Scale Space Edge Detection

Nick Draper, Jonathan Hayase
Seminar in Differential Geometry
Harvey Mudd College

Abstract

Scale space representation is the idea that a two dimensional image can be represented by a collection of smoothed images. This paper documents how using such a representation can be useful for detecting edges in an image. The scale space allows for a classification of how strong different edges are in the image from very fine to coarse ones.

1 Edge Detection Background

Detecting edges in images has long been a problem with many uniques solutions to approach it. Generally speaking, the majority of algorithms are usually checking the image for some of the following features:

- large discontinuities in luminance values
- discontinuities in different object orientations
- large discontinuities in the intensity gradient

Some of the common methods for edge detection include the Sobel, Canny, Prewitt, Roberts, and Fuzzy Logic algorithms. However, these methods do not yield a lot of imformation with regards to the strength of the edges detected. The majority of these algorithms will usually convolve the image with a static matrix to caclulate edges and does adapt enough to the image.

This is why for our edge detection method, we will be using the scale space approach. The benefit of using a scale space appraoch for edge detection, is we have the ability to classify the strength of the edges in the image. This allows for a range of edges from very large immediate changes in intensity to very gradual.

2 Scale Space and Its Derivatives

To understand how exactly we detect images in the scale space, we must first define what the scale space is. If we have a continuous function of multiple variables such as $f(x, y)$, then we define the scale space representation of such a function as

$$L(x; t) = g(x, y; t) * f(x, y) \tag{1}$$

Here t represents the scale parameter, and can be thought of how much smoothing is applied to the function. The function g is the Gaussian kernel given by

$$g(x, y; t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/(2t)} \tag{2}$$

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

With the scale space representation defined, we can now take derivatives of it as it is a continuous well-defined function. Spatial derivatives are relatively simple being defined as the following

$$L_{x^\alpha y^\beta}(\cdot;t) = \partial_{x^\alpha y^\beta} L(\cdot;t) = g_{x^\alpha y^\beta}(\cdot;t) * f(\cdot) \tag{3}$$

However, when taking the partial derivative with respect to the scale t, it becomes more interesting. The scale space representation collection is a solution for the diffusion equation. Therefore is has the useful property of

$$\partial_t L = \frac{1}{2}\nabla^2 L = \frac{1}{2}(\partial_{xx} + \partial_{yy})L \tag{4}$$

with the inital condition of $L(x, y; 0) = f(x, y)$. So now, scale derivatives can be represented as spatial derivatives.

Now all these representations and operators are useful for continuous functions, but the images we deal with are discrete and contain quantized inetsnity values. So we must now understand how these operations and properties apply to the discrete domain.

The scale space representation is still defined in a similar fashion. The following is the discrete version of the scale space operationg on the function $f(x)$, which only has a single spatial variable.

$$L(x;t) = (T(\cdot;t) * f(\cdot))(x;t) \tag{5}$$

In this expression, T represents the discrete version of the Gaussian kernel and is further evaluated as

$$T(n;t) = e^{-t} I_n(t) \tag{6}$$

where I_n is the modified Bessel functions of integer order given by

$$I_n(x) = i^{-\alpha} J_\alpha(ix) = \sum_{m=0}^{\infty} \frac{1}{m!\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha} \tag{7}$$

Now that the one dimensional case is understood for the scale space, we can expand this to two dimensions. After all, images are compsed of two dimesnisons, so it makes sense that these operators can act on two dimensional functions. The two dimensional scale space representation for discrete variables is given by the follwoing

$$L(x, y; t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T(m;t)T(n;t)f(x-m, y-n) \tag{8}$$

Even with the discrete case, the scale space representation must still satisfy the semidiscretized version of the diffusion equation. Therfore by taking a scale derivative of the function we must have the following

$$\partial_t L = \frac{1}{2}((1-\gamma)\nabla_5^2 L + \gamma \nabla_\times^2 L) \tag{9}$$

where $\gamma \in [0, 1]$ is a hyperparameter and,

$$(\nabla_5^2 f)_{0,0} = f_{-1,0} + f_{+1,0} + f_{0,-1} + f_{0,+1} - 4f_{0,0} \tag{10}$$

$$(\nabla_\times^2 f)_{0,0} = \frac{1}{2}(f_{-1,-1} + f_{-1,+1} + f_{+1,-1} + f_{+1,+1} - 4f_{0,0}) \tag{11}$$

Note that $f_{-1,1}$ represents $f(x-1, y+1)$. Now that we have defined what discrete scale spaces and their derivatives look like, we can start to define what an edge is.

3 Defining an Edge in Scale Space

We classify edge points as points which have gradient magnitueds that assume a maximum in the direction of the gradient. Let us denote the magnitude of the gradient of L as L_g. Then we define an edge in the scale space with the following conditions

$$\begin{aligned} L_{gg} &= 0 \\ L_{ggg} &< 0 \end{aligned} \tag{12}$$

The way we can then represent these conditions in terms of spatial derivatives is as follows

$$\begin{aligned} L_{gg} &= L_x^2 L_{xx} + 2L_x L_y L_{xy} + L_y^2 L_{yy} = 0 \\ L_{ggg} &= L_x^3 L_{xxx} + 3L_x^2 L_y L_{xxy} + 3L_x L_y^2 L_{xyy} + L_y^3 L_{yyy} < 0 \end{aligned} \tag{13}$$

This is useful for determining edges at a single scale, but if we are to determine edges over multiple scales, we must also develop an edge strength metric, $\varepsilon_{norm} L$. Thus this adds two more conditions that must be satisified for an edge to be classified in the scale space.

$$\begin{aligned} \partial_t(\varepsilon_{norm} L(x, y; t)) &= 0 \\ \partial_{tt}(\varepsilon_{norm} L(x, y; t)) &< 0 \end{aligned} \tag{14}$$

Then equation 12 in conjunction with 14 form the neccessary constraints for us to define an edge in the scale space. Now we must define what we mean by edge strength and give a clearer idea of ε_{norm}.

The approach we took was to let our edge strength meteric be defined by the gradient magnitude that has been normalized by our γ factor. In this case we can define it as

$$\begin{aligned} G_\gamma L &= L_{g,\gamma}^2 \\ &= t^\gamma (L_x^2 + L_y^2) \end{aligned} \tag{15}$$

Then the strength of the edges is found by computing the path integral over the maximal connected edge Γ given by

$$G(\Gamma) = \int_{(x;t)\in\Gamma} \sqrt{(G_\gamma L)(x;t)}\ ds \tag{16}$$

where $ds^2 = dx^2 + dy^2$.

4 Implementation

For our implementation we will be using the following image to run our tests on

Figure 1: Lena Image

Figure 1 is a standard test image for image processing and is good for our purposes since it contains a variety of different edges with different strengths.

So what we first do is define our convolution functions that we will be using in code.

```
function combine_kernels(kers...)
    return reduce(conv2, kers)
end

function convolve_image(I, kers...)
    kernel = combine_kernels(kers...)
    return imfilter(I, centered(kernel))
end

function convolve_scale_space(L, kers...)
    return mapslices(
        scale_slice -> convolve_image(scale_slice, kers...),
        L,
        (1,2)
    )
end

function convolve_gaussian(img, sigma)
    # The dimension of the convolution matrix
    length = 8*ceil(Int, sigma) + 1
    return imfilter(img, reflect(Kernel.gaussian((sigma, sigma),
         (length, length))))
end
```

These will be used for computing all the neccessary two dimensional matrix convolutions that we will use throughout the rest of the code. The next step is to then define our range of scales used and which derivative kernels we will be using to compute two dimensional discrete spatial derivatives. The following code shows the definitions for our variables that will be used for all further calculations.

```
# Parameters
gamma = 1
scales = exp.(linspace(0, log(50), 40))

# Load the image
img = float.(ColorTypes.Gray.(testimage("lena_color_512")))

# Define derivative convolution matrices
Dy = Array(parent(Kernel.ando5()[1]))
Dx = Array(parent(Kernel.ando5()[2]))
```

References