EECS 4422/5323 Computer Vision

Image Representation Lecture 2

Calden Wloka

16 September, 2019

Announcments

- Lab today and Wednesday: Template matching
- Think about your project
  - White paper due next Monday
  - Next lecture we will talk about ideas

Outline

- Recap
- Linear Filtering Continued
- Image Pyramids
- Non-linear Operators

Recap of Last Lecture

- Images are made up of pixels

Recap of Last Lecture

- Images are made up of pixels
  - Pixels have spatial location (x, y)

Recap of Last Lecture

- Images are made up of pixels
  - Pixels have spatial location (x, y)
  - Pixels have an associated value, *e.g.* intensity

Recap of Last Lecture

- Images are made up of pixels
  - Pixels have spatial location (x, y)
  - Pixels have an associated value, *e.g.* intensity
- Point operators map input pixel values to new output values

Recap of Last Lecture

- Images are made up of pixels
  - Pixels have spatial location (x, y)
  - Pixels have an associated value, *e.g.* intensity
- Point operators map input pixel values to new output values
- Neighbourhood operators take surrounding values into account, too

Recap of Last Lecture

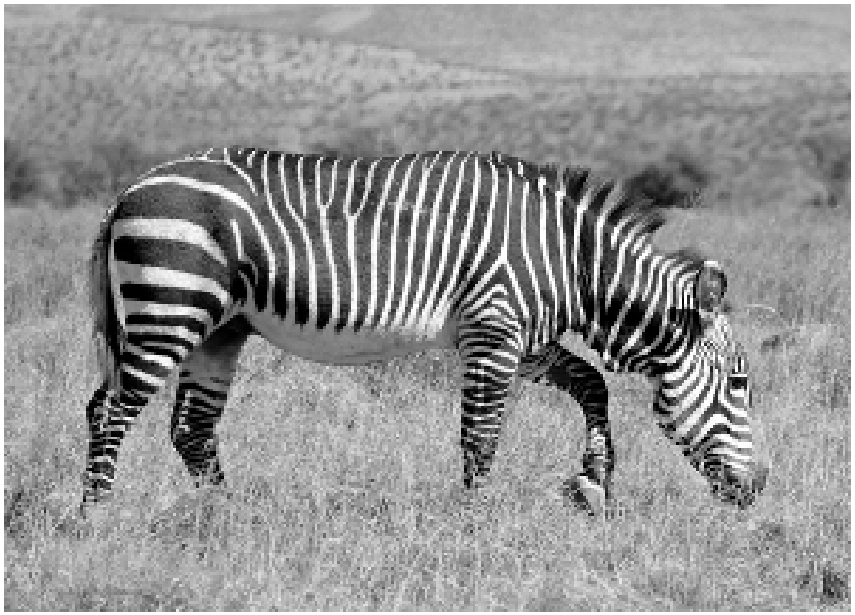
- Images are made up of pixels
  - Pixels have spatial location (x, y)
  - Pixels have an associated value, *e.g.* intensity
- Point operators map input pixel values to new output values
- Neighbourhood operators take surrounding values into account, too
  - Linear operators are weighted sums

Recap of Last Lecture

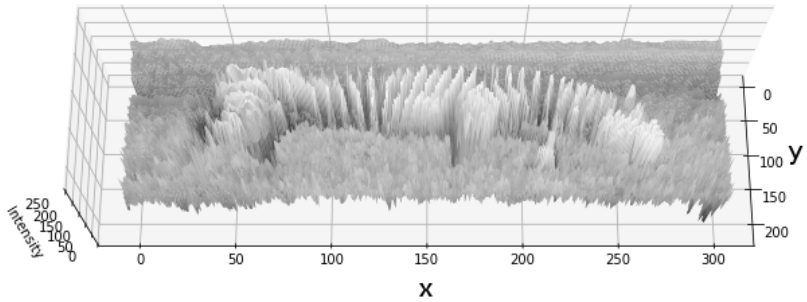
- Images are made up of pixels
  - Pixels have spatial location (x, y)
  - Pixels have an associated value, *e.g.* intensity
- Point operators map input pixel values to new output values
- Neighbourhood operators take surrounding values into account, too
  - Linear operators are weighted sums
  - Linear operator weights are stored in *kernels*

Picturing an Image as a Function

It is conceptually helpful to recognize that an image can be viewed as a (discrete) function in two spatial dimensions, forming a surface manifold.



Image



Manifold visualization

The Complexity of Filter Calculations

For a kernel K with dimensions $n \times n$ and an image F of size $w \times h$, how many operations do we need to perform to filter F by K?

The Complexity of Filter Calculations

For a kernel K with dimensions $n \times n$ and an image F of size $w \times h$, how many operations do we need to perform to filter F by K?

Each pixel requires n^2 operations and we have wh pixels, so our total number of filtering operations is whn^2.

The Complexity of Filter Calculations

For a kernel K with dimensions $n \times n$ and an image F of size $w \times h$, how many operations do we need to perform to filter F by K?

Each pixel requires n^2 operations and we have wh pixels, so our total number of filtering operations is whn^2.

For big kernels, large images, or many filters, this can become a significant strain on computational resources.

Separable Filters

Over the years a number of methods have been developed for reducing the computational cost of image filtering. One major approach is to take advantage of *separable filters*.

Separable Filters

Over the years a number of methods have been developed for reducing the computational cost of image filtering. One major approach is to take advantage of *separable filters*.

A filter K is *separable* if it can be broken into x and y components which may be applied independently.

$$K = K_y K_x$$

Separable Filters

If K is size $n \times n$, then both K_x and K_y contain n elements.

Separable Filters

If K is size $n \times n$, then both K_x and K_y contain n elements.

Applying K_x to F takes whn operations, and applying K_y to F takes whn operations, for a total of $2whn$

Separable Filters - Worked Example

Take a (reduced) example from previous class, first showing the computation using the 2D kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Kernel

18	27	54	9
45	27	72	36
63	63	27	54
45	72	27	36

Image patch

Separable Filters - Worked Example

Using zero padding, apply the kernel to the first element:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

0	0	0	0	0	0
0	18	27	54	9	0
0	45	27	72	36	0
0	63	63	27	54	0
0	45	72	27	36	0
0	0	0	0	0	0

Input patch

13	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

Output patch

$$\frac{0}{9}+\frac{0}{9}+\frac{0}{9}+\frac{0}{9}+\frac{18}{9}+\frac{27}{9}+\frac{0}{9}+\frac{45}{9}+\frac{27}{9}=2+3+5+3=13$$

Separable Filters - Worked Example

Apply the kernel to the second element:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

0	0	0	0	0	0
0	18	27	54	9	0
0	45	27	72	36	0
0	63	63	27	54	0
0	45	72	27	36	0
0	0	0	0	0	0

Input patch

13	27	-	-
-	-	-	-
-	-	-	-
-	-	-	-

Output patch

$$\frac{0}{9}+\frac{0}{9}+\frac{0}{9}+\frac{18}{9}+\frac{27}{9}+\frac{54}{9}+\frac{45}{9}+\frac{27}{9}+\frac{72}{9}=2+3+6+5+3+8=27$$

Separable Filters - Worked Example

And so on, until we compute our filtered patch:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

0	0	0	0	0	0
0	18	27	54	9	0
0	45	27	72	36	0
0	63	63	27	54	0
0	45	72	27	36	0
0	0	0	0	0	0

Input patch

13	27	25	19
27	44	41	28
35	49	46	28
27	33	31	16

Output patch

Recommended Exercise: Make sure you understand how to fill in the full output patch on your own.

Separable Filters - Worked Example

Note that:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = (\frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix})(\frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix})$$

So our kernel can be written as:

$$K_x = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \qquad K_y = \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

Separable Filters - Worked Example

We take our input patch and apply K_x, then apply K_y to the output of that calculation.

0	0	0	0	0	0
0	18	27	54	9	0
0	45	27	72	36	0
0	63	63	27	54	0
0	45	72	27	36	0
0	0	0	0	0	0

Input patch

0	0	0	0
15	33	30	21
24	48	45	36
42	51	48	27
39	48	45	21
0	0	0	0

After K_x

13	27	25	19
27	44	41	28
35	49	46	28
27	33	31	16

After K_y

Designing Filters

Filters are used for many different types of computation over images, and we often want to be able to adjust the behaviour of a filter to accomplish a specific goal.

Designing Filters

Filters are used for many different types of computation over images, and we often want to be able to adjust the behaviour of a filter to accomplish a specific goal.

Many common filters belong to a larger filter class which allows us to compute the specific desired filter based on a selection of parameters.

Gaussian Filters

The Gaussian function forms the basis for a wide range of image filters.

- Long history of use in signal processing

Gaussian Filters

The Gaussian function forms the basis for a wide range of image filters.

- Long history of use in signal processing
- Nice mathematical properties (including separability!)

Gaussian Filters

The Gaussian function forms the basis for a wide range of image filters.

- Long history of use in signal processing
- Nice mathematical properties (including separability!)
- Can be parametrized by kernel dimensions and σ

Gaussian Filters

The Gaussian function forms the basis for a wide range of image filters.

- Long history of use in signal processing
- Nice mathematical properties (including separability!)
- Can be parametrized by kernel dimensions and σ

Go to the Gaussian Filter section of the Steerable Filters demo.

Convolution vs. Cross-Correlation

There are two major styles of filter operation in images: *convolution* and *cross-correlation*.

For a kernel K of dimensions $2k+1 \times 2k+1$, an image F, and output G, we compute **cross-correlation** as:

$$G_{cc}(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} K(i,j)F(x+i,y+j)$$

which is often written as:

$$G_{cc} = K \otimes F$$

Convolution vs. Cross-Correlation

There are two major styles of filter operation in images: *convolution* and *cross-correlation*.

For a kernel K of dimensions $2k+1 \times 2k+1$, an image F, and output G, we compute **convolution** as:

$$G_c(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} K(i,j)F(x-i,y-j)$$

which is often written as:

$$G_c = K * F$$

Why does it matter?

Up until now we have been relatively non-specific about which type of filtering we use (though the order of our worked example calculations suggests correlation). This ambiguity is possible for *symmetric* filters, because then the correlation and convolution results are identical.

Why does it matter?

Up until now we have been relatively non-specific about which type of filtering we use (though the order of our worked example calculations suggests correlation). This ambiguity is possible for *symmetric* filters, because then the correlation and convolution results are identical.

What sorts of filters might not be symmetric?

Why does it matter?

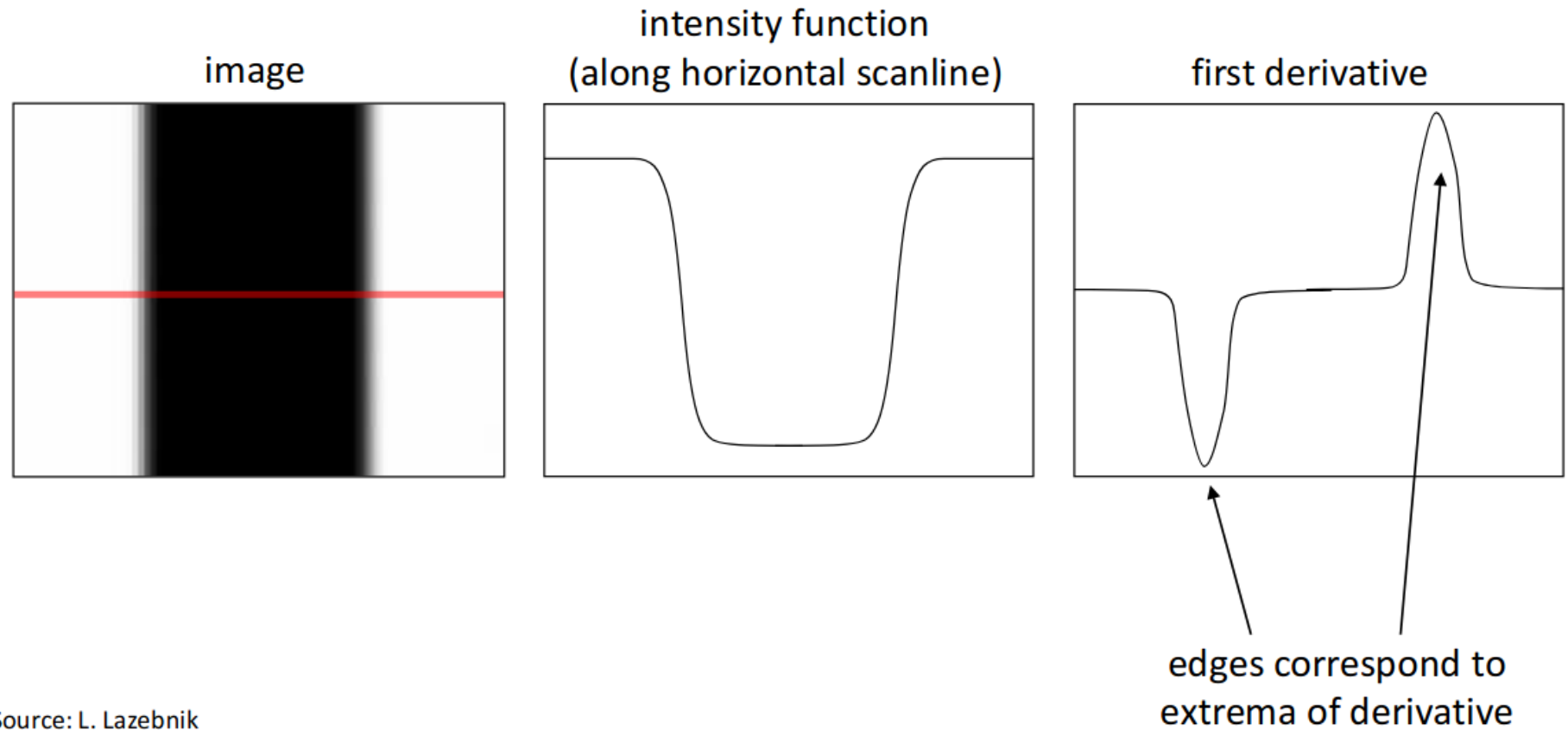
Up until now we have been relatively non-specific about which type of filtering we use (though the order of our worked example calculations suggests correlation). This ambiguity is possible for *symmetric* filters, because then the correlation and convolution results are identical.

What sorts of filters might not be symmetric?

Next several slides adapted from Noah Snavely's Computer Vision slides.

Edges

- An edge is a place of rapid change in the image intensity function



Source: L. Lazebnik

Differentiation in a Digital Image

Digital images can be treated as discrete functions. When dealing with a discrete function, it is common to use *finite differences* to approximate a differential.

For an image $F(x, y)$, we can find the differential in the x direction by:

$$\frac{\partial}{\partial x} F(x, y) \approx F(x + 1, y) - F(x, y)$$

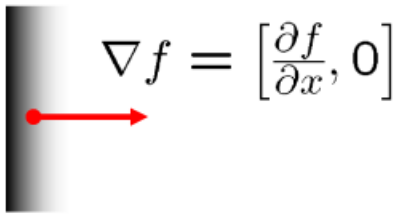
and in the y direction:

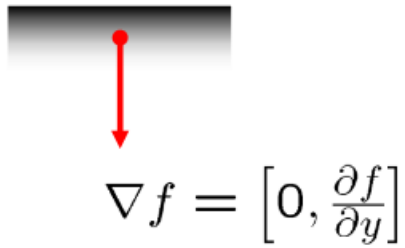
$$\frac{\partial}{\partial y} F(x, y) \approx F(x, y + 1) - F(x, y)$$

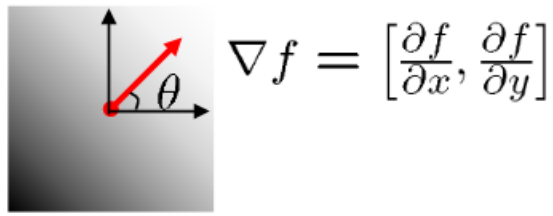
Image Gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The gradient points in the direction of most rapid increase in intensity







The *edge strength* is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

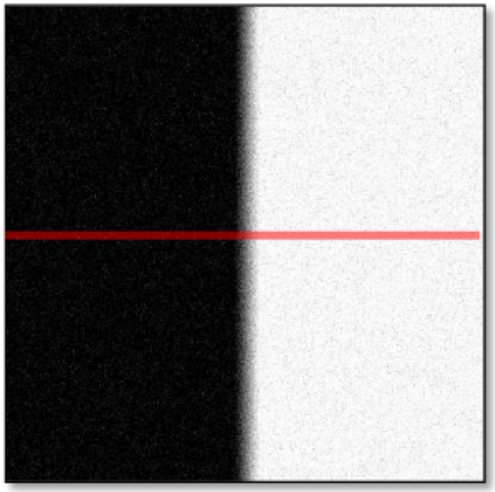
The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

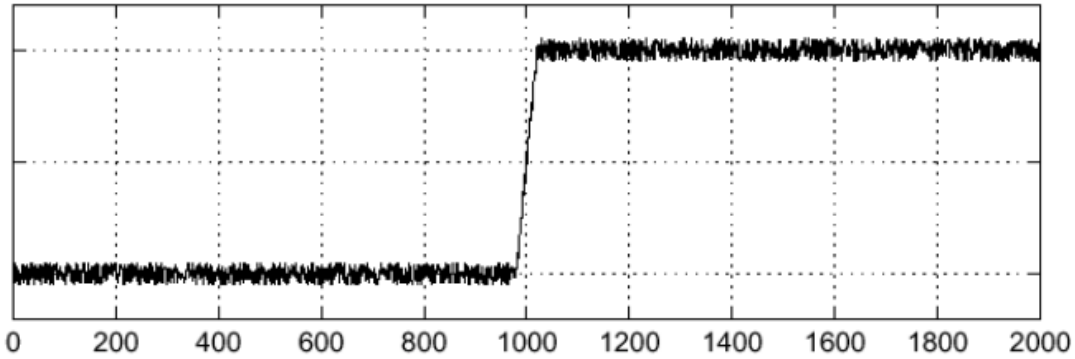
Source: Steve Seitz

The Picture Gets Complicated by Noise

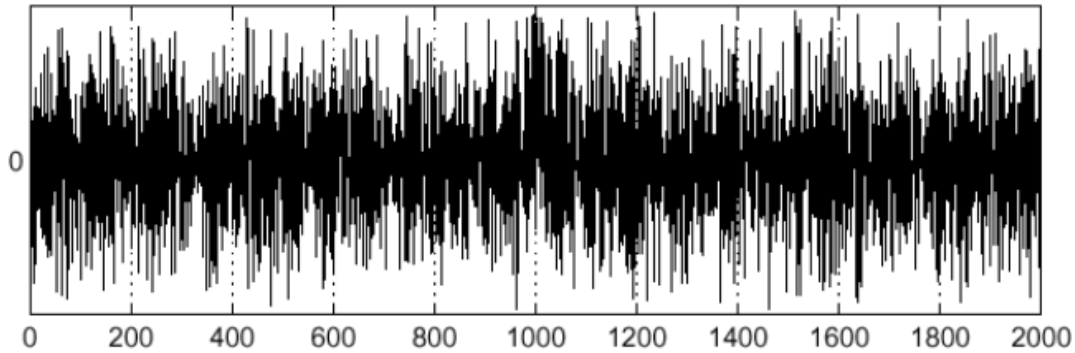


Noisy input image

$f(x)$



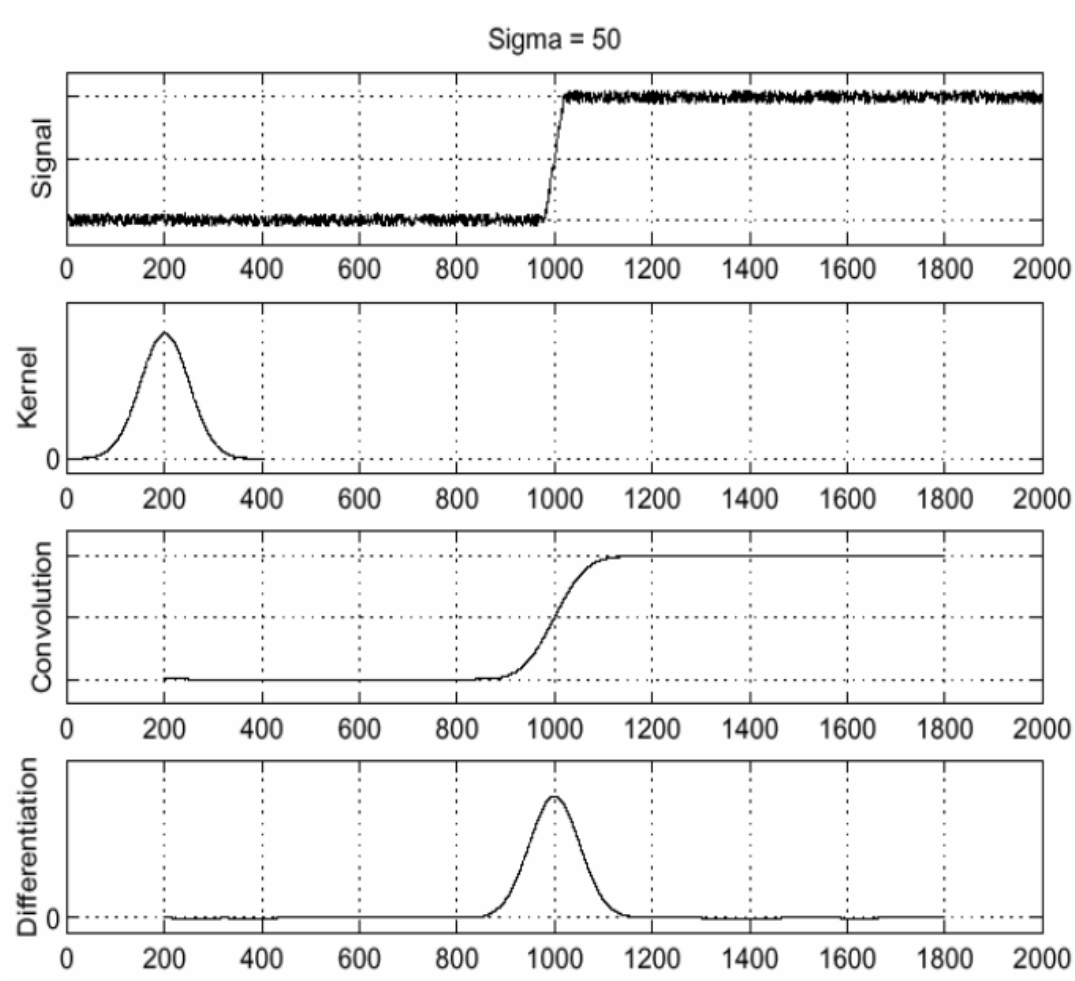
$\frac{d}{dx}f(x)$



Where is the edge?

Source: S. Seitz

Expanding Our Context (*e.g.* with Smoothing) Can Help



To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

Convolution is Commutative

Convolution is commutative, and our finite differences equation is a convolution, so we can combine the smoothing and convolution into one step.

In other words, an effective edge detector can be constructed by taking the differential of a Gaussian.

But Edges Have Orientations

The specific orientation of an edge and which side is light and which side dark carries useful information.

Being able to turn a filter to reflect an arbitrary orientation is known as *steering* a filter.

But Edges Have Orientations

The specific orientation of an edge and which side is light and which side dark carries useful information.

Being able to turn a filter to reflect an arbitrary orientation is known as *steering* a filter.

Go to steerable filter demo.

Sums over Filter Weights

You may have noticed that most example filters presented so far have one of two possible sums: 1 or 0.

Sums over Filter Weights

You may have noticed that most example filters presented so far have one of two possible sums: 1 or 0.

- Filters which sum to 1 tend to (mostly) recreate the original image
  - If a filter sums to more than 1 or less than 0, you increase your chance of saturation

Sums over Filter Weights

You may have noticed that most example filters presented so far have one of two possible sums: 1 or 0.

- Filters which sum to 1 tend to (mostly) recreate the original image
  - If a filter sums to more than 1 or less than 0, you increase your chance of saturation
- Filters which sum to 0 tend to look for a specific pattern or feature
  - If a filter sums to more or less than 0, you will respond to light or dark regions, respectively, independent of the actual pattern

How do we choose a kernel size?

For many of the filters we've already discussed (*e.g.* smoothing filters), the size of the kernel really just dictates how great an effect the filter has (although this in many ways still depends on the relative side of the filter and the image).

How do we choose a kernel size?

For many of the filters we've already discussed (*e.g.* smoothing filters), the size of the kernel really just dictates how great an effect the filter has (although this in many ways still depends on the relative side of the filter and the image).

For more complex filters, (*e.g.* looking for a specific pattern), the ability of the filter to identify its target feature greatly depends on how well its size matches the image size of its target.

Humans and Scale

Human observers are remarkably robust to the spatial scale of objects.

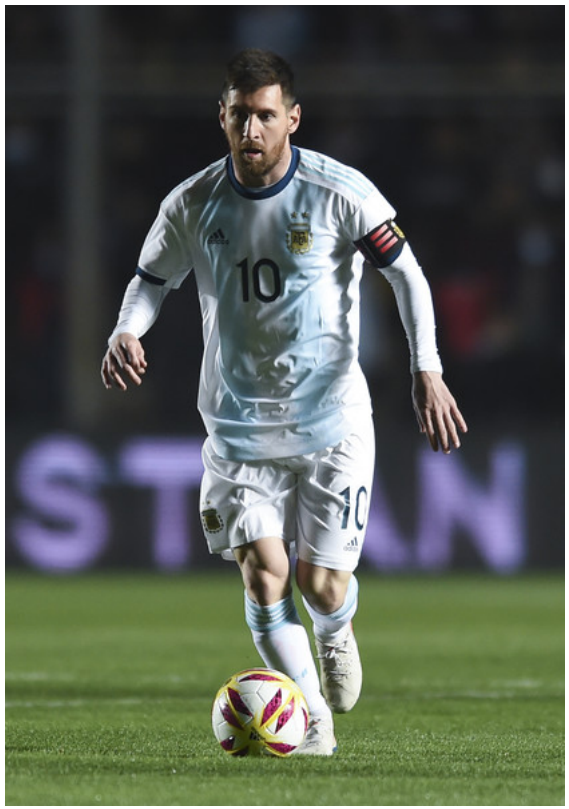
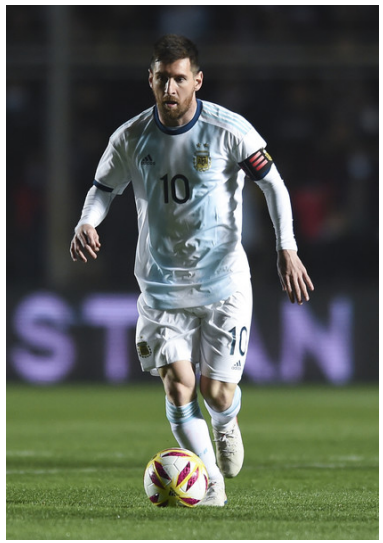
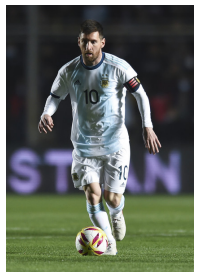


Image Source: Zimbio.com





Example: Template Matching

Imagine we want to identify all occurences of the letter 'E' in an eye chart:

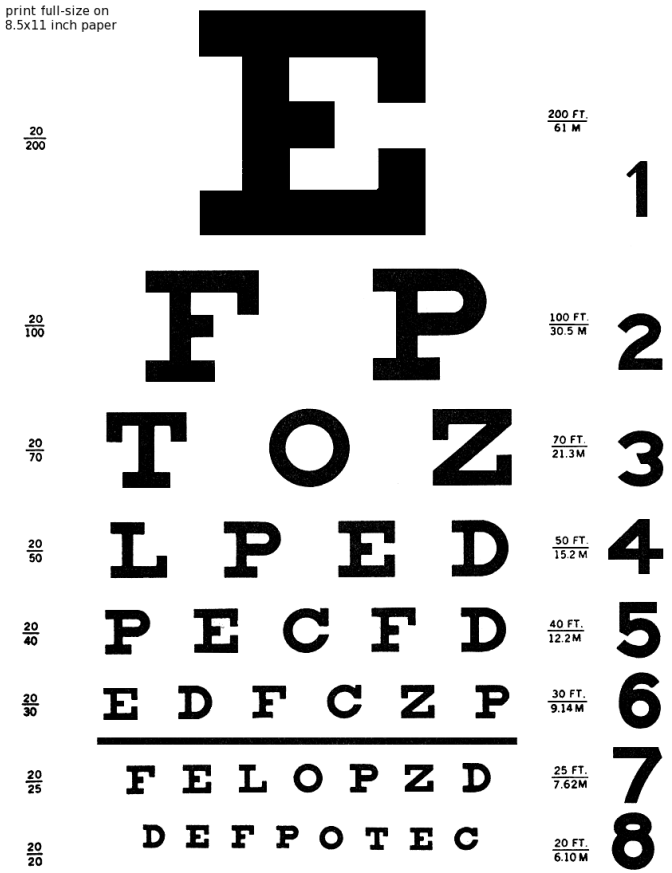
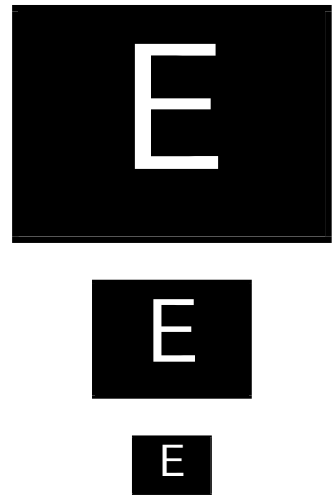


Image Source: Original source unknown



Which kernel should we use?

Example: Template Matching

We could build a whole bank of filter kernels at different sizes (*scales*) and say that we find an 'E' any time one of them yields a sufficiently high value, but this could get costly for very large kernels.

Example: Template Matching

We could build a whole bank of filter kernels at different sizes (*scales*) and say that we find an 'E' any time one of them yields a sufficiently high value, but this could get costly for very large kernels.

A more efficient option is to use a smaller kernel, but run it over different sizes of the image.

Image Pyramids

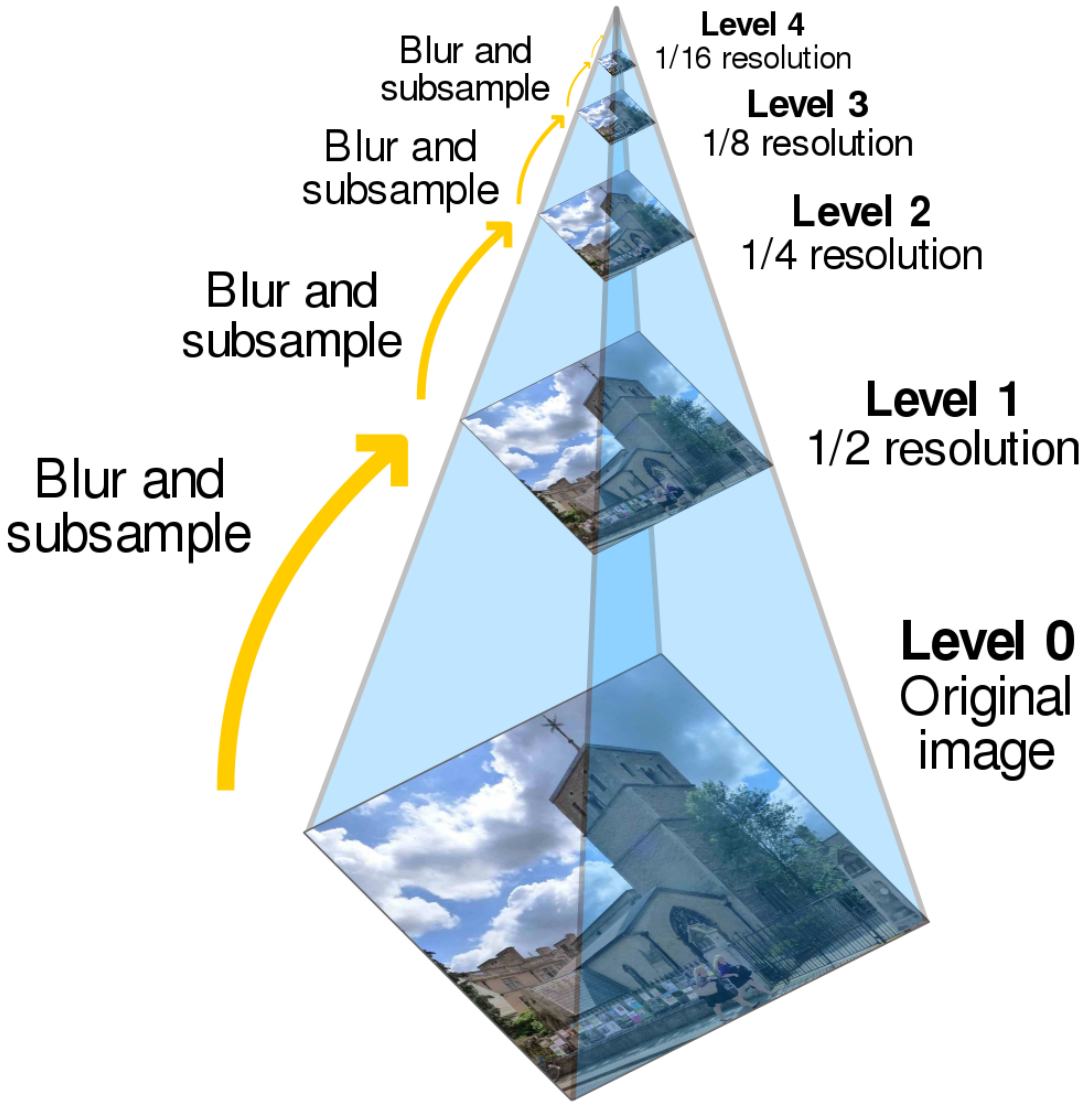


Image Source: Wikipedia

An image pyramid is a common multiscale image representation.

- Typically formed at half resolution steps (*Why?*)

Image Pyramids

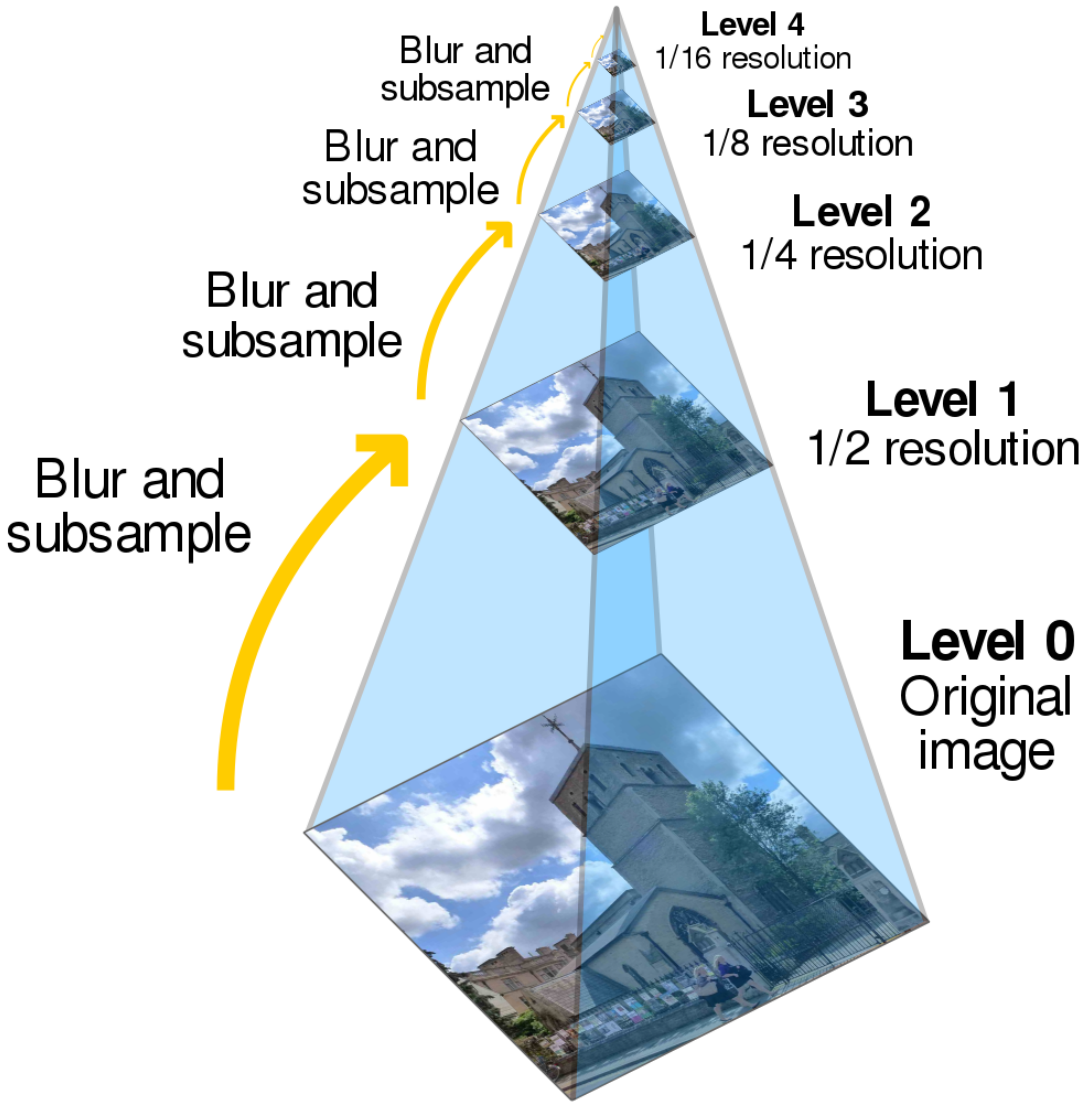


Image Source: Wikipedia

An image pyramid is a common multiscale image representation.

- Typically formed at half resolution steps (*Why?*)
- Downsampling usually involves a blurring step

Image Pyramids

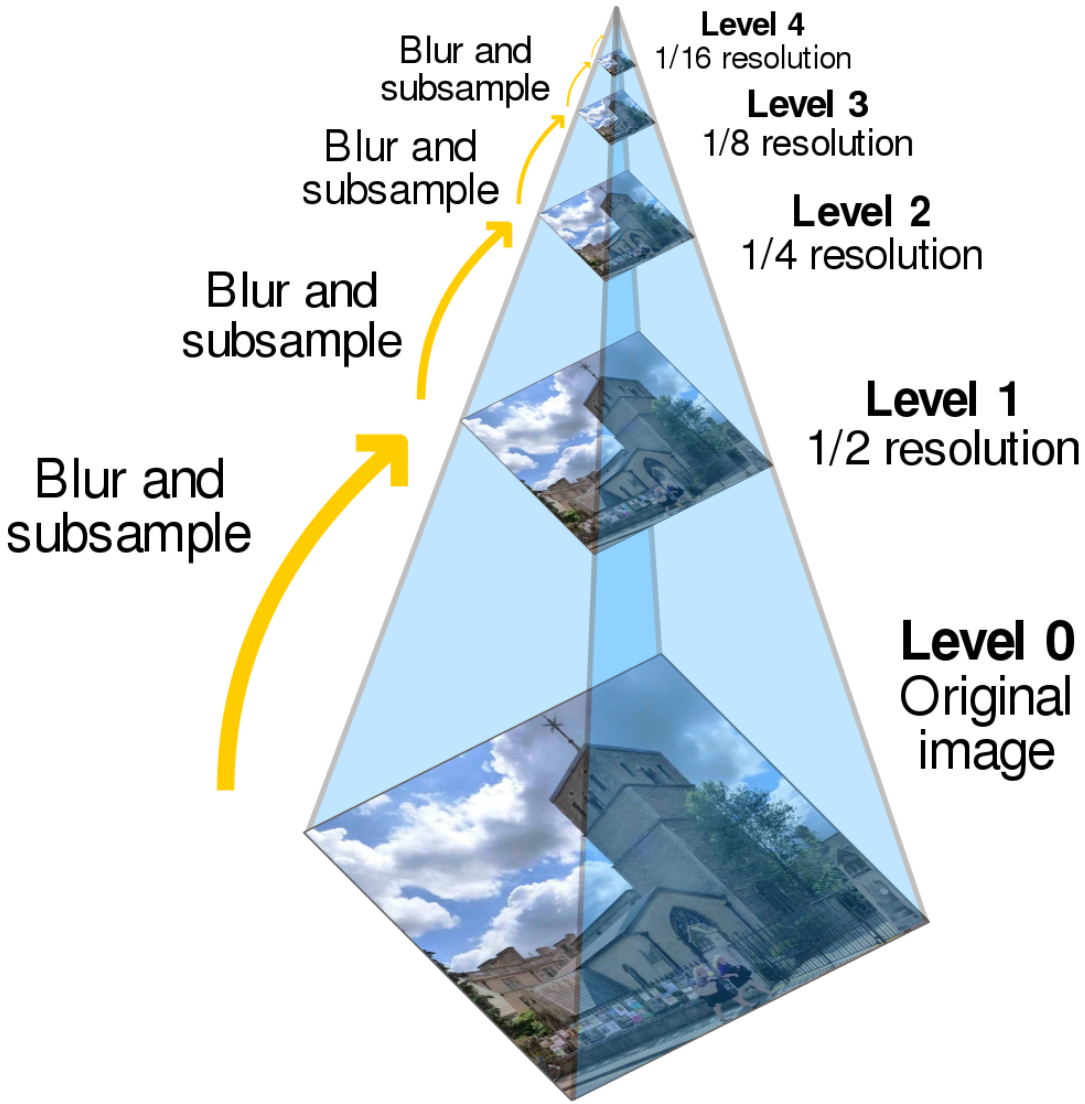


Image Source: Wikipedia

An image pyramid is a common multiscale image representation.

- Typically formed at half resolution steps (*Why?*)
- Downsampling usually involves a blurring step
- Taking the difference between pyramid layers often yields interesting structural features

Rules Based Filtering

Sometimes we don't want to treat all pixels as equivalently valid, and instead want to apply nonlinear rule.

Rules Based Filtering

Sometimes we don't want to treat all pixels as equivalently valid, and instead want to apply nonlinear rule.

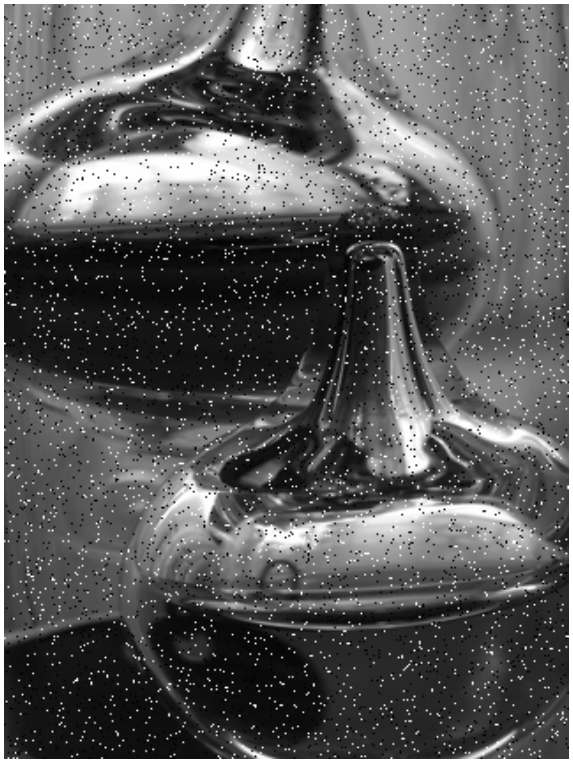


Image Source: Wikipedia

"Salt and pepper" noise is a common form of image degredation, and blurring will incorporate the unwanted noise into the surrounding pixels.

Rules Based Filtering

Sometimes we don't want to treat all pixels as equivalently valid, and instead want to apply nonlinear rule.

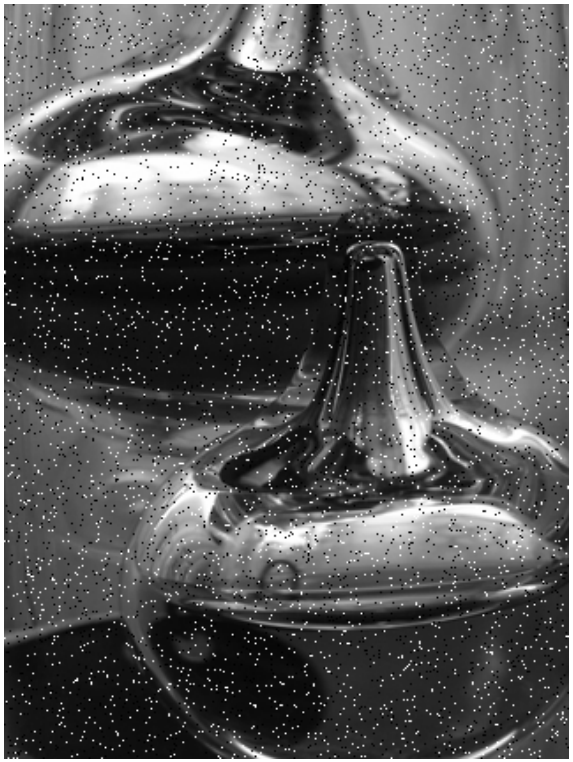


Image Source: Wikipedia

"Salt and pepper" noise is a common form of image degredation, and blurring will incorporate the unwanted noise into the surrounding pixels.

Taking the median pixel in a neighbourhood will more effectively remove the isolated noise peaks.

Morphology

Another class of non-linear filter which is a very useful tool to have tucked into your image processing toolkit are *morphology* tools.

Morphology

Another class of non-linear filter which is a very useful tool to have tucked into your image processing toolkit are *morphology* tools.

Morphology tools are typically applied to binary images, making them very useful for cleaning up and manipulating masks.

Morphology Examples - Erosion

Erosion uses a *structuring element* in a kernel with the rule that a pixel is set to 1 only if *all* elements under the structuring element are equal to 1, otherwise it is set to 0.

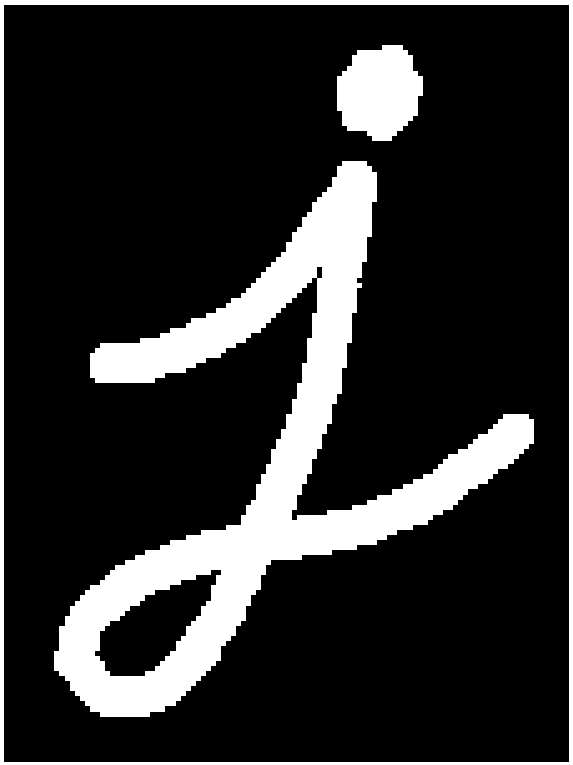


Image Source: OpenCV Morphology Tutorial

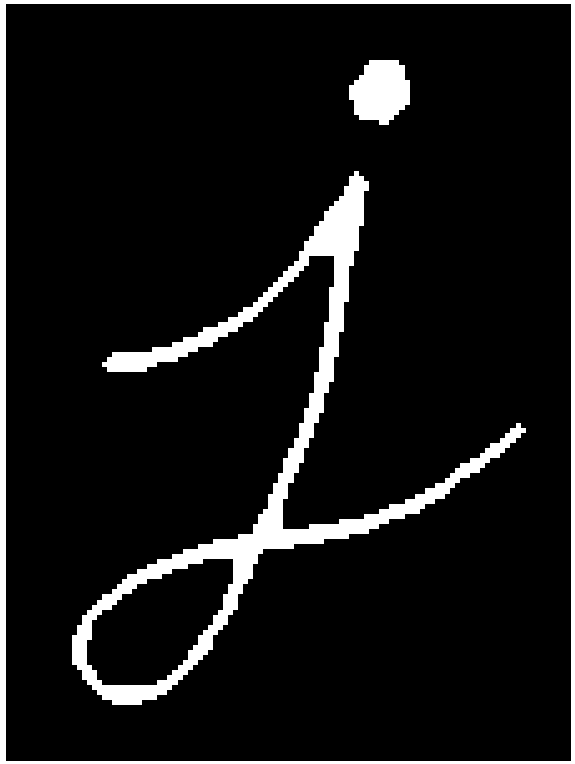


Image Source: OpenCV Morphology Tutorial

Morphology Examples - Dilation

Dilation uses a *structuring element* in a kernel with the rule that a pixel is set to 1 if *any* elements under the structuring element are equal to 1, otherwise it is set to 0.

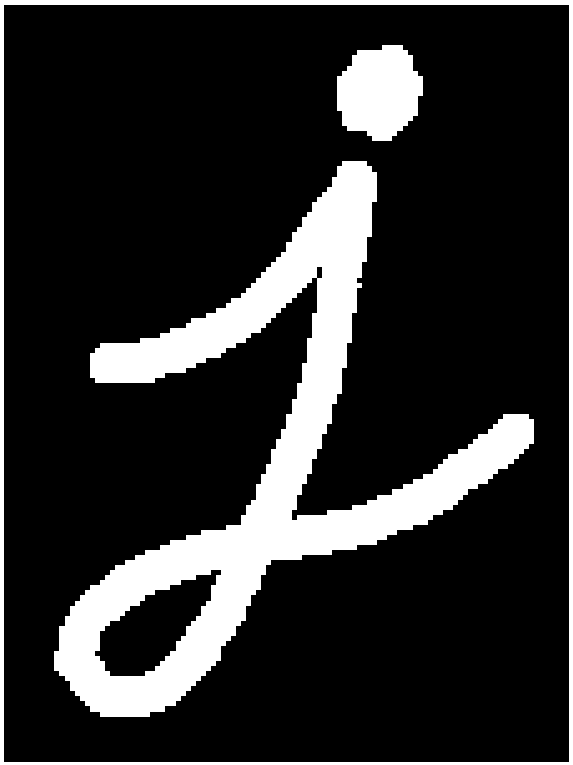


Image Source: OpenCV Morphology Tutorial

Image Source: OpenCV Morphology Tutorial

Morphology Examples - Closing

When dilation is followed by erosion, it is referred to as *closing*.

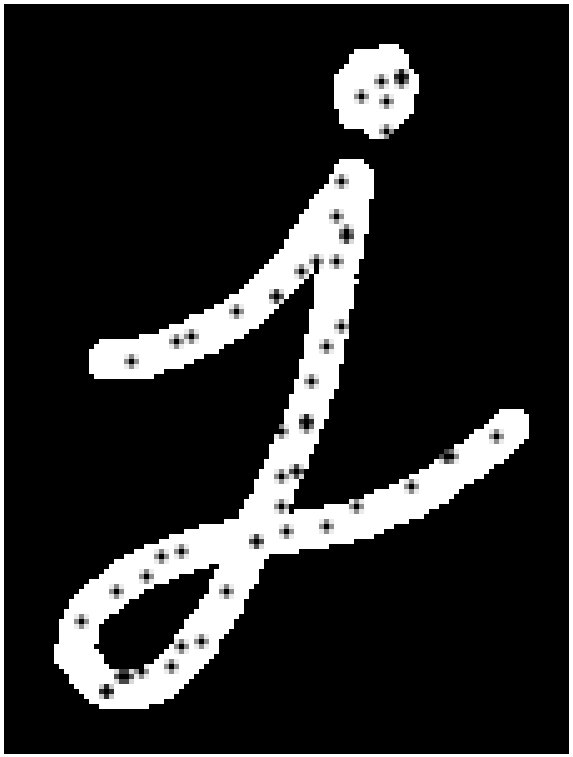


Image Source: OpenCV Morphology Tutorial

Image Source: OpenCV Morphology Tutorial

Morphology Examples - Opening

When erosion is followed by dilation, it is referred to as *opening*.

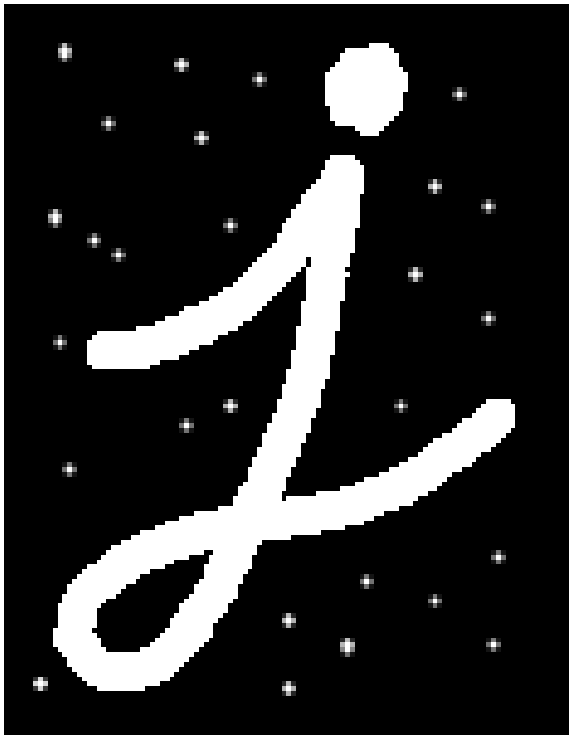


Image Source: OpenCV Morphology Tutorial

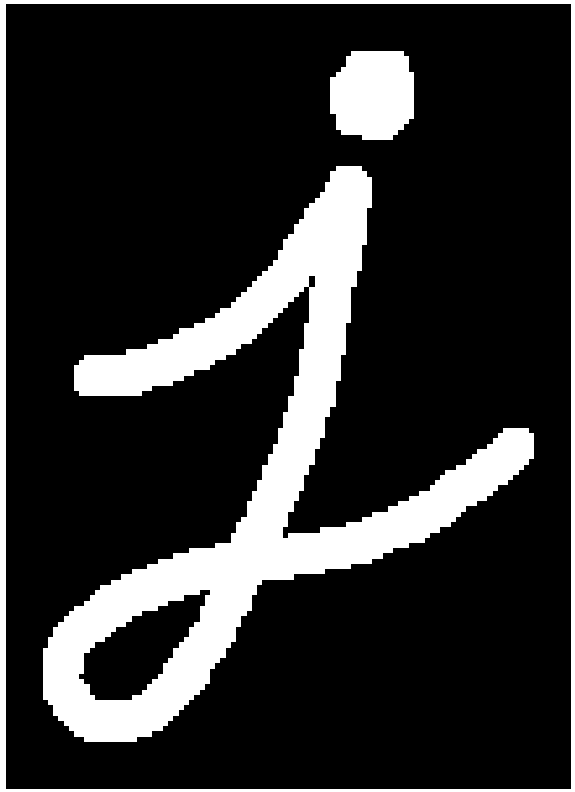


Image Source: OpenCV Morphology Tutorial

Morphology Application Demo

Go to Morphology Demo.