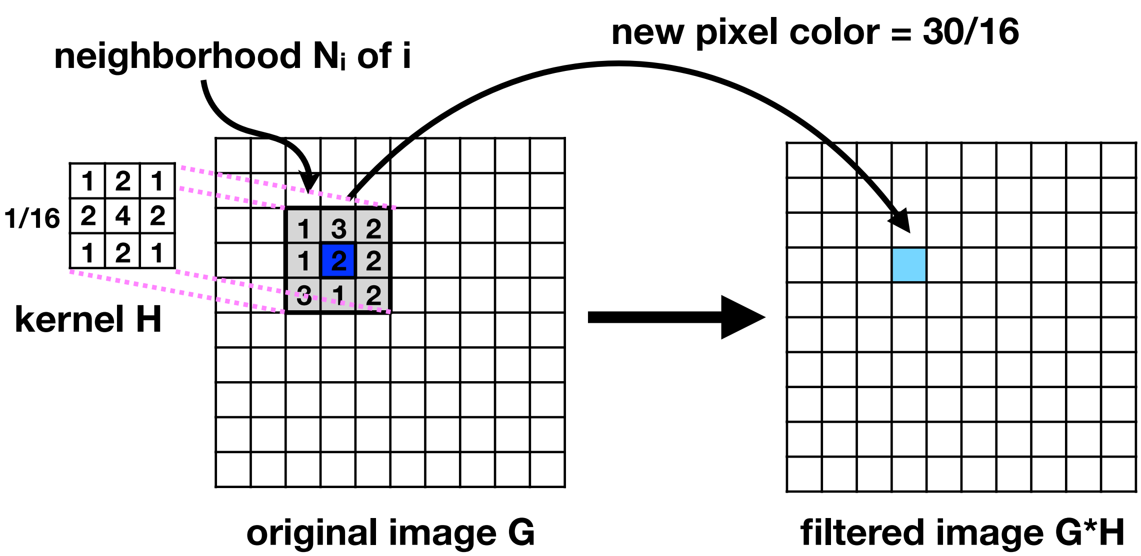
Box Filtering

$$\frac{1}{9} * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$\frac{1}{25} * \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Convolution

- This process of adding up pixels multiplied by various weights is called **convolution**



Kernels

- Convolution employs a rectangular grid of coefficients, known as a **kernel**
- Kernels are like a neighborhood mask, they specify which elements of the image are in the neighborhood and their relative weights.
- A kernel is a set of weights that is applied to corresponding input samples that are summed to produce the output sample.
- For smoothing purposes, the sum of weights must be 1

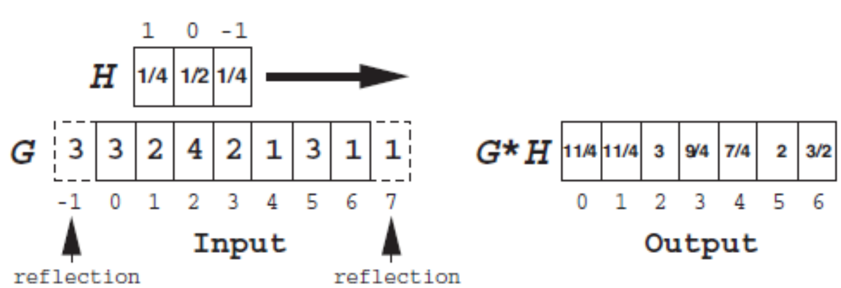
$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \quad \frac{1}{13}\begin{pmatrix}1 & 1 & 1\\ 1 & 5 & 1\\ 1 & 1 & 1\end{pmatrix} \quad \frac{1}{37}\begin{pmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 2 & 2 & 2 & 1\\ 1 & 2 & 5 & 2 & 1\\ 1 & 2 & 2 & 2 & 1\\ 1 & 1 & 1 & 1 & 1\end{pmatrix}$$

One-dimensional Convolution

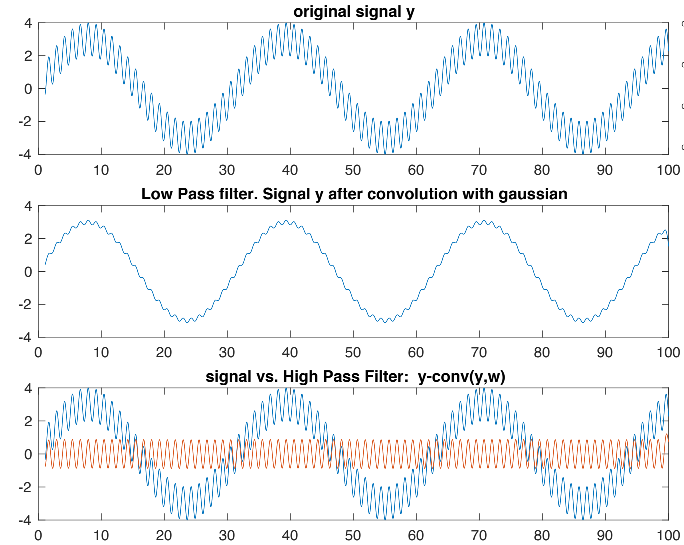
- Can be expressed by the following equation, which takes a filter H and convolves it with G:

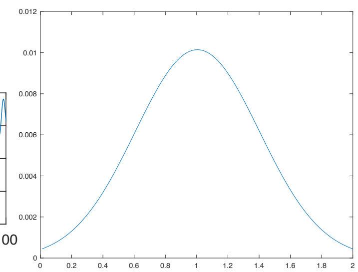
$$\hat{G}[i] = (G * H)[i] = \sum_{j=i-n}^{i+n} G[j]H[i-j],\ i \in [0, N-1]$$

- Equivalent to sliding a window



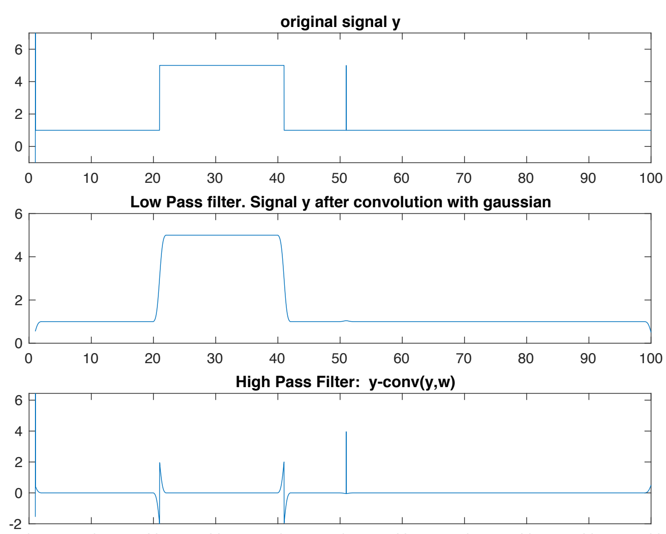
Low pass and hight pass filters

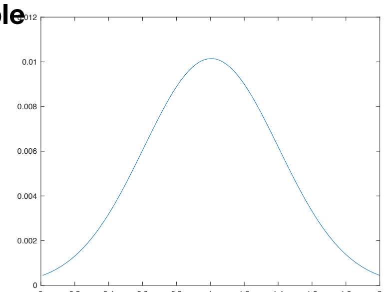




We convolved the original signal y with this gaussian

Low pass and hight pass filters - another example



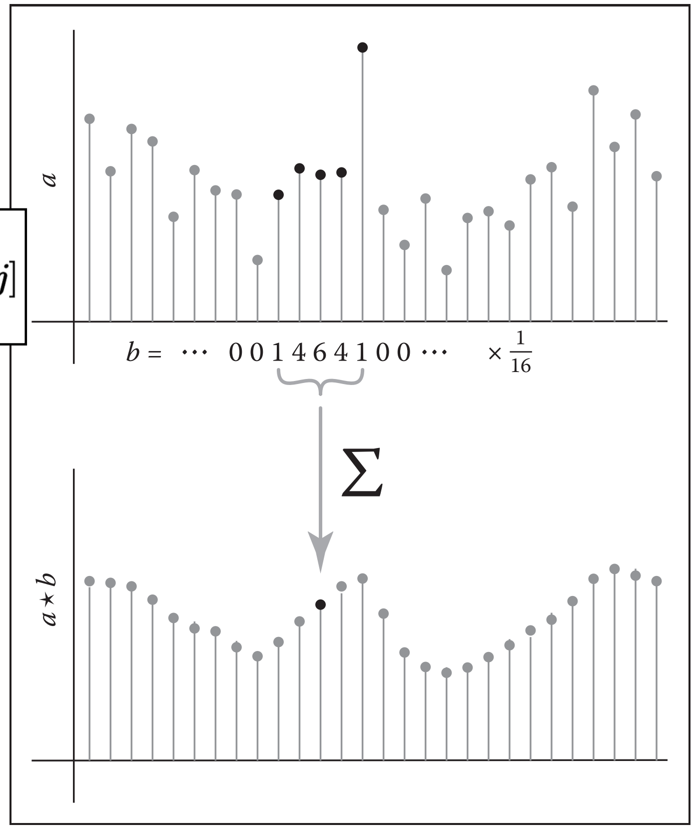


We convolved the original signal y with this gaussian

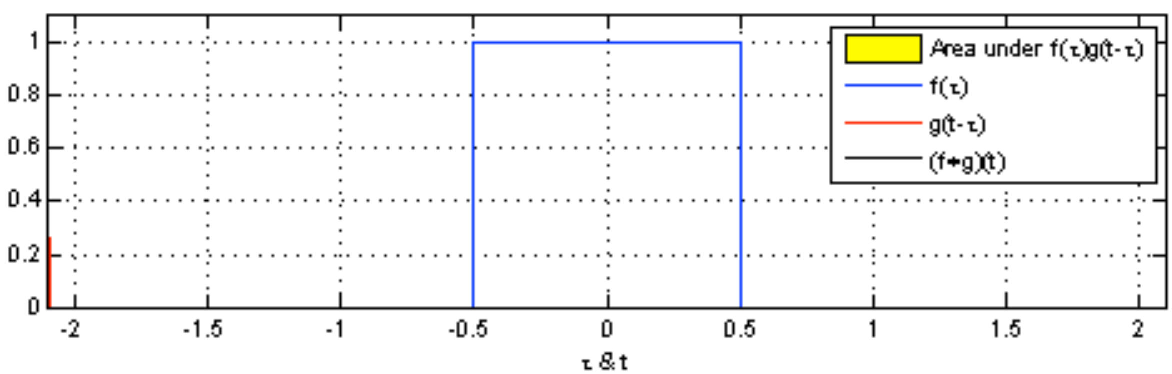
Convolution is a Moving, Weighted Average

$$(a \star b)[i] = \sum_{j=i-r}^{i+r} a[j]b[i-j]$$

- Mathematically, this is equivalent to integrating the product of a and b with a shift in the domain
- Compare a to a*b on the right



Box Filters (Animated)



- The above is continuous, but a discrete version could be imagined as:

H = 111

G = ...0000011100000...

G*H = ...0000012100000...

http://en.wikipedia.org/wiki/Convolution

More examples: http://math.mit.edu/daimp/ConvFlipDrag.html

2-Dimensional Version

- Given an image a and a kernel b with $(2r+1)^2$ values, the convolution of a with b is given below as a*b:

$$(a \bigstar b)[i,j] = \sum_{i'=i-r}^{i+r} \sum_{j'=j-r}^{j+r} a[i',j']\,b[i-i',j-j']$$

- The (i-i') and (j-j') terms can be understood as reflections of the kernel about the central vertical and horizontal axes.

- The kernel weights are multiplied by the corresponding image samples and then summed together.

A Note on Indexing

- Convolution **reflects** the filter to preserve orientation.
- **Correlation** does **not** have this reflection.
    - But we often use them interchangeably since most kernels are symmetric!!

Convolution reflects and shifts the kernel

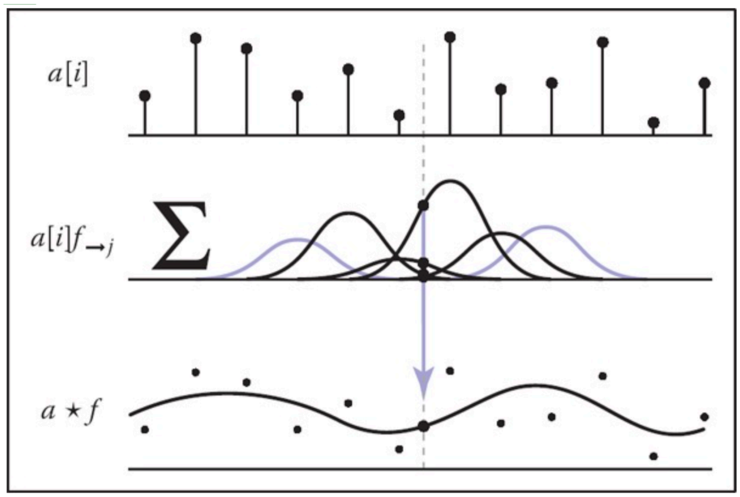
Given kernel H = $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$

$$\begin{matrix} 9 & 8 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 5 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 8 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

G*H

Convolution Can Also Convert from Discrete to Continuous

- Discrete signal a
- Continuous filter f
- Output a*f defined on positions x as opposed to discrete pixels i



$$(a \star f)(x) = \sum_{i=\lceil x-r \rceil}^{\lfloor x+r \rfloor} a[i] f(x-i)$$

B g

Filtering helps to reconstruct the signal better when rescaling

Inverse Rescaling

Reconstructed w/ Discrete-to-Continuous

Discrete-Continuous Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let x = col/k;
      let y = row/k;
      let index = row*k*W + col;
      output[index] = reconstruct(input,x,y);
   }
}
```

Types of Filters: Smoothing

Smoothing Spatial Filters

- Any weighted filter with positive values will smooth in some way, examples:

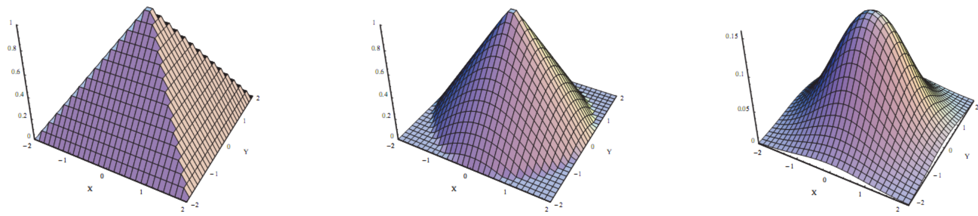
$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \qquad \frac{1}{16} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

- Normally, we use integers in the filter, and then divide by the sum (computationally more efficient)
- These are also called **blurring** or **low-pass** filters

Smoothing Kernels

$$f(x,y) = -\alpha \cdot \max(|x|,|y|)$$

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$



(a) Pyramid. (b) Cone. (c) Gaussian.

$$f(x,y) = -\alpha \cdot \sqrt{x^2+y^2}$$

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

(a) Pyramid.

0	0	1	0	0
0	2	2	2	0
1	2	5	2	1
0	2	2	2	0
0	0	1	0	0

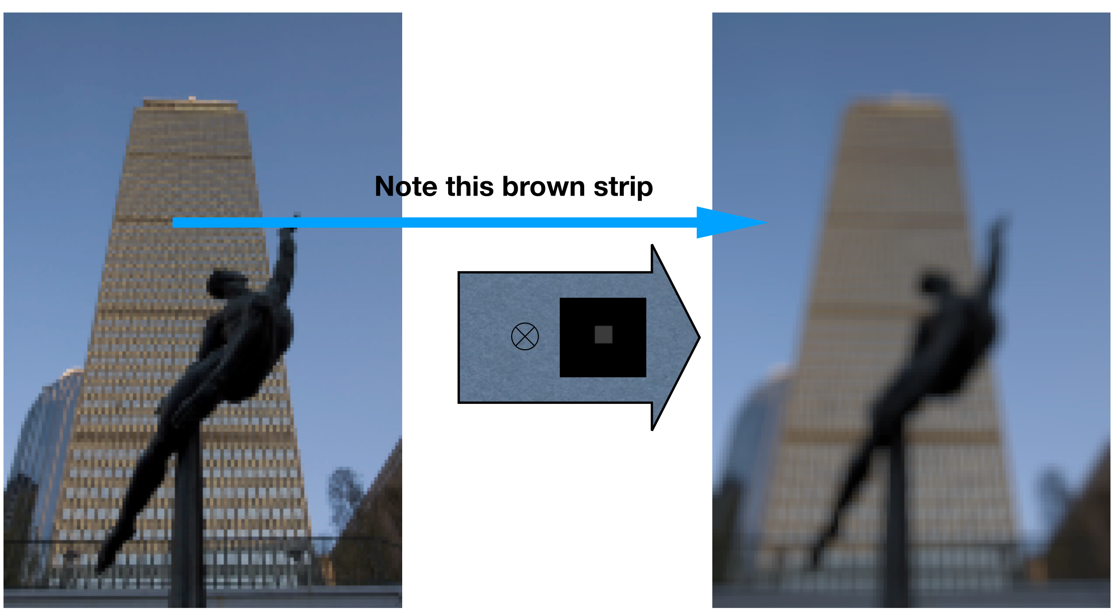
(b) Cone.

1	4	7	4	1
4	16	28	16	4
7	28	49	28	7
4	16	28	16	4
1	4	7	4	1

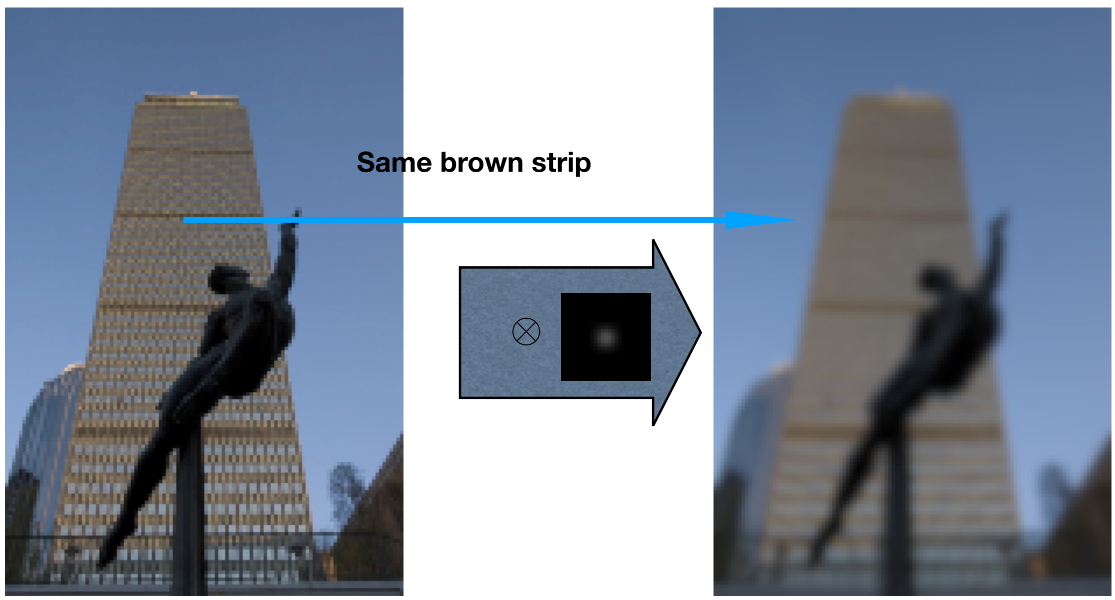
(c) Gaussian.

Table 6.1. Discretized kernels.

Box Filter



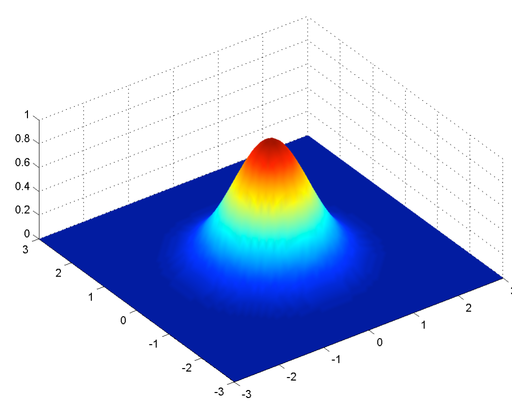
Gaussian Filter



Gaussians

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- Gaussian kernel is parameterized on the standard deviation σ
- Large σ's reduce the center peak and spread the information across a larger area
- Smaller σ's create a thinner and taller peak
- Gaussians are smooth everywhere.
- Gaussians have infinite **support**
    - >0 everywhere
- But often truncate to 2σ or 3σ
- Volume =1 (sum of weights =1)



http://en.wikipedia.org/wiki/Gaussian_function

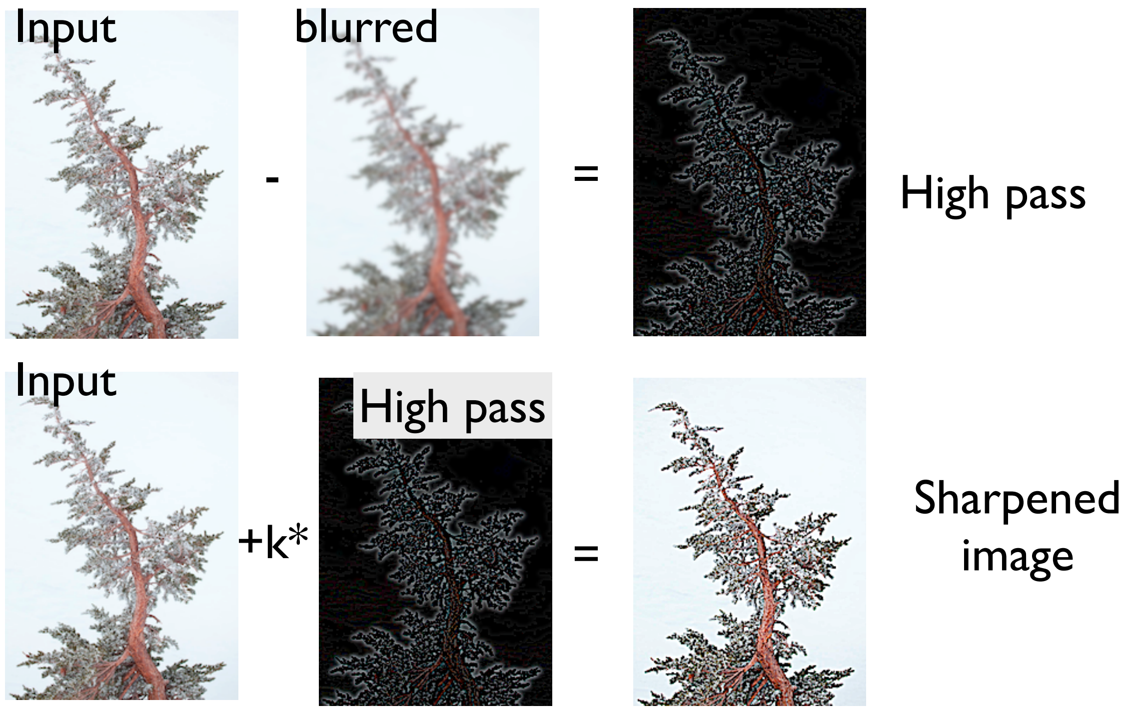
Smoothing Comparison

(a) Source image. (b) 17 × 17 Box. (c) 17 × 17 Gaussian.

Figure 6.10. Smoothing examples.

Types of Filters: Sharpening

Sharpening (Idea)

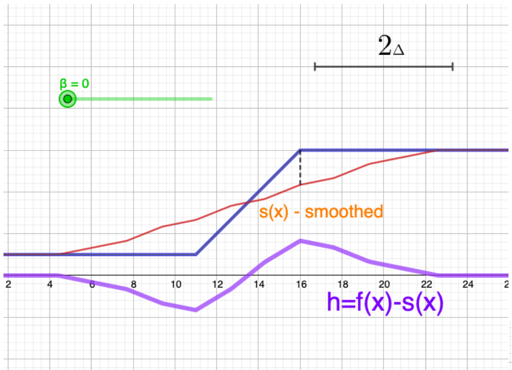


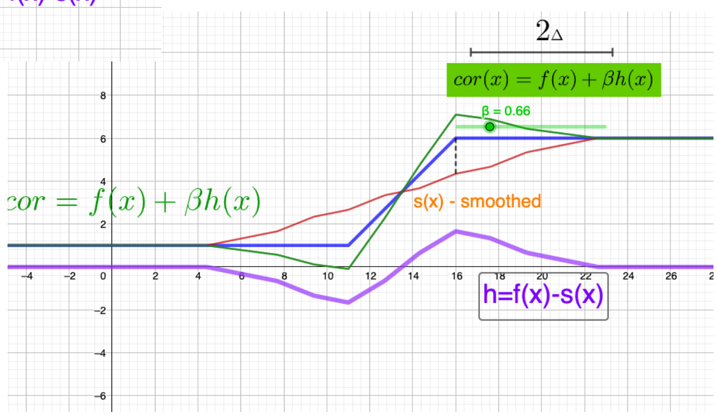
Another example

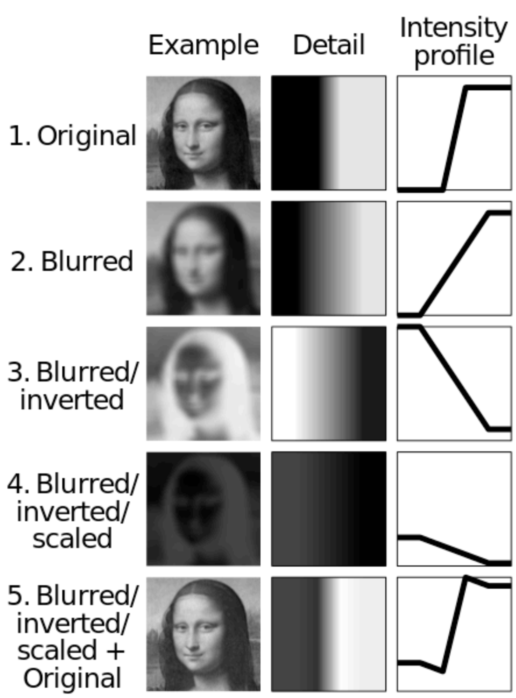
Original Image, Imaged convolved

Left: difference (only boundaries are non-black)
Right Imaged minus differences convolved

Why does it work







Credit: Wikipedia

Sharpening is a Convolution

- This procedure can then expressed as a single kernel
- Assume $I(p)$ is the intensity at pixel p.
- We create its smoothed (low-pass) averaging by convolving it with a kernel K. So $S = I \otimes K$.
- For example $K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ Lets define: $\mathbf{1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- As usual, we obtain the high-frequencies by removing the low frequencies from I. That is, $H = I - (I \otimes K)$.
- User picks a value $\alpha > 0$. To corrected image

$$
\begin{aligned}
I_{sharp} &= I + \alpha \cdot H \\
& I + \alpha(I - S) \\
&= I + \alpha(I - (I \otimes K)) \\
&= I(1+\alpha) - \alpha(I \otimes K) \\
&= I \otimes = K_{sharp}
\end{aligned}
$$

$$K_{sharp} = \frac{1}{9}\begin{pmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & 9+8\alpha & -\alpha \\ -\alpha & -\alpha & -\alpha \end{pmatrix}$$

If our image has only one raw

$$K_{sharp} = [-\alpha \quad 1+2\alpha \quad -\alpha]$$

Sharpening is a Convolution

$$
\begin{aligned}
I_{\text{sharp}} &= (1+\alpha)I - \alpha(I \star f_{g,\sigma}) \\
&= I \star ((1+\alpha)d - \alpha f_{g,\sigma}) \\
&= I \star f_{\text{sharp}}(\sigma, \alpha),
\end{aligned}
$$

Note: could also define d as $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

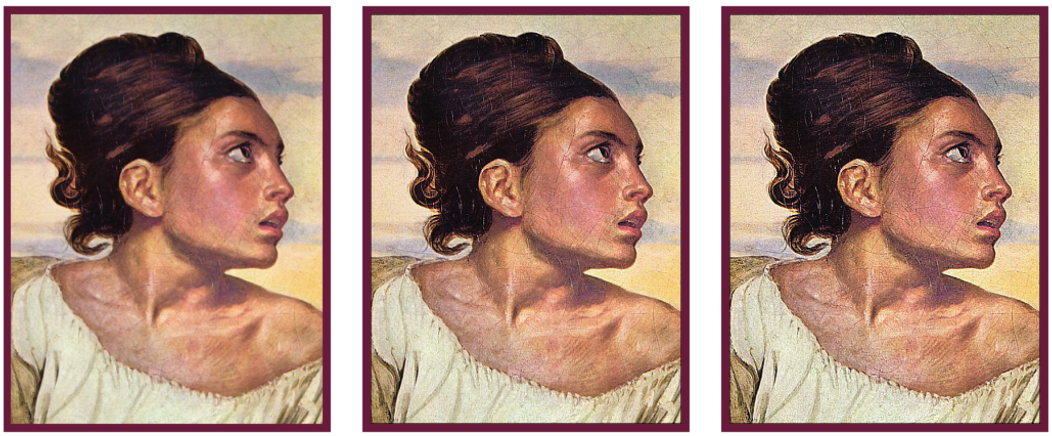
$$d = \frac{1}{9} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$f_{g,\sigma} = \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

$$((1+\alpha)d - \alpha f_{g,\sigma}) = \frac{1}{9} \times \begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}$$

Edge Enhancement

- The parameter α controls how much of the source image is passed through to the sharpened image.



(a) Source image. (b) $\alpha = .5$. (c) $\alpha = 2.0$.

Figure 6.20. Image sharpening.

Defining Edges

- Sharpening uses negative weights to enhance regions where the image is changing rapidly
  - These rapid transitions between light and dark regions are called **edges**
- Smoothing reduces the strength of edges, sharpening strengthens them.
  - Also called **high-pass** filters
- Idea: smoothing filters are weighted averages, or integrals. Sharpening filters are weighted differences, or derivatives!

Edges

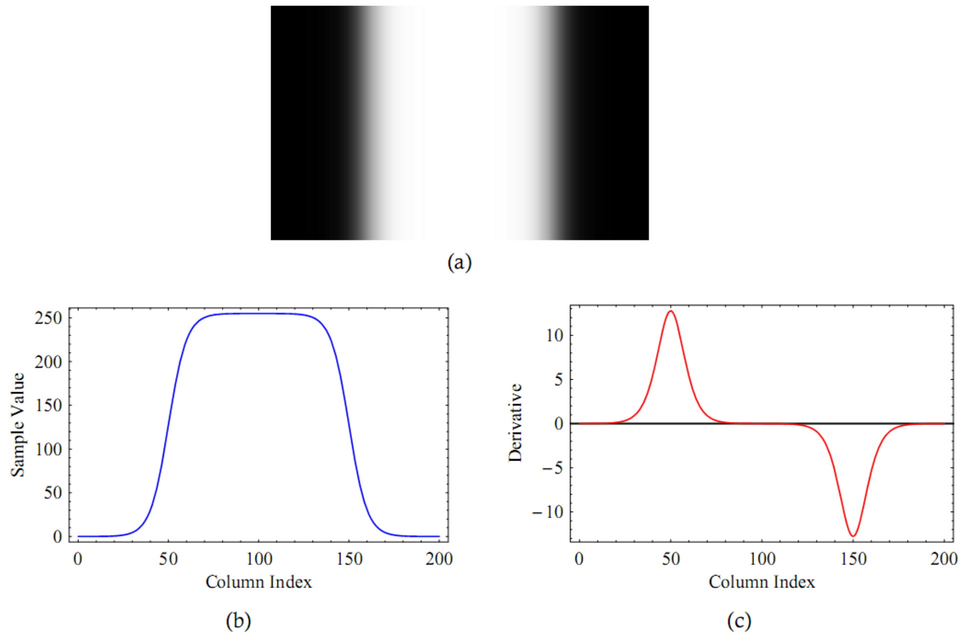


Figure 6.11. (a) A grayscale image with two edges, (b) row profile, and (c) first derivative.

(Review?) Derivatives via Finite Differences

- We can approximate the derivative with a kernel w:

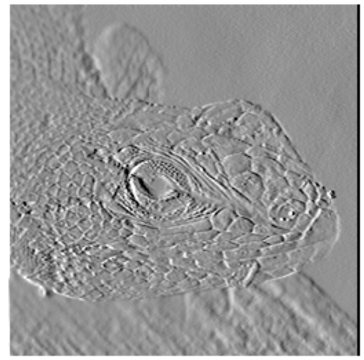
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+h,y)-f(x-h,y)}{2h} \approx \frac{f(x+1,y)-f(x-1,y)}{2}$$

$$\frac{\partial f}{\partial x} \approx w_{dx} \circ f \qquad w_{dx} = \begin{array}{|c|c|c|}\hline -\frac{1}{2} & 0 & \frac{1}{2} \\ \hline\end{array}$$

$$\frac{\partial f}{\partial y} \approx w_{dy} \circ f \qquad w_{dy} = \begin{array}{|c|}\hline -\frac{1}{2} \\ \hline 0 \\ \hline \frac{1}{2} \\ \hline\end{array}$$

Taking Derivatives with Convolution

$$\begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$$



$$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

Gradients with Finite Differences

- These partial derivatives approximate the image gradient, ∇I.
- Gradients are the unique direction where the image is changing the most rapidly, like a slope in high dimensions
- We can separate them into components kernels G_x, G_y. $\nabla I = (G_x, G_y)$

$$\nabla I(x,y) = \begin{pmatrix} \delta I(x,y)/\delta x \\ \delta I(x,y)/\delta y \end{pmatrix}.$$

$$G_x = [1, 0, -1] \qquad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix};$$

$$\nabla I = \begin{pmatrix} \delta I/\delta x \\ \delta I/\delta y \end{pmatrix} \simeq \begin{pmatrix} I \otimes G_x \\ I \otimes G_y \end{pmatrix}.$$

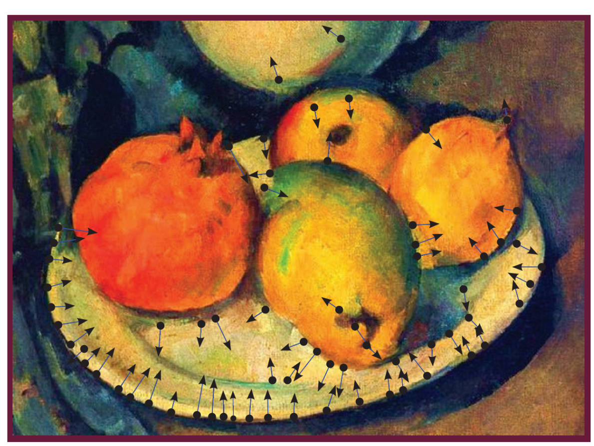
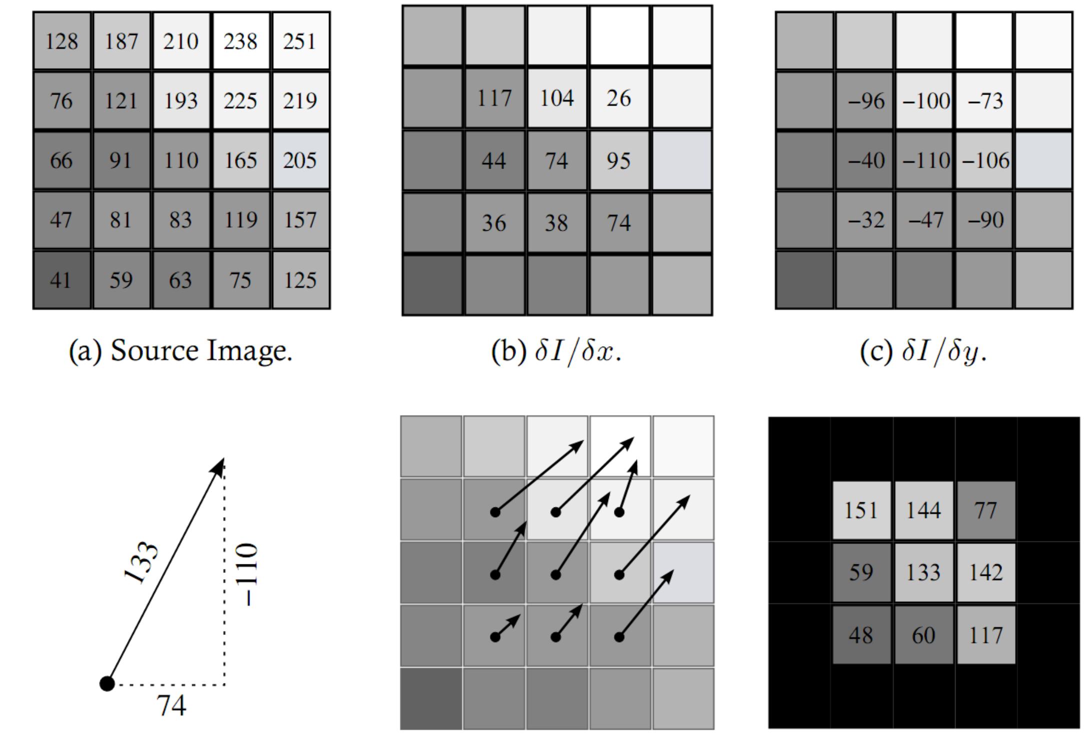


Figure 6.12. Image gradient (partial).

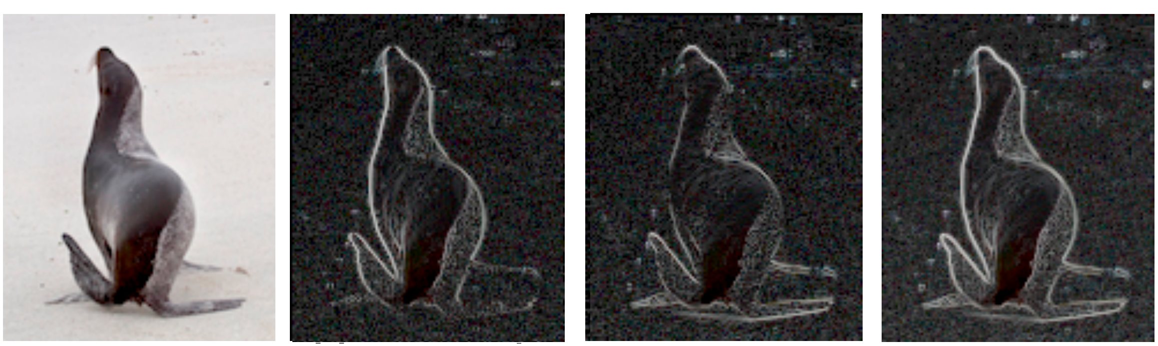


(a) Source Image. (b) $\delta I / \delta x$. (c) $\delta I / \delta y$.

(d) Center sample gradient. (e) Gradient. (f) Magnitude of gradient.

Figure 6.14. Numeric example of an image gradient.

Gradients G_x, G_y



$|G_x|$ $|G_y|$ $|G|$

$$|G| = \sqrt{(G_x^2 + G_y^2)}$$

Second Derivatives (Sharpening, almost)

- Partial derivatives in x and y lead to two kernels:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

and, similarly, in the y-direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

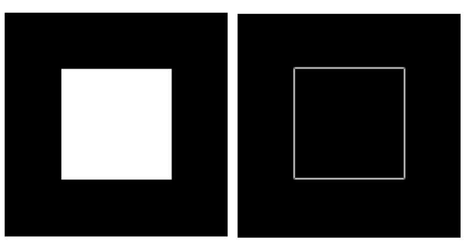
Compare with Sharpening filter: unbalanced counts!

$$\begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}$$

1	1	1
1	-9	1
1	1	1

Examples

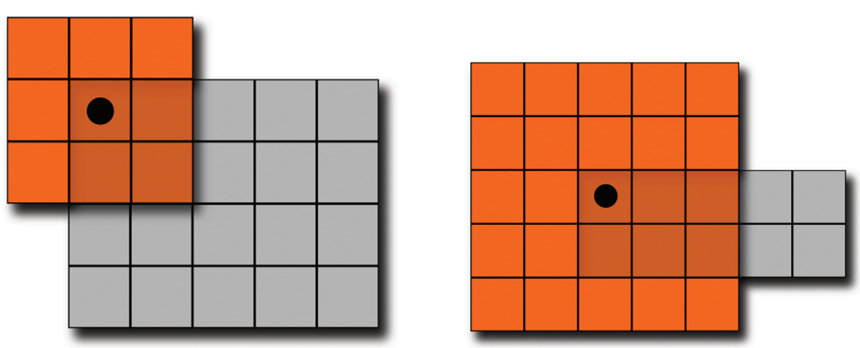
0	1	0
1	−4	1
0	1	0

a)  b)

Boundaries

Handling Image Boundaries

- What should be done if the kernel falls off of the boundary of the source image as shown in the illustrations below?



(a) Kernel at $I(0,0)$. (b) Kernel larger than the source.

Figure 6.4. Illustration of the edge handling problem.

Handling Image Boundaries

- When pixels are near the edge of the image, neighborhoods become tricky to define
- Choices:
  1. Shrink the output image (ignore pixels near the boundary)
  2. Expanding the input image (padding to create values near the boundary which are “meaningful”)
  3. Shrink the kernel (skip values that are outside the boundary, and reweigh accordingly)

Boundary Padding

- When one pads, they pretend the image is large and either produce a constant (e.g. zero), or use circular / reflected indexing to tile the image:

Figure 6.5. (a) Zero padding, (b) circular indexing, and (c) reflected indexing.