Auto Thresholding

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

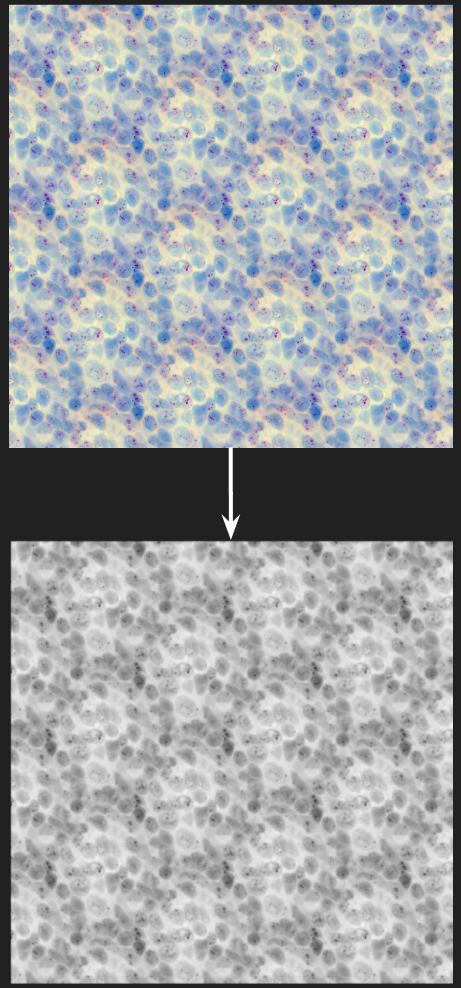
source code:
https://github.com/tadeohepperle/advanced_bioimage_programming/tree/master/image_ops_julia

Using a 4000x4000 example image

cells microscope image made into seamless texture

```
using Images
using IJulia
using Plots
using Interpolations
function resize(img)
    imresize(img, (300, 300), method=Interpolations.Constant())
end

Image = Matrix{Gray{Float32}}

img = load("./images/input/cells4000.jpg")
img = Gray{Float32}.(img)
```

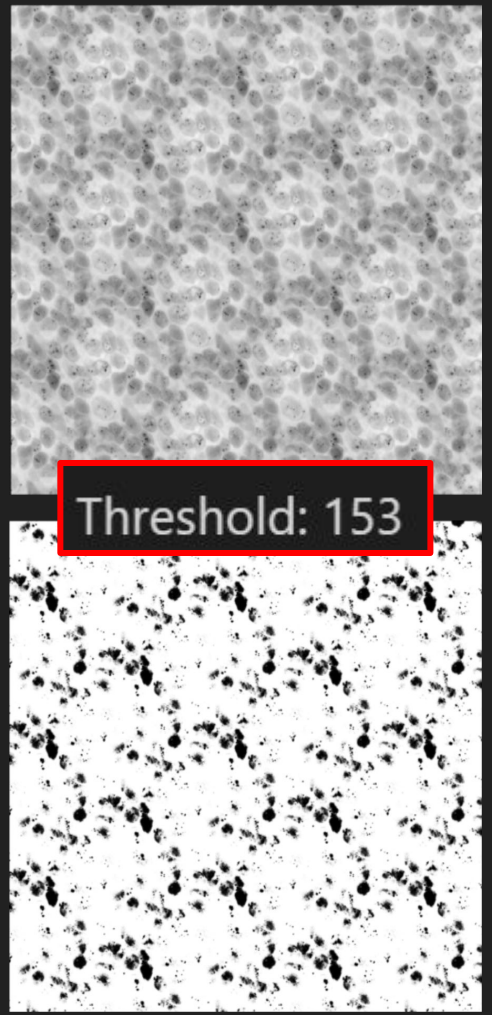
✓ 1.5s



Simple Thresholding Function

```
function thresholding(img::Image, thresh)::Image
    (h,w) = size(img)
    output = Gray.(zeros(Float32,h,w))
    for i in 1:h
        for j in 1:w
            output[i,j] = img[i,j] > thresh ? 1.0 : 0
        end
    end
    output
end

thresholding(img, 0.6)
```

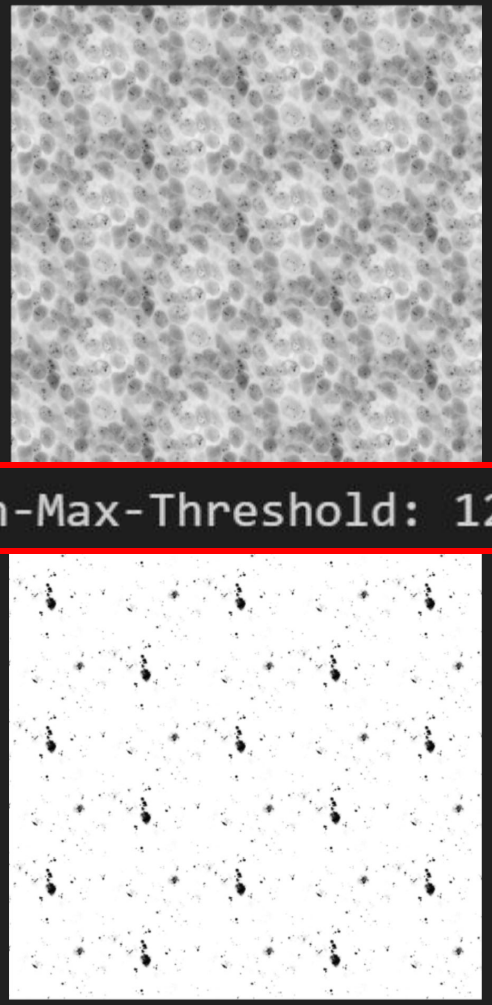
✓ 0.3s



Min-Max-Thresholding

```
function min_max_thresholding(img::Image)::Image
    min = Inf
    max = -Inf
    (h,w) = size(img)
    for i in 1:h
        for j in 1:w
            pix = img[i,j]
            if pix > max
                max = pix
            end
            if pix < min
                min = pix
            end
        end
    end
    thresholding(img, (max-min) / 2)
end
min_max_thresholding(img)
```

✓ 0.3s



How to make a histogram

```
function hist(img::Image)::Array{Float32}
    buckets = zeros(Int, 256)
    (h,w) = size(img)
    for i in 1:h
        for j in 1:w
            index = Int(ceil(img[i,j] * 256 + eps(Float32) ))
            buckets[index]+=1
        end
    end
    return map(buckets) do x
        x/(h*w)
    end
end
```

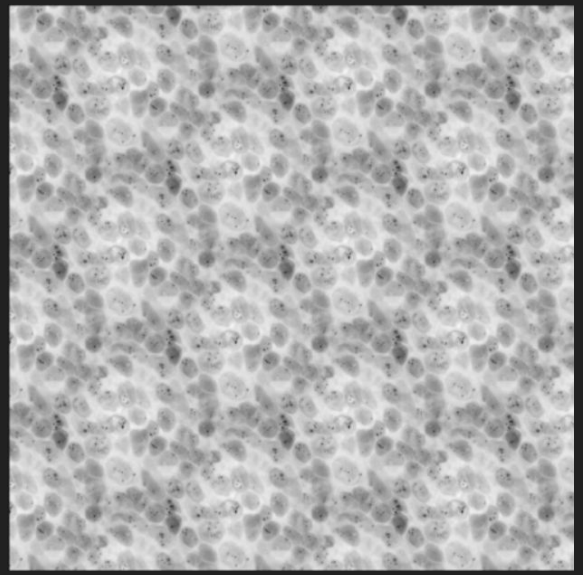
✓ 0.1s

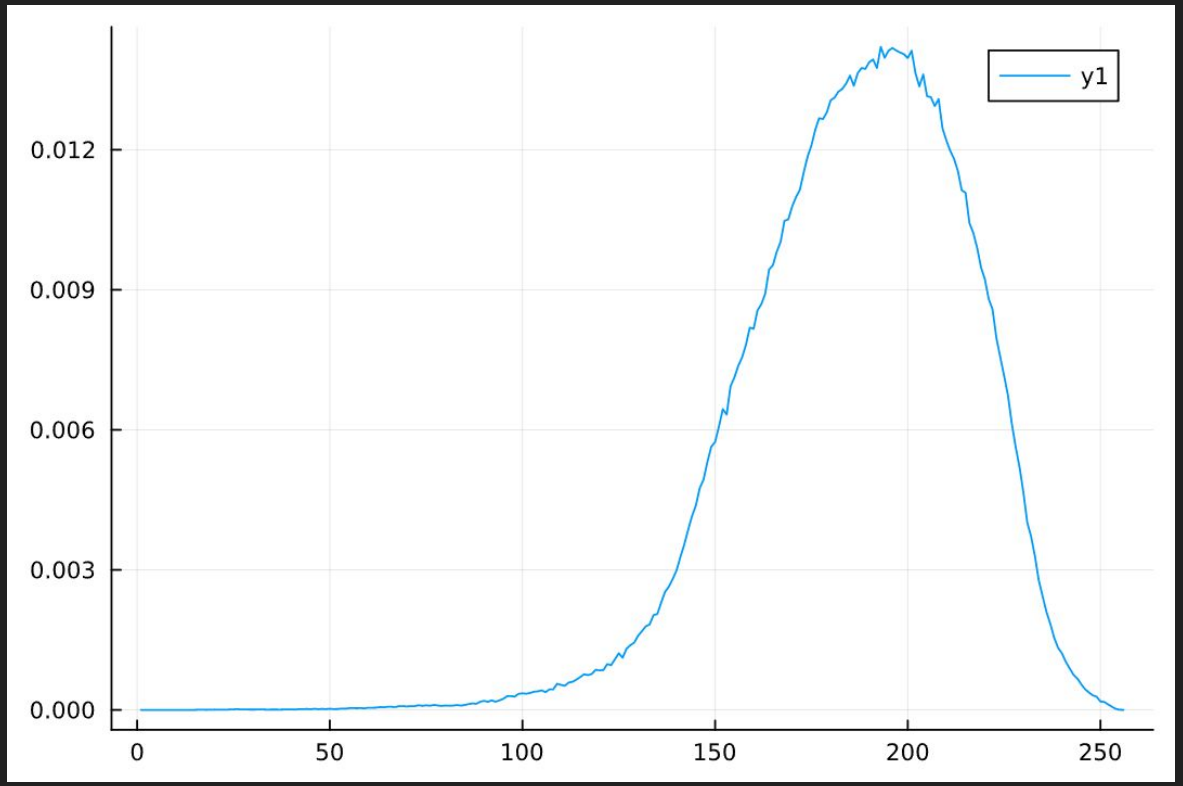
on 4000x4000

```
256-element Vector{Float32}:
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 ⋮
 0.0003128125
 0.0002841875
 0.0001790625
 0.000170625
 0.00011625
 6.61875f-5
 2.11875f-5
 2.8125f-6
 0.0
```

How to make a histogram

Input (4000x4000):



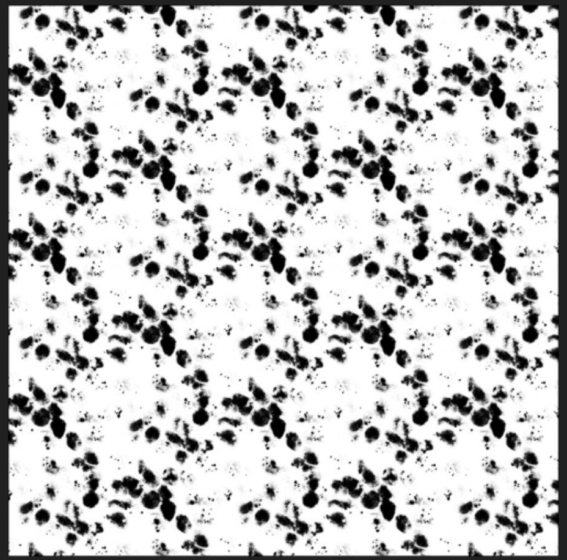


Quantile Thresholding

```
function quantile_thresholding(img::Image, quantile=0.5)::Image
    h = hist(img)
    acc = 0.0
    i = 0
    for _ in 1:length(h)
        i+=1
        acc += h[i]
        if acc >= quantile
            break
        end
    end
    thresholding(img, Float32(i) / 256.0)
end
```

Quantile Thresholding

```
quantile_thresholding(img, 0.2)
```

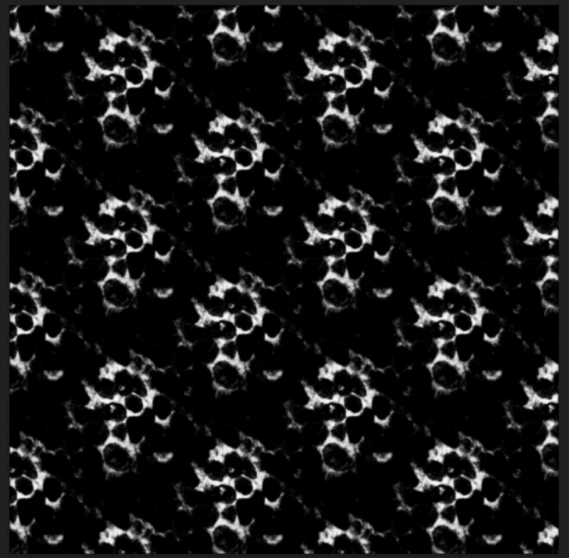


✓ 0.3s

```
quantile_thresholding(img, 0.5)
```

✓ 0.3s

```
quantile_thresholding(img, 0.9)
```



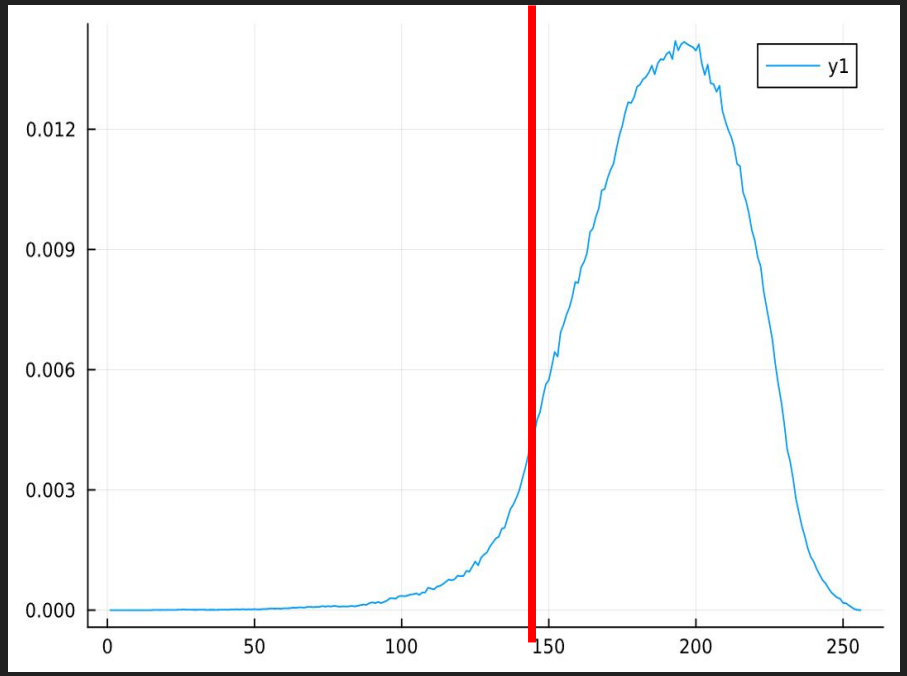
✓ 0.3s

Otsu's Method: Computing all between-class variances

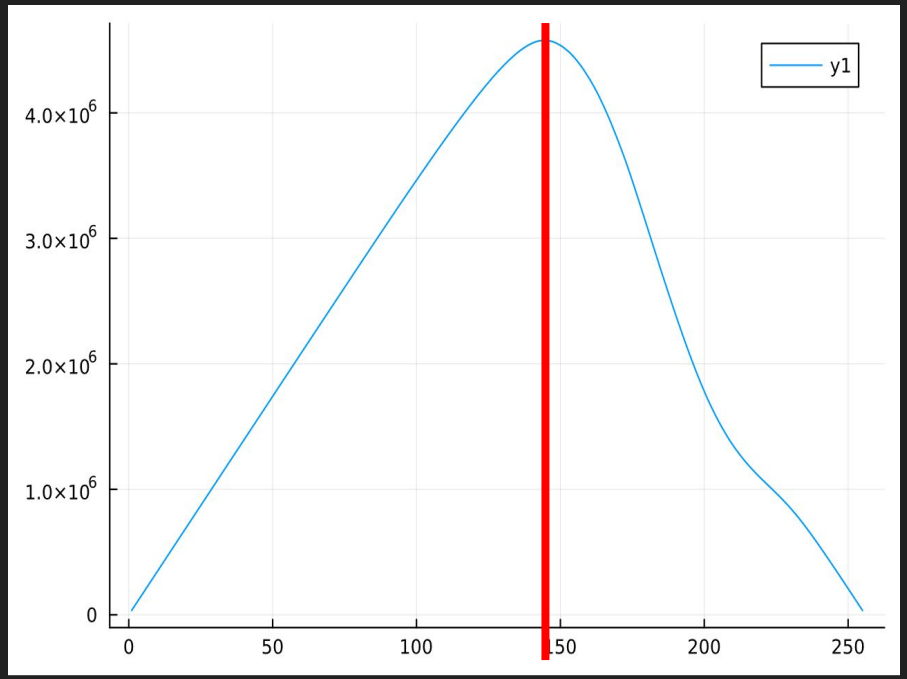
```
function histmean(h::Array{Float32}; start_val=1)::Float32
    sum(collect(start_val:length(h)+start_val-1) .* h)
end

function otsu_between_variances(hist::Array{Float32})::Array{Float32}
    mean =  histmean(hist)
    between_variances = zeros(length(hist)-1)
    for i in 1:(length(hist)-1)
        # all pixels <= i belong to one category, > i to the other
        # calculate between class variance:
        sec1 = range(1,i)
        sec2 = range(i+1,length(hist))
        sec1_mean = histmean(hist[sec1])
        sec2_mean = histmean(hist[sec2], start_val=i+1)
        between_variances[i] = (sec1_mean-mean)^2 * length(sec1) + (sec2_mean -mean)^2 * length(sec2)
    end
    between_variances
end
```

Otsu’s Method: Computing all between-class variances

Histogram:



Between-class variance:



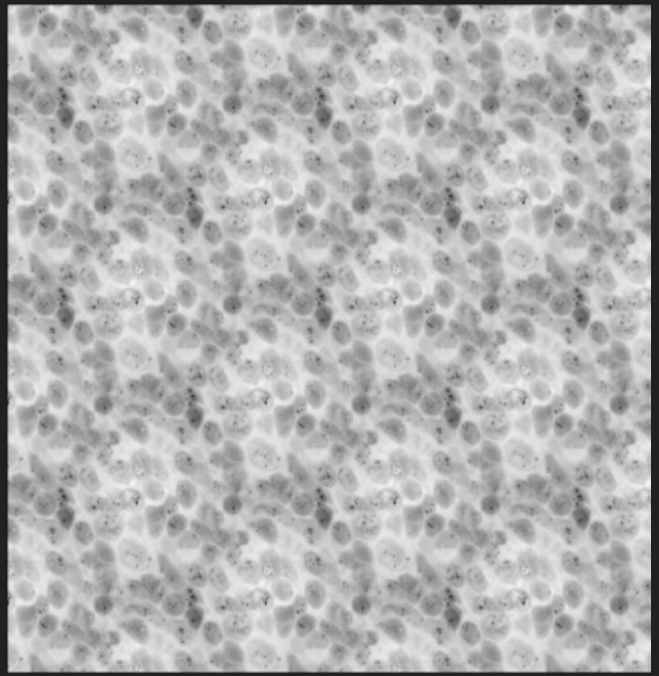
Otsu’s Method: Picking Threshold with highest between-class variance

```
function otsu_thresholding(img::Image)::Image
    _, index = findmax(otsu_between_variances(hist(img)))
    print("Otsu using value $(index)/255 as threshold")
    thresholding(img, Float32(index) / 256.0)
end
otsu_thresholding(img)
```

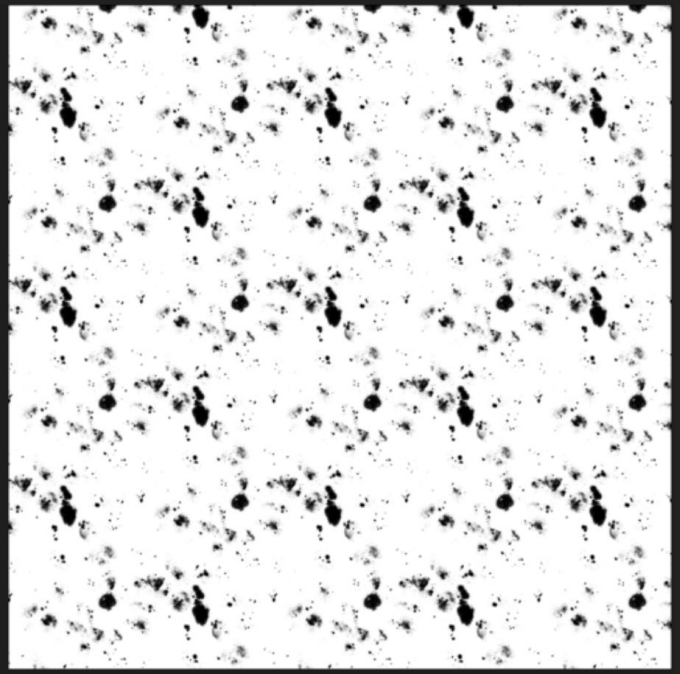
✓ 0.3s

```
Otsu using value 145/255 as threshold
```

Otsu’s Method: Picking Threshold with highest between-class variance

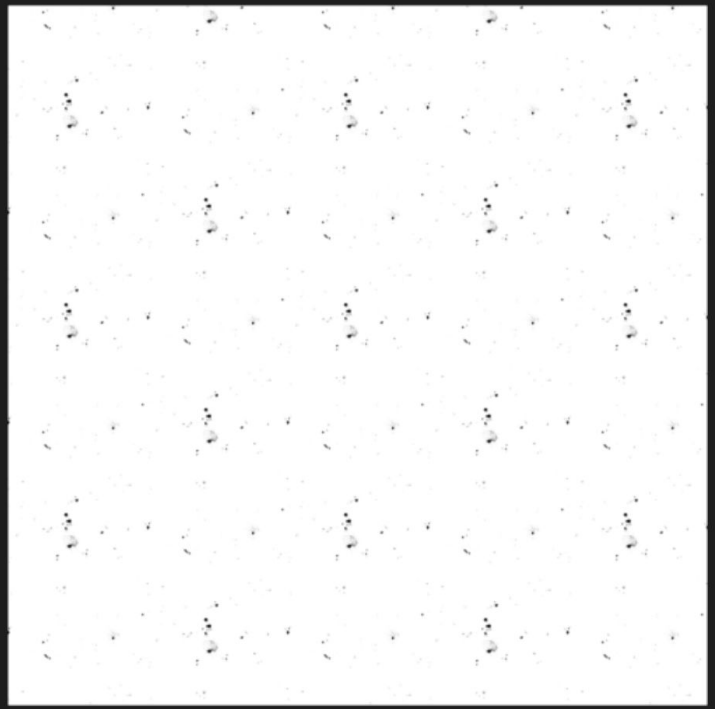


```
Otsu using value 145/255 as threshold
```



Isodata Thresholding

```
q = 186
q = 93
Isodata threshold: 93
```



✓ 0.3s

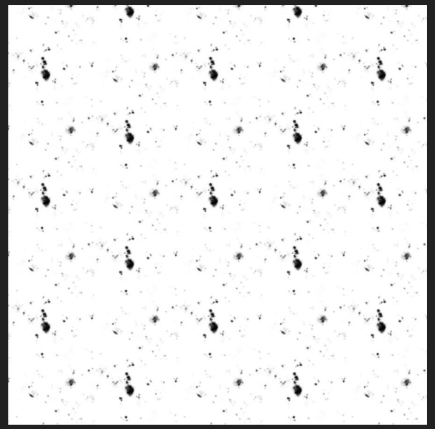
```
function isodata_threshold(img::Image)::Image
    h = hist(img)
    k = length(h)
    q = floor(Int, histmean(h))
    while true
        @show q
        sec1 = h[1:q]
        sec2 = h[q+1:end]
        if sum(sec1)==0 || sum(sec2) ==0
            throw("No threshold found")
        end
        mu0 = histmean(sec1)
        mu1 = histmean(sec2, start_val=q+1)
        q_old = q
        q = floor(Int, (mu0 + mu1) / 2)
        if q_old == q
            break
        end
    end
    println("Isodata threshold: $(q)")
    thresholding(img, Float32(q) / 256.0)
end
isodata_threshold(img)
```

✓ 0.3s

Comparison

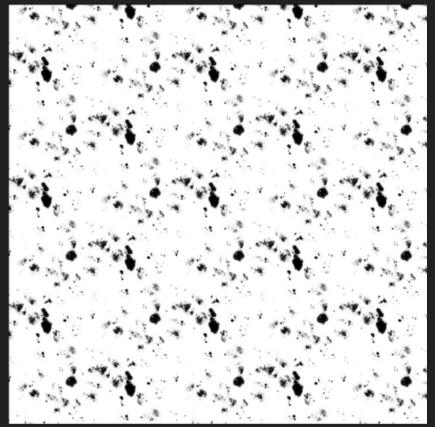
All finish within
0.3 seconds

Min-Max
(121)



Quantile 0.5
(v: 189)

Otsu’s Method
(145)



Isodata
(v: 93)