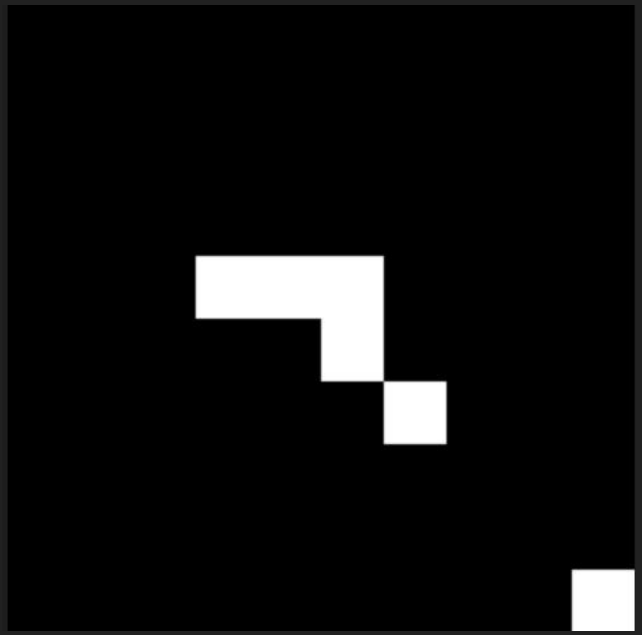
Morphological Filters

Tadeo Hepperle
Advanced Bioimage Programming (NTUST, 2022)
Prof. Dr. Ching-Wei Wang 王靖維 教授

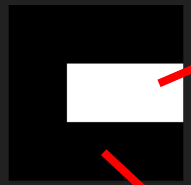
source code:
https://github.com/tadeohepperle/advanced_bioimage_programming/tree/master/image_ops_julia

Binary Dilution

Image



Kernel



active pixels

non-active

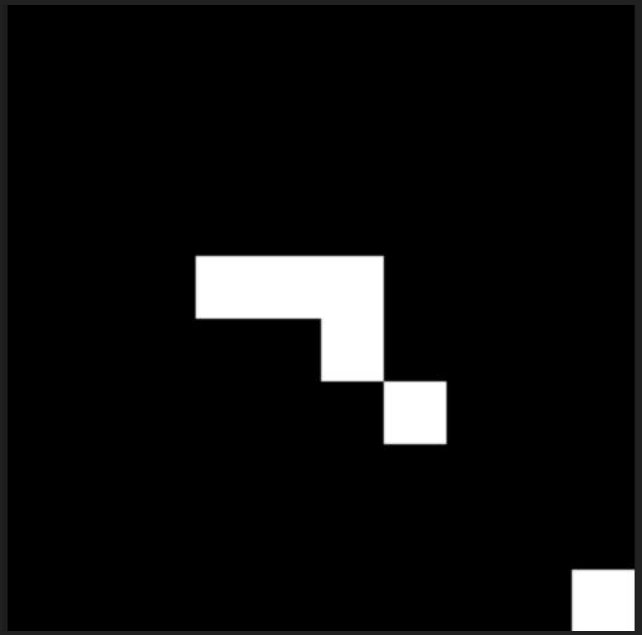
Dilution = All possible position combinations of active pixels in image and kernel

Binary Dilution

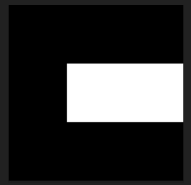
```
# Dilution Function
function dilution_binary(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    y_kernel_dim, x_kernel_dim = size(kernel) # eg. 3,3
    focus_y, focus_x = ceil(Int, y_kernel_dim / 2), ceil(Int, x_kernel_dim / 2) # eg. 2,2
    height, width = size(image) # eg. 10,10
    result_image = Gray.(zeros(height + y_kernel_dim - 1, width + x_kernel_dim - 1)) #eg. 12,12
    for j in 1:y_kernel_dim
        for i in 1:x_kernel_dim
            if kernel[j, i] == 1.0
                # the kernel pixel is active
                offset_y, offset_x = j - focus_y, i - focus_x
                for y in 1:height
                    for x in 1:width
                        if image[y, x] == 1.0
                            result_image[y+offset_y+focus_y-1, x+offset_x+focus_x-1] = 1.0
                        end
                    end
                end
            end
        end
    end
    return result_image[2:end-1, 2:end-1]
end
```

Binary Dilution

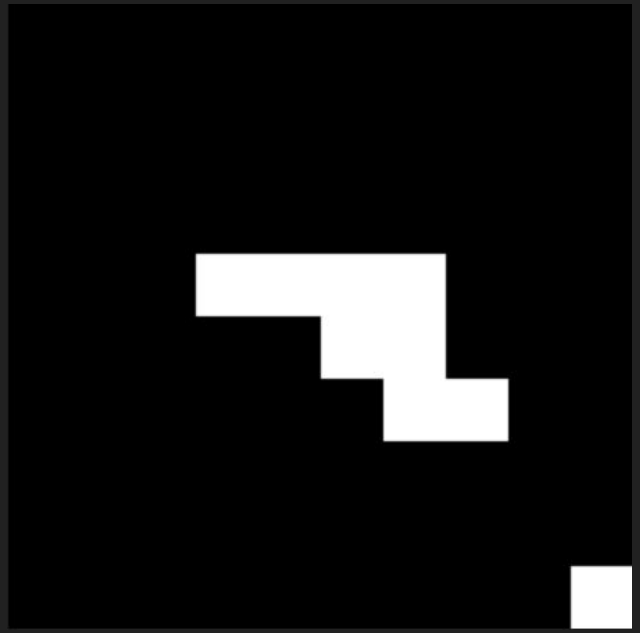
Image



Kernel



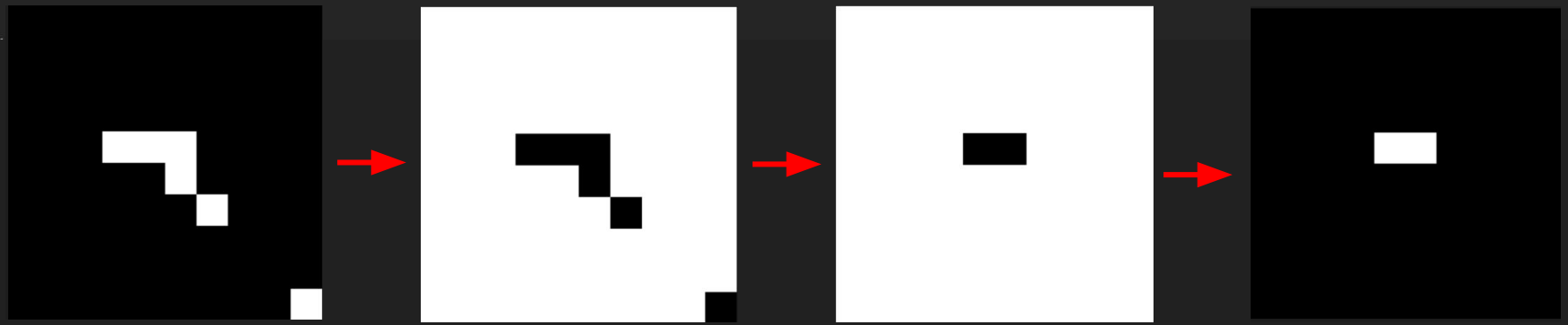
Diluted Image



Binary Erosion

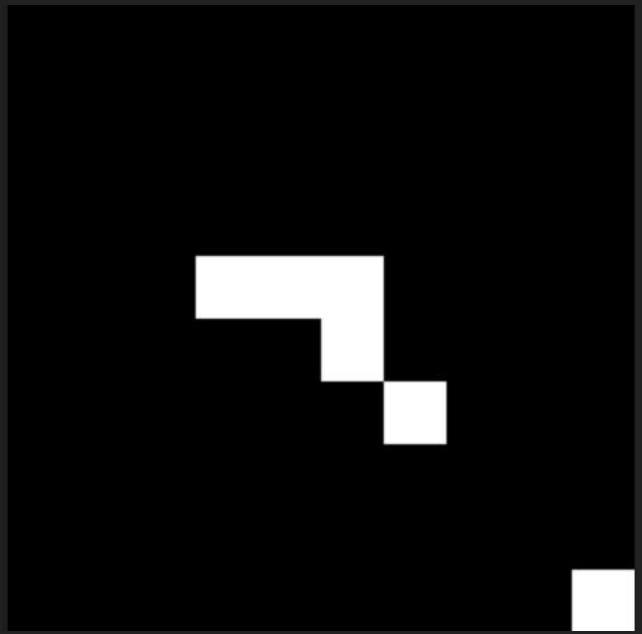
Erosion = Invert(Dilution(Invert(img)))

```
invert(image::Matrix{Gray{Float64}}) = image .* -1 .+ 1

function erosion_binary(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    invert(dilution_binary(invert(image), kernel))
end

resize(erosion_binary(img1, kernel1))
```

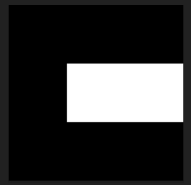


Binary Erosion

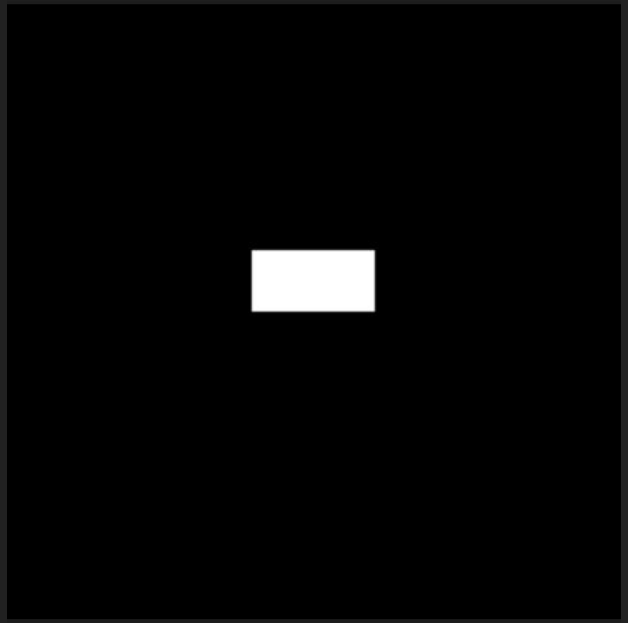
Image



Kernel



Eroded Image



Continuous Dilution

```
function dilution_continous(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    y_kernel_dim, x_kernel_dim = size(kernel) # eg. 3,3
    focus_y, focus_x = ceil(Int, y_kernel_dim / 2), ceil(Int, x_kernel_dim / 2) # eg. 2,2
    height, width = size(image) # eg. 10,10
    result_image = Gray.(zeros(height + y_kernel_dim - 1, width + x_kernel_dim - 1)) #eg. 12,12
    for j in 1:y_kernel_dim
        for i in 1:x_kernel_dim
            if kernel[j, i] != 99.0
                # the kernel pixel is active
                offset_y, offset_x = j - focus_y, i - focus_x
                for y in 1:height
                    for x in 1:width
                        prev_cell = result_image[y+offset_y+focus_y-1, x+offset_x+focus_x-1]
                        maybe_next_cell = kernel[j, i] * image[y, x]
                        if maybe_next_cell > prev_cell
                            result_image[y+offset_y+focus_y-1, x+offset_x+focus_x-1] = maybe_next_cell
                        end
                    end
                end
            end

        end
    end
    return result_image[2:end-1, 2:end-1]
end
```

99 used as missing value / “ignore” encoding

Continuous Dilution

= max of possible combinations of multiplying image and kernel

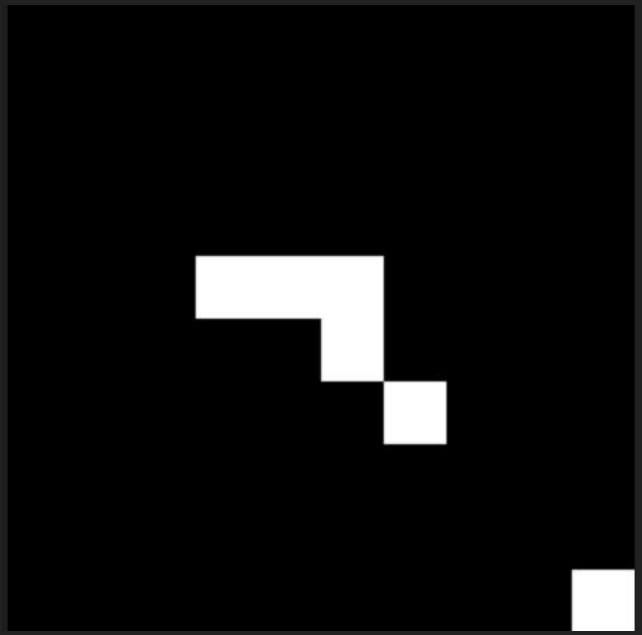
= Convolution but use only max instead of adding it up

```
prev_cell = result_image[y+offset_y+focus_y-1, x+offset_x+focus_x-1]
maybe_next_cell = kernel[j, i] * image[y, x]
if maybe_next_cell > prev_cell
    result_image[y+offset_y+focus_y-1, x+offset_x+focus_x-1] = maybe_next_cell
end
```

Continuous Dilution

```
kernel_c = Gray.([
    0.1 0.2 0.1
    0.8 1.0 0.8
    99.0 99.0 99.0])
```

Image



Kernel

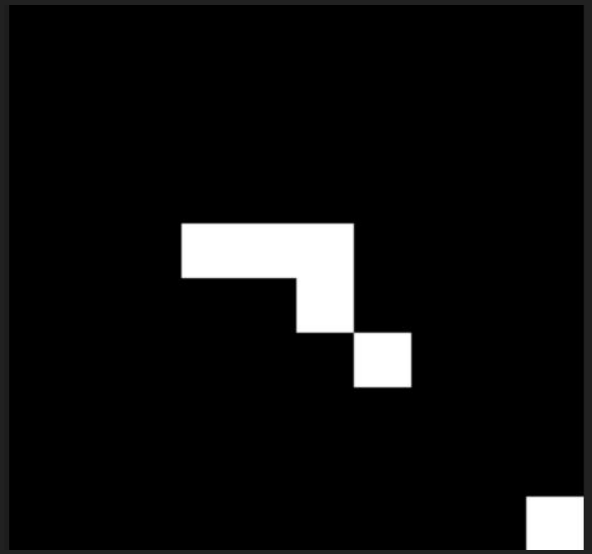
Diluted Image

Continuous Erosion

Erosion = Invert(Dilution(Invert(img)))

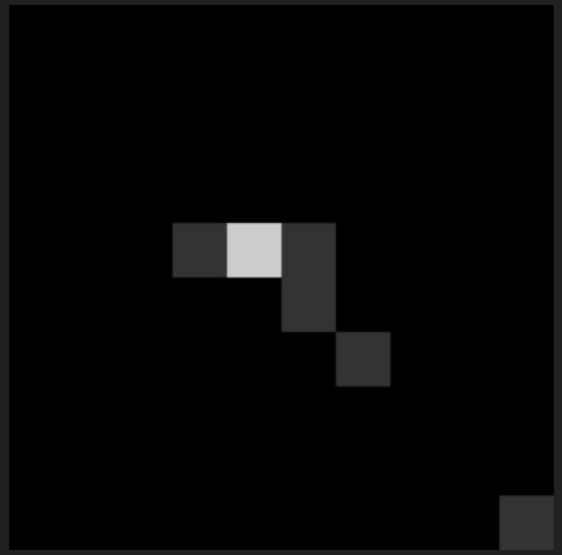
```
function erosion_continous(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    invert(dilution_continous(invert(image), kernel))
end
```

Image

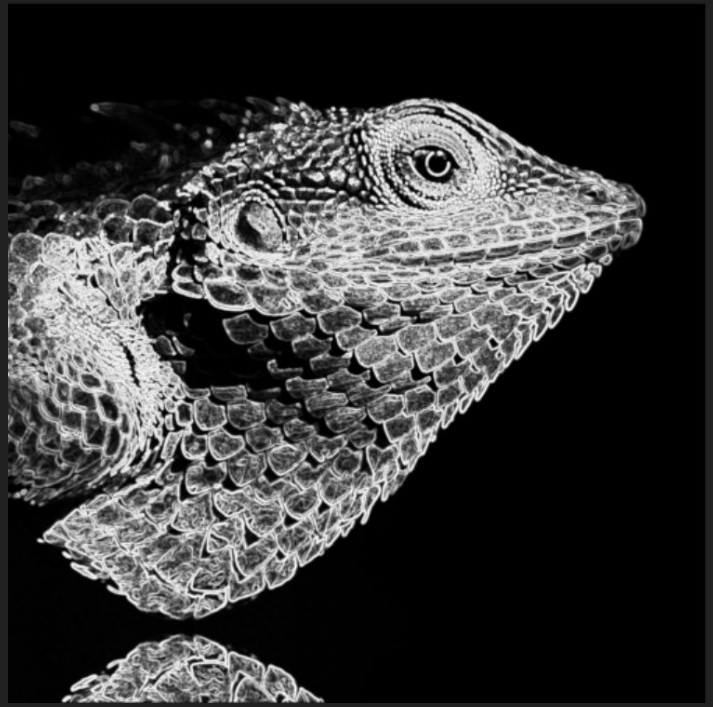


Kernel

Eroded Image



Let’s look at some bigger examples:

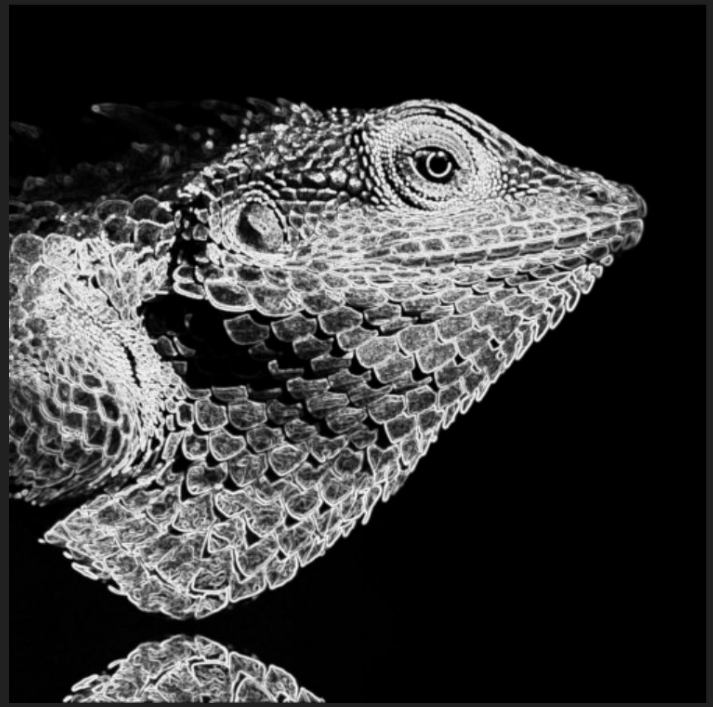


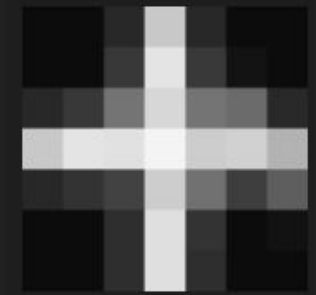
Lizard (4000x4000)

Robot (700x500)

Dilution Continuous

Original



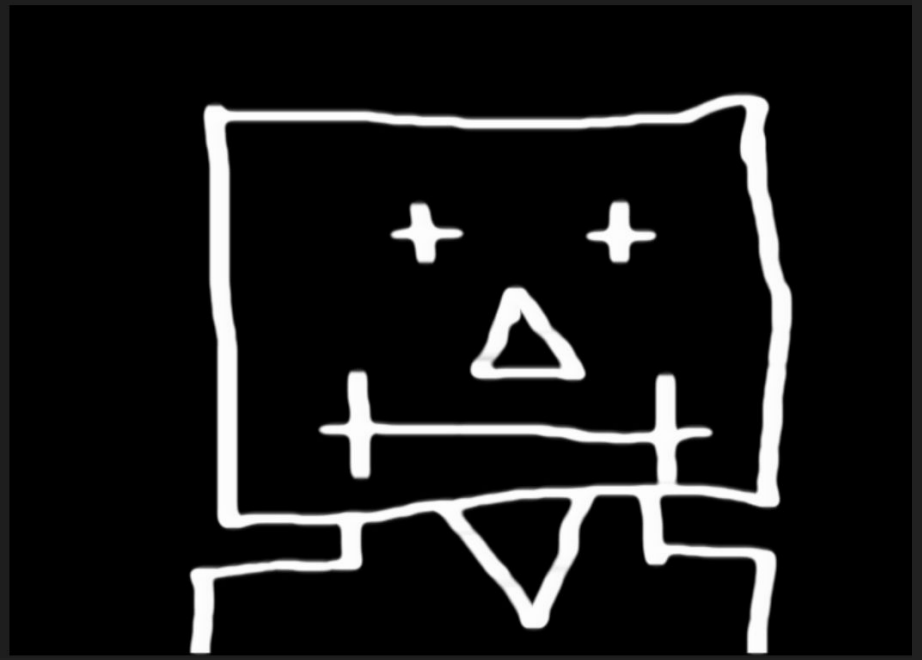


```
dilution_continous(img_lizard, kernels[4])
```
✓ 13.3s

Dilution Continuous

```
dilution_continous(dilution_continous(img_robot, kernels[5]), kernels[5])
```

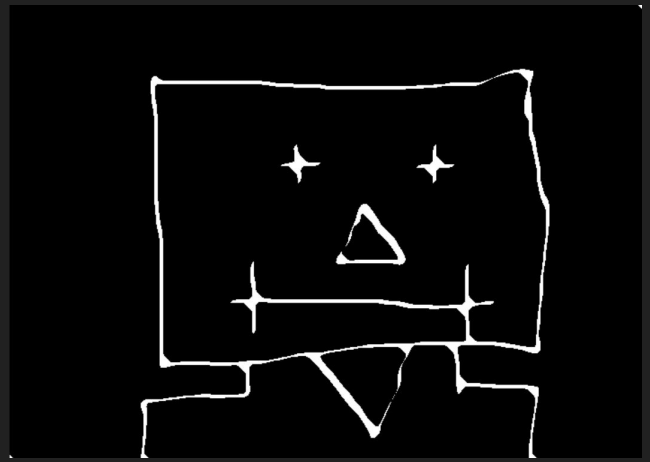
✓ 0.8s



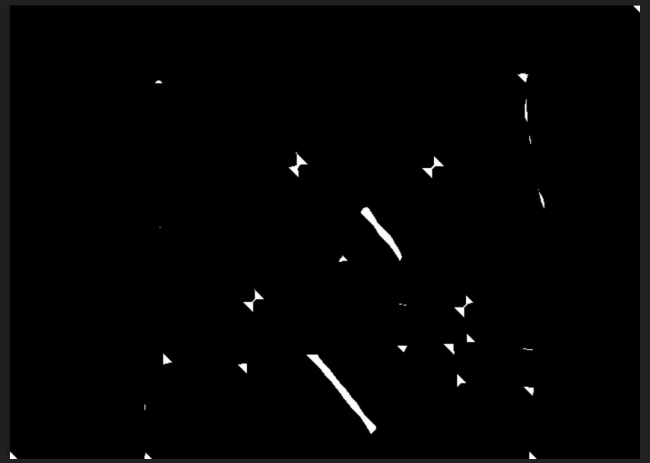
Erosion Binary

Original

1x



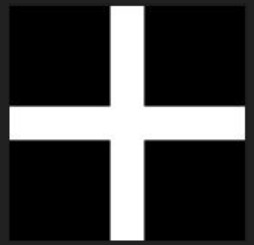
2x

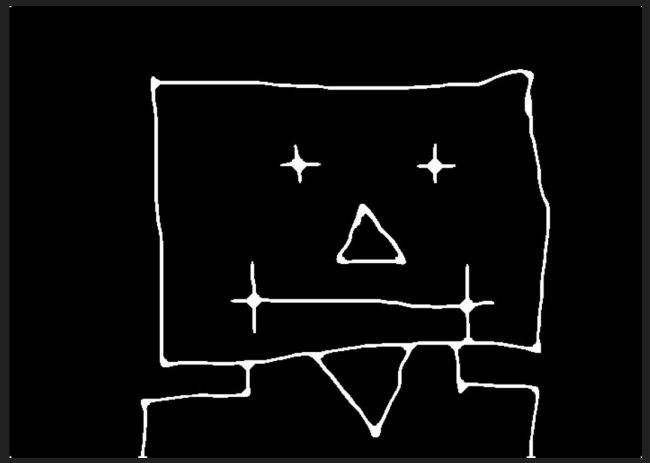


Erosion Binary

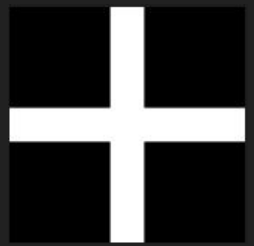
Original

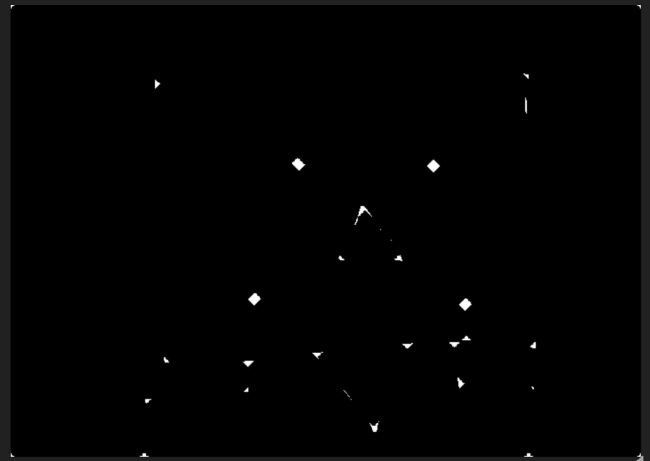
1x





2x





Runtime Analysis: Dilution Binary / Erosion Binary

```
kernel2 = Gray.(ones((3,3)))
```

`2.115488 seconds (6 allocations: 366.455 MiB)` binary 3x3 kernel with 9 active points

```
kernel1 = Gray.([0.0 0.0 0.0; 0.0 1.0 1.0; 0.0 0.0 0.0])
```

`0.593144 seconds (6 allocations: 366.455 MiB)` binary 3x3 kernel with 2 active points

Continuous Dilation: Effect of kernel size

```
map([(3,3), (5,5), (10,2), (7,7)]) do (x,y)
    kernel = Gray.(rand(x,y))
    println("random $(x)x$(y) kernel")
    @time i = dilution_continous(img_lizard, kernel)
    IJulia.display(i)
end
```

Runtime on 4000x4000 image

```
  3.865078 seconds (6 allocations: 366.455 MiB, 1.30% gc time)

random 3x3 kernel

  8.136037 seconds (6 allocations: 366.822 MiB, 0.20% gc time)

random 5x5 kernel

  5.275202 seconds (6 allocations: 367.005 MiB, 0.58% gc time)

random 10x2 kernel

 12.386467 seconds (6 allocations: 367.188 MiB, 0.11% gc time)

random 7x7 kernel
```

Opening and Closing

```julia
function opening_binary(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    dilation_binary(erosion_binary(image, kernel),kernel)
end

function closing_binary(image::Matrix{Gray{Float64}}, kernel::Matrix{Gray{Float64}})
    erosion_binary(dilation_binary(image, kernel),kernel)
end
```

Input Continuous

Input Binary

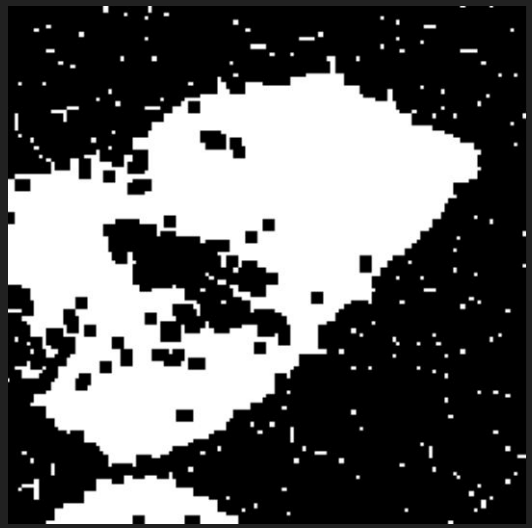
Opening and Closing Binary

```
kernel = Gray.(ones((3,3)))
```

Opening

Original

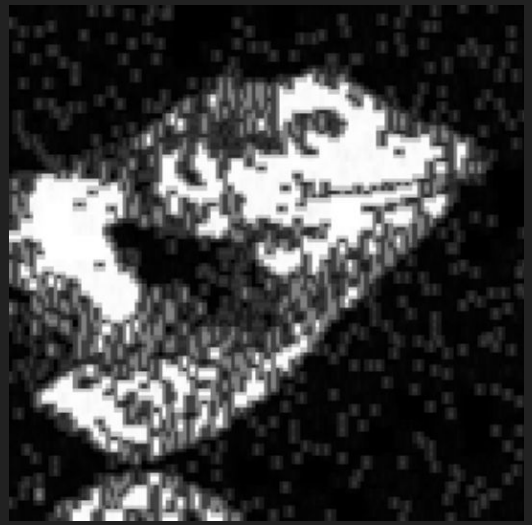
Closing



Opening and Closing Continuous

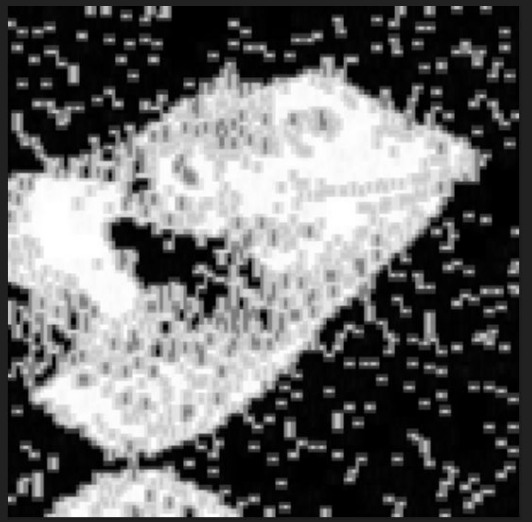
```
kernel = Gray.([0.5 0.8 0.5; 0.5 1.0 0.5; 0.5 0.8 0.5])
```

Opening



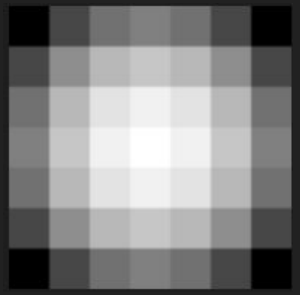
Original

Closing

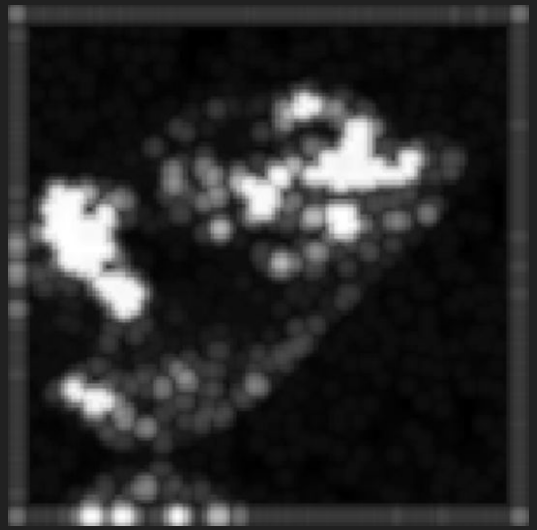


Opening and Closing Continuous

```
kernel = map(enumerate(ones(7,7))) do x
    (i,_) = x
    x = (i-1)÷7 -3
    y = (i-1)%7 -3
    Gray((18 - x^2 - y^2) / 18)
end
```



Opening



Original

Closing