Zoom, Enhance!

What is edge detection?

Process a 2D image to find boundaries between regions. One possibility is to detect steep changes in pixel values by computing some sort of gradient.

A simple gradient from one pixel to the next is more sensitive to noise than a gradient across a larger window → convolve a kernel and compute a metric

$$Edge(x,y) = \sum_m \sum_n Kernel(m,n)\, Image(x-m, y-n)$$

Common methods (kernels, metrics) shown in next slides.

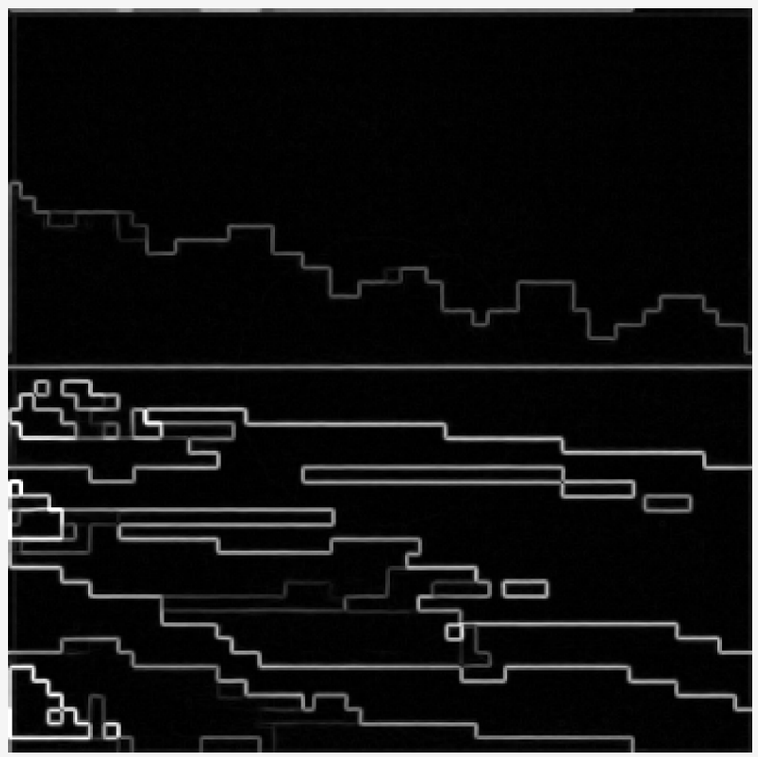
Each method has strengths and weaknesses (different performance for horizontal edge detection, noisy images, etc.)

Or invent your own method using a larger kernel.

Test images included in git folder (text and jpg versions).

Raw image

Edge detection result



What is Gaussian blur?

Average-out noise by summing neighboring pixel values using Gaussian weights (Fourier transforming a Gaussian gives you a low-pass filter in frequency space, which removes high frequency noise)

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

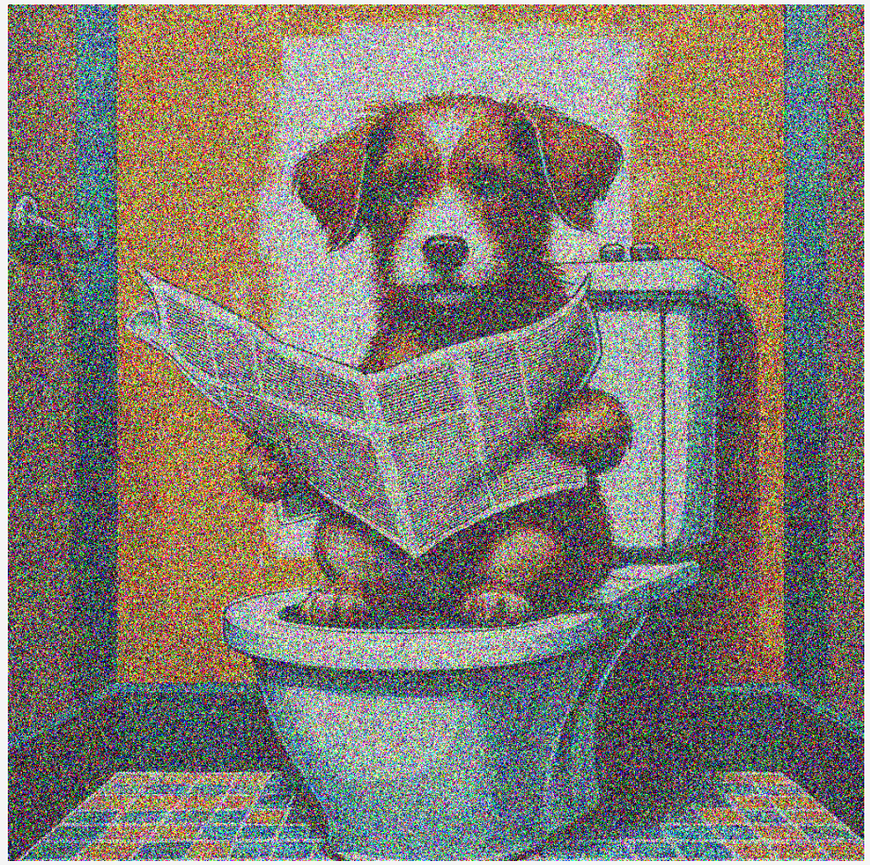
Width controls the strength of the blur.

$$\text{Blurred}(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} G(i, j) \cdot \text{Image}(x + i, y + j)$$

Two independent 1D convolutions.

Test data in git folder.

Raw image



Blurred image

Roberts gradient edge detection

```
(* Roberts Cross-Gradient*)
robertsEdgeDetection[image_] := Module[{gx, gy, magnitude}, (*Roberts kernels (2x2)*)
   robertsX = {{1, 0}, {0, -1}};
   robertsY = {{0, 1}, {-1, 0}};
   {gx, gy} = ParallelMap[convolve2D[image, #] &, {robertsX, robertsY}];
   magnitude = Sqrt[gx^2 + gy^2];
   <|"Magnitude" → magnitude, "Gx" → gx, "Gy" → gy|>];
```

Gradient 2x2 kernels separately in the X and Y directions.

- Kernel convolution gives one quantity for each kernel (gx, gy)
- The sum in quadrature of gx and gy (magnitude) is a measure of edge strength
- The components gx and gy are a measure of the edge direction.
- Simple to implement but small kernel increases sensitivity to noise.

Prewitt edge detection

```
(* Prewitt Edge Detection*)
prewittEdgeDetection[image_] := Module[{gx, gy, magnitude}, (*Prewitt kernels*)
   prewittX = {{-1, 0, 1}, {-1, 0, 1}, {-1, 0, 1}};
   prewittY = {{-1, -1, -1}, {0, 0, 0}, {1, 1, 1}};
   {gx, gy} = ParallelMap[convolve2D[image, #] &, {prewittX, prewittY}];
   magnitude = Sqrt[gx^2 + gy^2];
   <|"Magnitude" → magnitude, "Gx" → gx, "Gy" → gy|>];
```

Gradient kernels separately in the X and Y directions.

- Kernel convolution gives one quantity for each kernel (gx, gy)
- The sum in quadrature of gx and gy (magnitude) is a measure of edge strength
- The components gx and gy are a measure of the edge direction.

Sobel edge detection

```
(* Sobel Edge Detection*)
sobelEdgeDetection[image_] := Module[{gx, gy, magnitude, direction}, (*Sobel kernels*)
    sobelX = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
    sobelY = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};
    (*Compute gradients in parallel*){gx, gy} = ParallelMap[convolve2D[image, #] &, {sobelX, sobelY}];
    (*Vectorized magnitude and direction computation*)magnitude = Sqrt[gx^2 + gy^2];
    direction = ArcTan[gx, gy];
    <|"Magnitude" → magnitude, "Direction" → direction, "Gx" → gx, "Gy" → gy|>];
```

Combine a gradient kernel in the X direction with Gaussian smoothing kernel in Y.

Less sensitive to some kinds of noise.

Laplacian edge detection

$\nabla^2 \text{Image} = \partial^2 \text{Image}/\partial x^2 + \partial^2 \text{Image}/\partial y^2$

```
(* Laplacian Edge Detection*)
laplacianEdgeDetection[image_] := Module[{result}, (*Laplacian kernel (detects edges in all directions)*)
    laplacianKernel = {{0, -1, 0}, {-1, 4, -1}, {0, -1, 0}};
    result = convolve2D[image, laplacianKernel];
    <|"Magnitude" → result|>];
```

Rotationally invariant edge measure.