E05

R語言
影像資料分析

吳漢銘
國立政治大學 統計學系

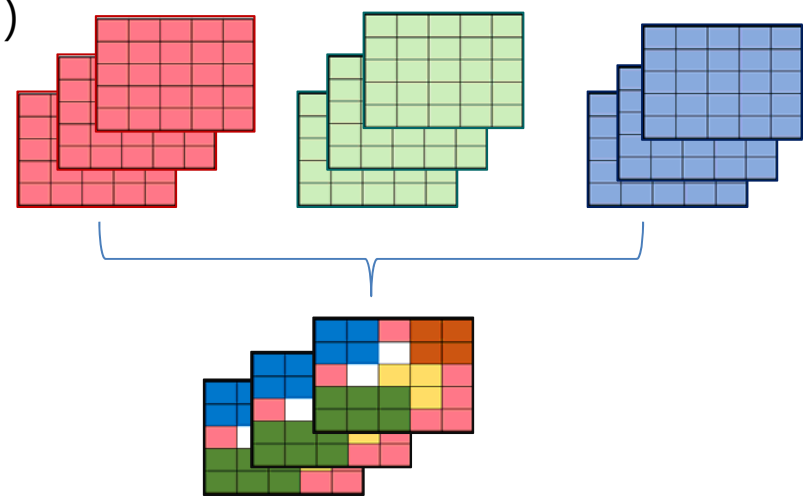
https://hmwu.idv.tw

R Packages for Image Processing

- CRAN Task View: Medical Image Analysis
  https://cran.r-project.org/web/views/MedicalImaging.html
- **imager**: Image Processing Library Based on 'CImg'
  https://cran.r-project.org/web/packages/imager/
- **magick**: Advanced Graphics and Image-Processing in R
  https://cran.r-project.org/web/packages/magick/
- **OpenImageR**: An Image Processing Toolkit
  https://cran.r-project.org/web/packages/OpenImageR/
- **EBImage**: Image processing and analysis toolbox for R
  https://bioconductor.org/packages/release/bioc/html/EBImage.html

Images: CImg's storage standard

- **Images** are represented as **4D numeric arrays** (R object, labelled x, y, z, c) (CImg's storage standard)
  - x, y: spatial dimensions,
  - z: depth or time,
  - c: colour. (RGB)



- **Grayscale images**: the objects will still be officially 4 dimensional, with two trailing flat dimensions.

- Pixels are stored in the following manner:
  - The image is scanned beginning at the upper-left corner, along the x axis. Once we hit the end of the scanline, we move to the next line. Once we hit the end of the screen, we move to the next frame (increasing z) and repeat the process.
  - If we have several colour channels, then once we' re done with the first colour channel we move to the next one.
  - All in all the different dimensions are represented in the x,y,z,c order.

Example images: `imager_extdata`

Leonardo_Birds

coins

boats

hubble

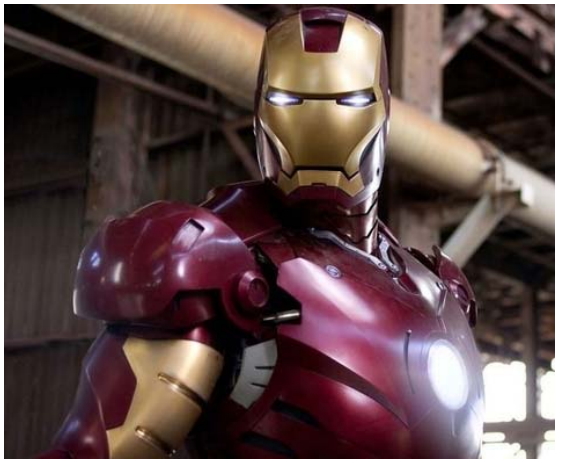
parrots

Example images: image_other

Car_movie.jpg

Cars-on-Busy-Street.jpg

plaza.jpg



ironman.jpg

Texture Images

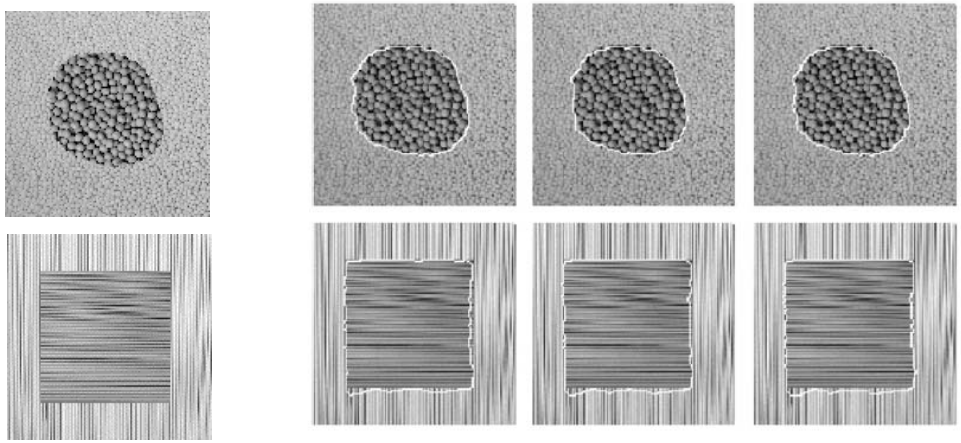


Fig. 6. The segmentation results by the SIR + NMC, SIR + LIBSVM and SIR + SVMTorch (from left to right) are displayed based on the clustering results of the MDS + K-Means for the texture images Figs. 4(a) and 4(b) with a block size of 8×8 in the space–frequency domain.

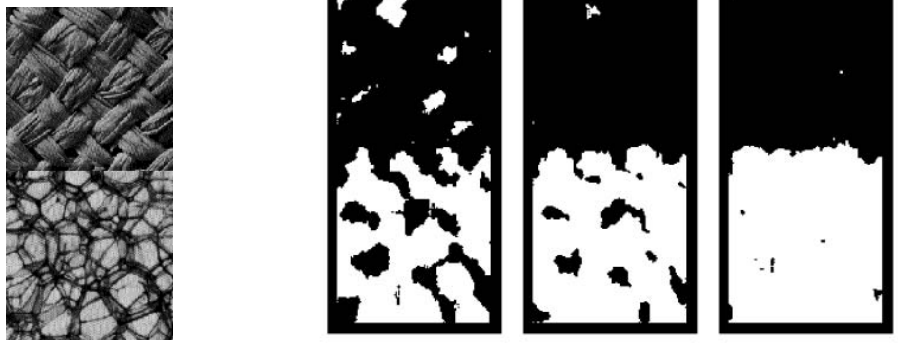
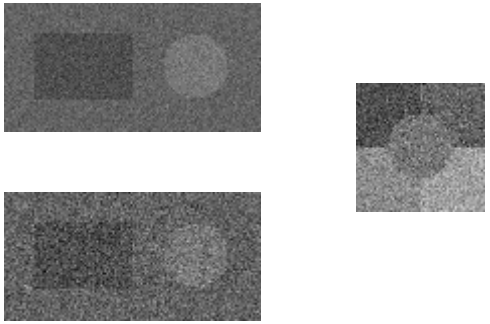


Fig. 7. The ISIR + NMC segmentation for the texture image Fig. 4(c) is demonstrated for the initial (left), five (middle) and 13 (right) iterations based on the MDS + K-Means in the frequency domain.



Segmentation of the cDNA microarray images

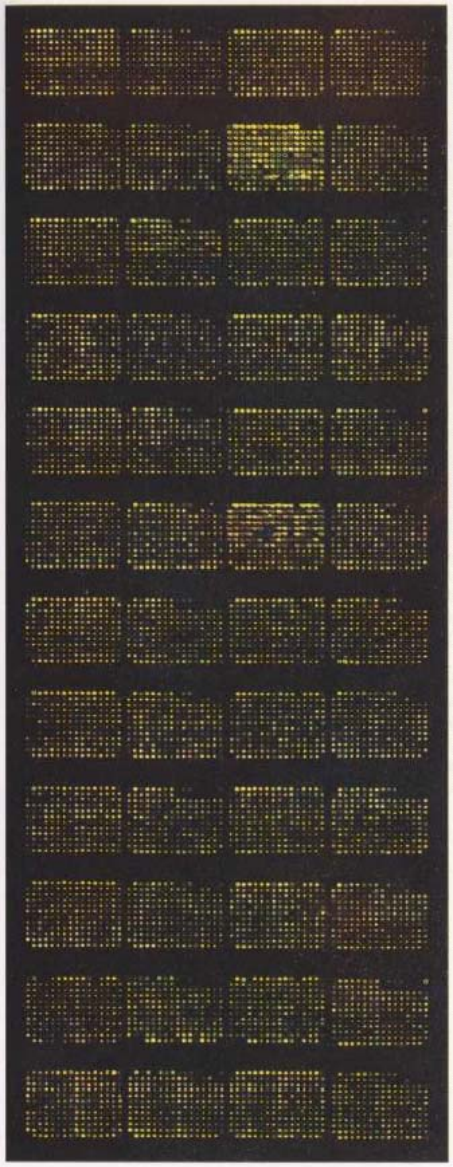
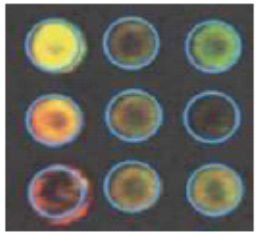


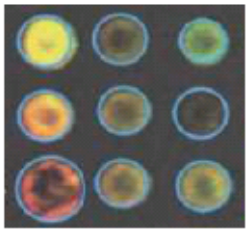
Image Processing for cDNA microarray.

1. Identifying the positions of the features: **Addressing**
2. Identifying the pixels that comprise the features (foreground (signal) or background.): **Segmentation**
3. Identifying the background pixels.
4. Calculation of numerical information: **Information extraction**

Fixed circle segmentation

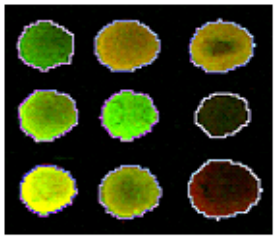


Variable circle segmentation



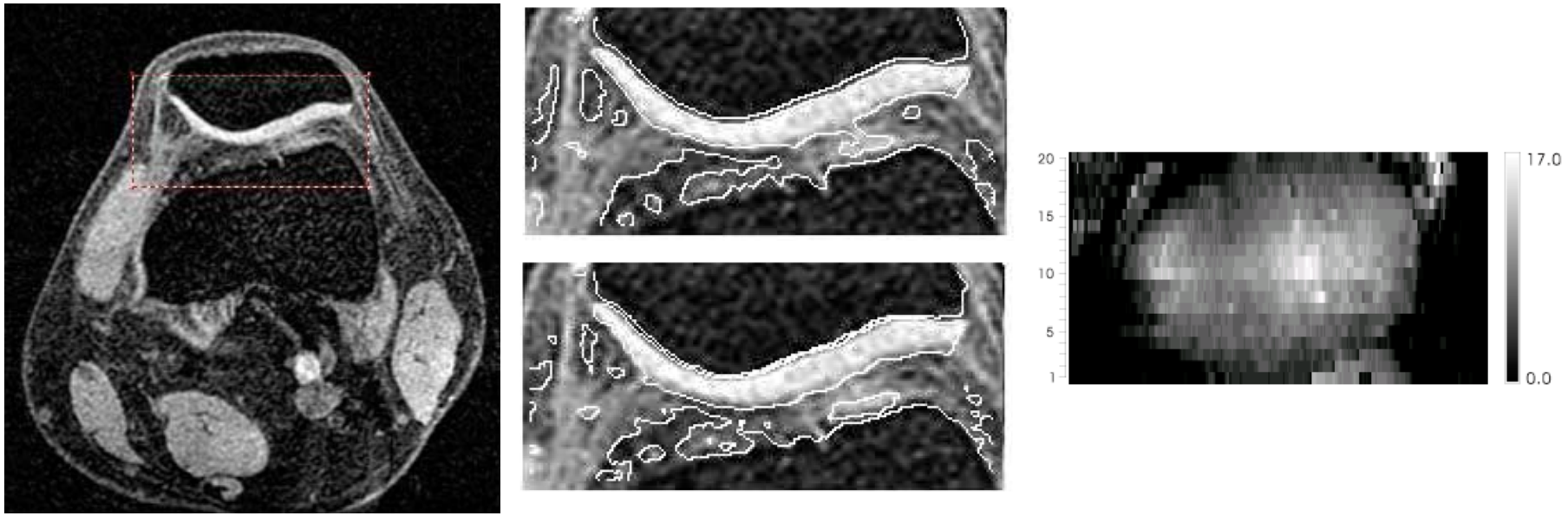
Histogram Segmentation

Adaptive shape segmentation



Medical Image Segmentation (fMRI)

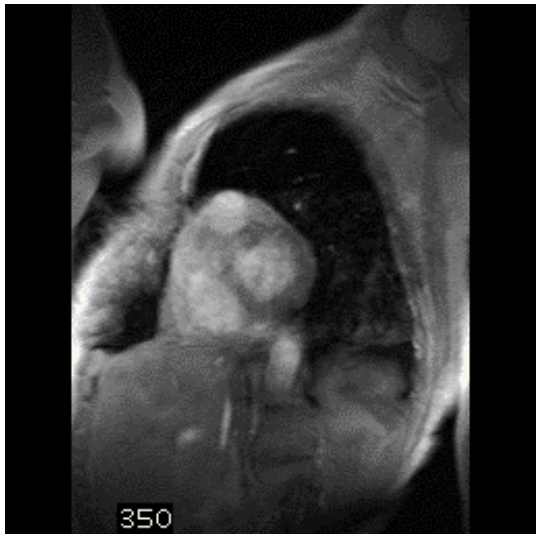
Medical images (fMRI) of a knee



The cartilaginous tissues (the brighter part) is the object-of-interest.

Wu, H. M., and Lu, H. H.-S.* (2007), Iterative Sliced Inverse Regression for Segmentation of Ultrasound and MR Images, Pattern Recognition 40(12), 3492-3502

Medical Image Segmentation: dynamic



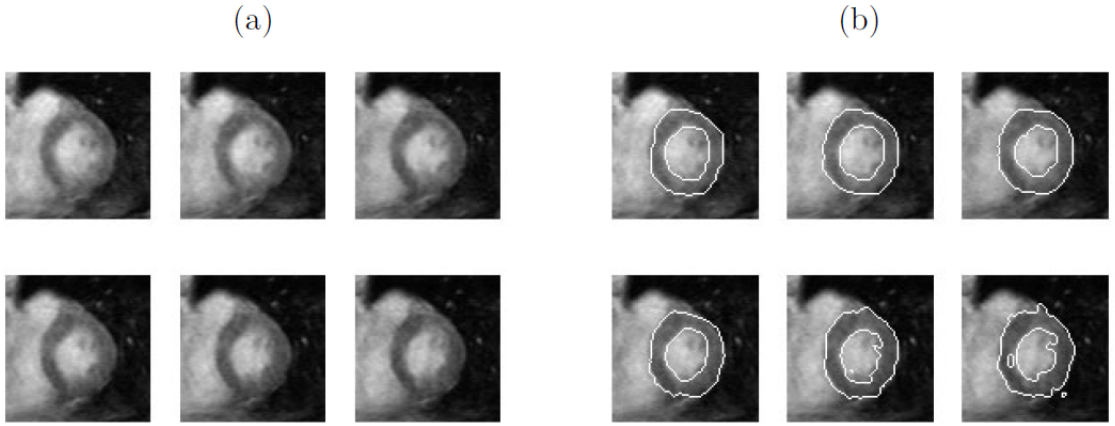
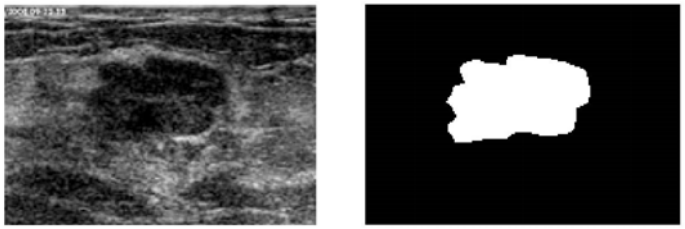


Figure 2. (a) The MR image sequence of myocardium of size 84×84 pixels in the ROI. The frame order is from left to right and from top to bottom. (b) The segmentation results are displayed for the MR image sequence of myocardium in space-frequency domain. The top three images display the classification results and the bottom three images show the prediction results.

Wu, H. M., and Lu, H. H.-S.* (2004), Supervised Motion Segmentation by Spatial-Frequential Analysis and Dynamic Sliced Inverse Regression, Statistica Sinica 14, 413-430.

Medical Image Segmentation (US)



Li, Y.Y. and Wu, H.M. (2019), Landmark isometric fuzzy sliced inverse regression with application to medical image segmentation, Journal of the Chinese Statistical Association, 57(1), 43-70.

真實胸部超音波影像及其真實切割 (來源: Xian et al., 2018)。

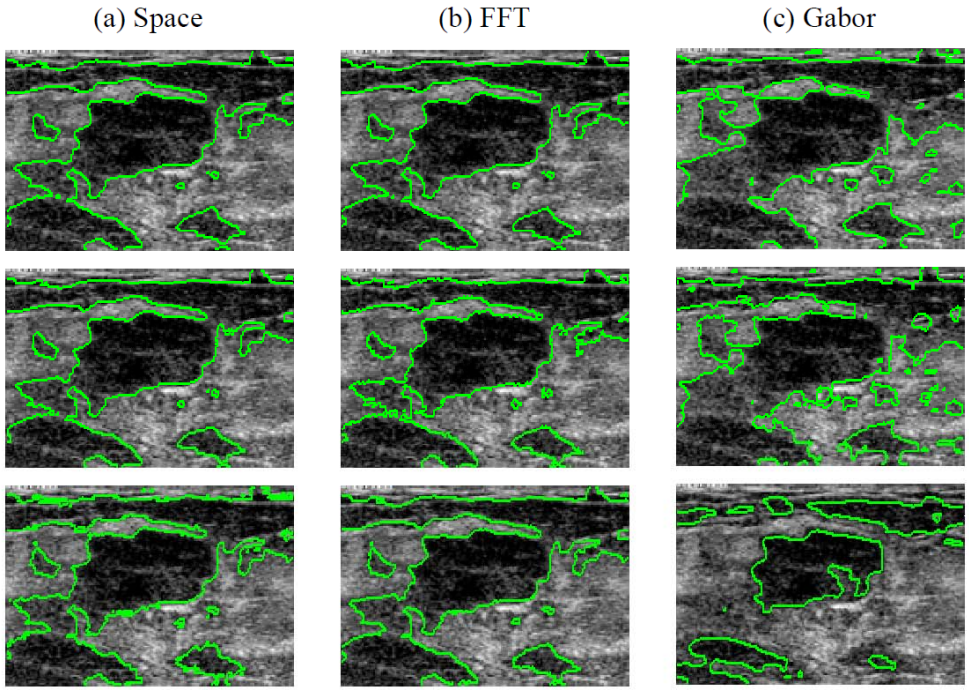


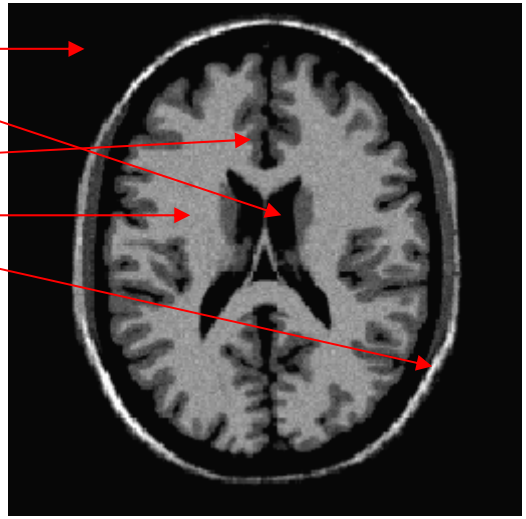
圖 9: 不同切割方法於真實胸部超音波影像在空間域、頻率域及空間-頻率域的切割結果。由上而下依序是 FCM、FSIR 及 L-ISOFSIR 方法。

Simulated Brain MRI images

BrainWeb: http://www.bic.mni.mcgill.ca/brainweb/

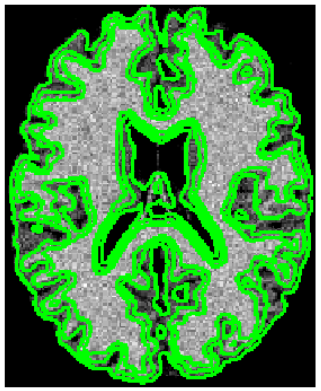
Modality: **T1**, Slice thickness: 1mm, Noise: 3%, Intensity non-uniformity: 0%

Background (背景)
CSF (腦脊液)
Grey Matter (灰質)
White Matter (白質)
Fat (脂肪)
Muscle (肌肉)
Muscle / Skin (肌肉/皮膚)
Skull (頭骨)
Vessels (血管)
Connective (軟組織)
Dura (硬腦膜)
Marrow (骨髓)

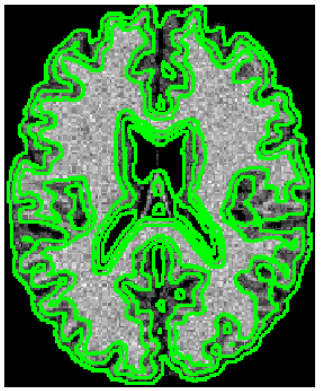


WM, GM, CSF

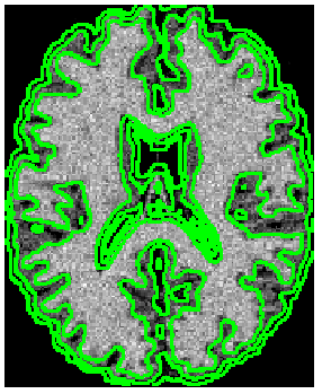
(a) Space



(b) FFT



(c) Gabor



Morphological Opening and Closing

外形影像處理

浸蝕運算 (Erosion)

《作用》浸蝕運算會將原影像去掉一層邊。只要影像中某一點 x 的周圍有一點是 0 則刪除 x 這一點，其運算如下：

$$G(x,y) = x \cap (x_0 \cap x_1 \cap x_2 \cap x_3 \cap x_4 \cap x_5 \cap x_6 \cap x_7)$$

G(x, y)：像素 (x, y) 的灰度值

點 x 和 x1 至 x7 的相對關係如下：

x3	x2	x1
x4	x	x0
x5	x6	x7

註：x = 1 代表物體；x = 0 代表背景。

增長運算 (Dilation)

《作用》增長運算會將原影像長一層邊。只要影像中某一點 x 的周圍有一點是 1 則輸出 1，其運算如下：

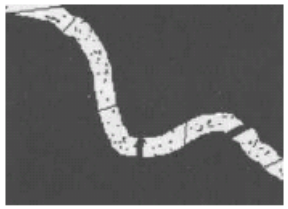
$$G(x,y) = x \cup (x_0 \cup x_1 \cup x_2 \cup x_3 \cup x_4 \cup x_5 \cup x_6 \cup x_7)$$

G(x, y)：像素 (x, y) 的灰度值

註：x = 1 代表物體；x = 0 代表背景。

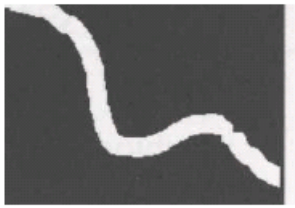
開閉運算 (Opening and Closing)

《作用》開運算爲連續 N 次浸蝕運算後再加上 N 次增長運算；
閉運算爲連續 N 次增長運算後再加上 N 次浸蝕運算。
開運算可將影像中藕斷絲連的物體切開；
閉運算則可補滿影像物體之間的細縫。

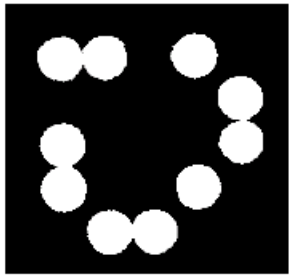


(a) original image

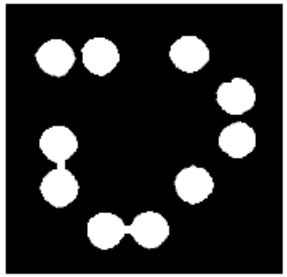
(b) 2 Dilation



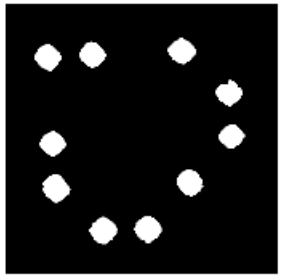
(c) 2 Erosion of (b)



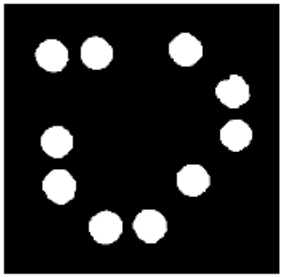
(a) original image



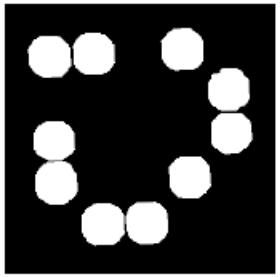
(b) 2 Erosion



(c) 5 Erosion



(d) 2 Dilation of (c)

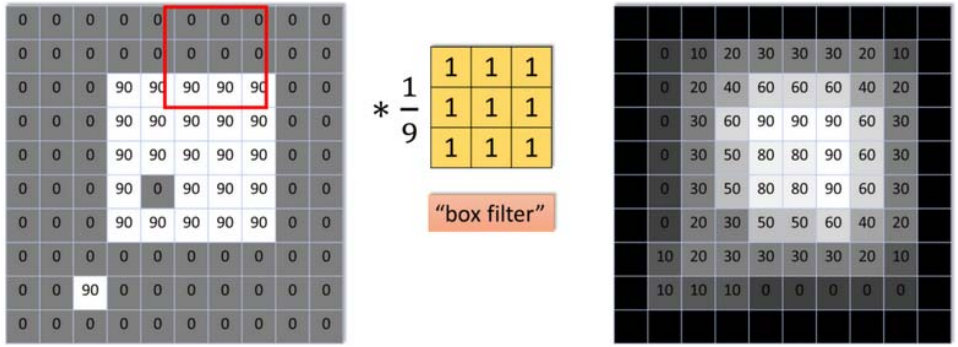


(e) 2 Dilation of (c)

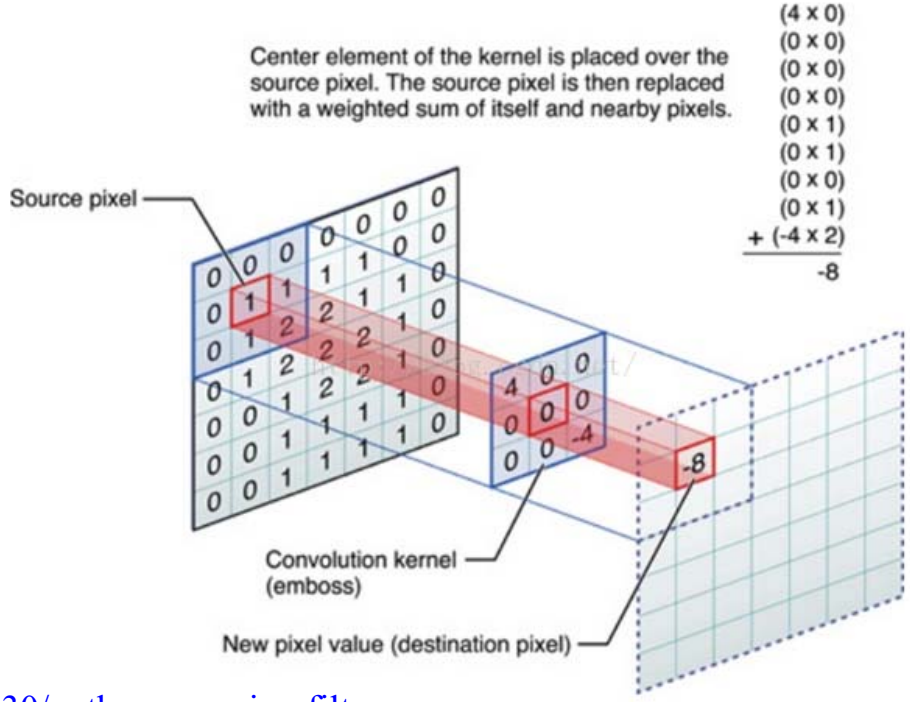
Filter, Convolution

Averaging filter

$$F(x,y) \quad * \quad H(u,v) \quad = \quad G(x,y)$$



$$G = F * H$$

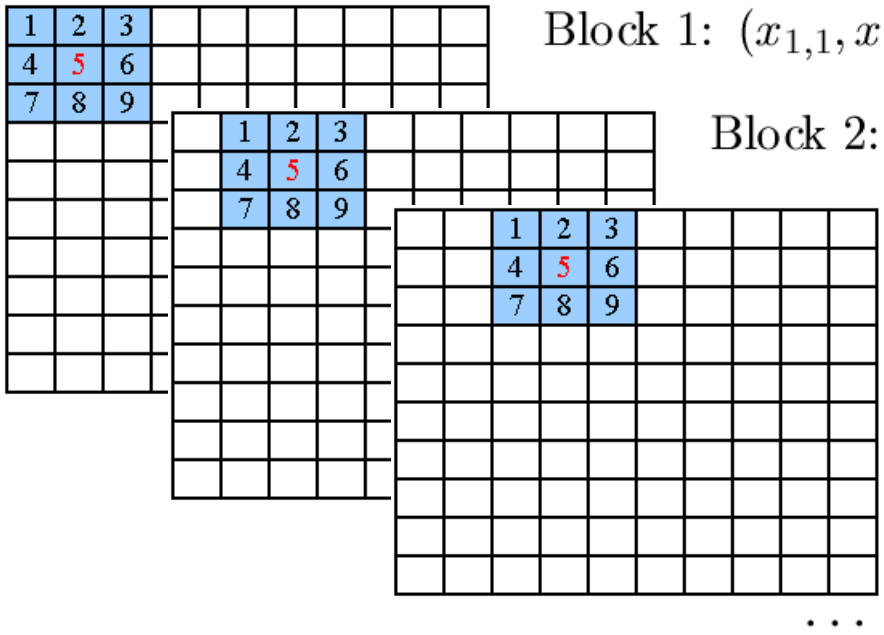


http://datahacker.rs/opencv-average-and-gaussian-filter/

- Gaussian filter: https://en.wikipedia.org/wiki/Gaussian_filter
- Gaussian Filter-概念與實作: https://medium.com/@bob800530/python-gaussian-filter-%E6%A6%82%E5%BF%B5%E8%88%87%E5%AF%A6%E4%BD%9C-676aac52ea17
- 空間濾波: http://ccy.dd.ncu.edu.tw/~chen/course/vision/ch5/ch5.htm
- Convolution in depth: https://sgugger.github.io/convolution-in-depth.html

Image Feature Extraction: Local Blocks

Block length = 3



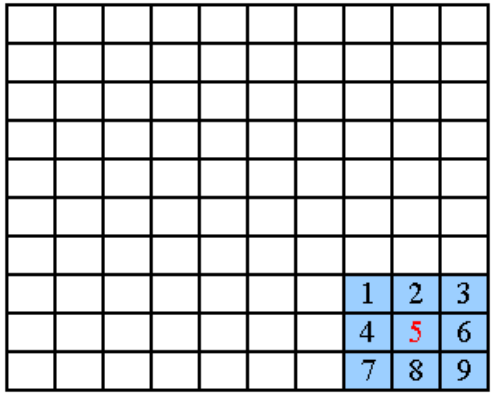
Block 1: $(x_{1,1}, x_{1,2}, \cdots, x_{1,9})$

Block 2: $(x_{2,1}, x_{2,2}, \cdots, x_{2,9})$

Block 3: $(x_{3,1}, x_{3,2}, \cdots, x_{3,9})$

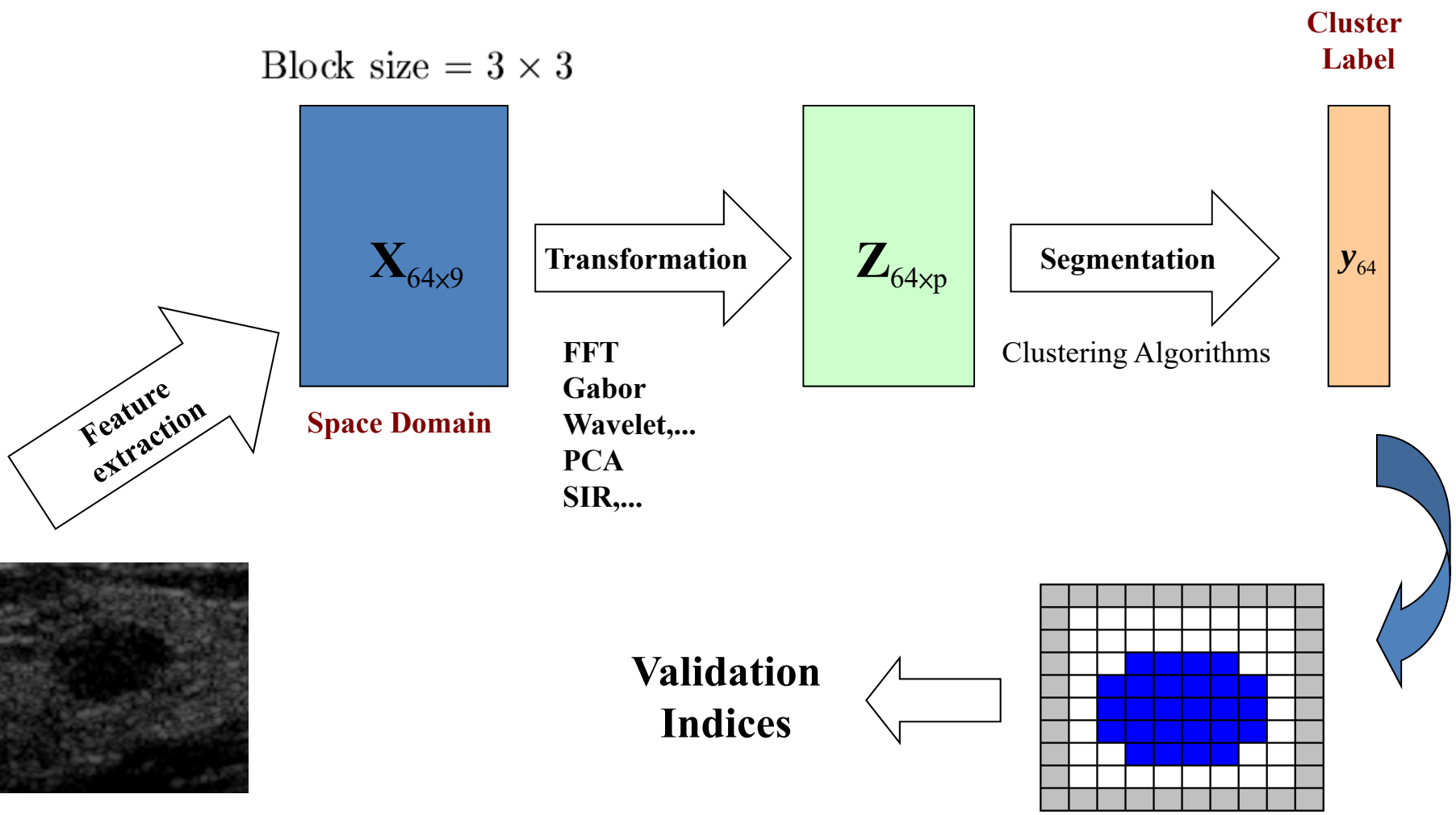
$\cdots$

Block 64: $(x_{64,1}, x_{64,2}, \cdots, x_{64,9})$



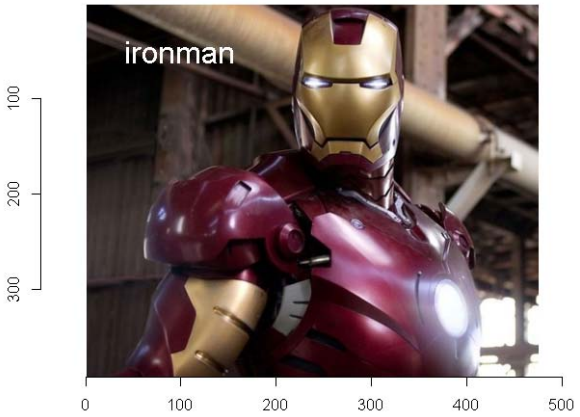
grey level: *f(x,y)=0,...,255*

Image Segmentation Procedure



imager Package: read image

```
> #install.packages("imager")
> library(imager)
> file <- "image_other/ironman.jpg"
> # im <- boats
> # im <- load.example("hubble")
> im <- load.image(file)
> im
Image. Width: 476 pix Height: 392 pix Depth: 1 Colour channels: 3
> dim(im)
[1] 476 392   1   3
> str(im)
 'cimg' num [1:476, 1:392, 1, 1:3] 0.00392 0.00784 0.00784 0.00784 0.01569 ...
> class(im)
[1] "cimg"  "imager_array"  "numeric"
> plot(im)
> text(100, 50, label="ironman", col="white", cex=2)
> width(im)
[1] 476
> height(im)
[1] 392
> #depth(im)
> spectrum(im)
[1] 3
> # arithmetic operations
> # 0-1 (dark to light)
> range(im)
[1] 0 1
> mean(im)
[1] 0.289028
> sd(im)
[1] 0.2607956
```

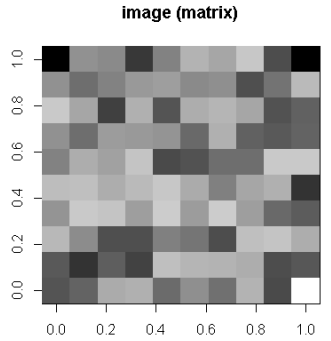


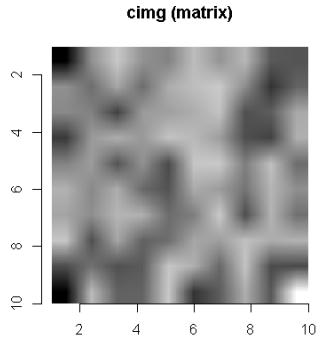

```
as.matrix(im)
as.data.frame(im)
```

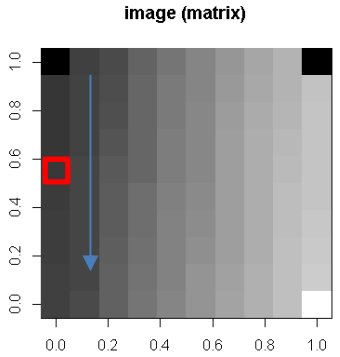
```
> log(im)+3*sqrt(im)
Image. Width: 476 pix Height: 392 pix Depth: 1 Colour channels: 3
> im[1:10, 1:5, 1, 1]
              [,1]        [,2]        [,3]        [,4]        [,5]
 [1,] 0.003921569 0.000000000 0.000000000 0.000000000 0.003921569
...
[10,] 0.007843137 0.011764706 0.011764706 0.011764706 0.011764706
>
> imager::save.image(im, "image-new.jpg")
```

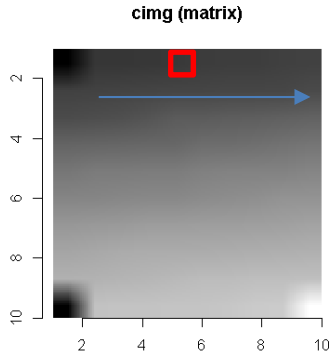
The cimg class

```
> # cimg class: a regular 4d array with an S3 class
> # construct a cimg image
> nc <- 10
> nr <- 10
> set.seed(12345)
> #noise <- matrix(runif(nc*nr, min = 0.2, max = 0.8), ncol=nc, nrow=nr)
> noise <- matrix(sort(runif(nc*nr, min = 0.2, max = 0.8)), ncol=nc, nrow=nr)
> noise[1, 1] <- 0
> noise[1, 10] <- 0
> noise[10, 10] <- 1
> noise[1:5, 1:5]
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0000000 0.2472880 0.3126275 0.3950572 0.4414911
[2,] 0.2035926 0.2482016 0.3278153 0.3960514 0.4575193
[3,] 0.2051887 0.2573843 0.3313687 0.4014830 0.4613183
[4,] 0.2115169 0.2804190 0.3358803 0.4169753 0.4722368
[5,] 0.2207213 0.2867071 0.3419483 0.4220625 0.4738886
> im.noise <- as.cimg(noise)
> par(mfrow=c(1, 2))
> image(t(noise)[, 10:1], main="image (matrix)", col=gray(0:100/100))
> plot(im.noise, main="cimg (matrix)")
>
> head(as.data.frame(im.noise))
  x y     value
1 1 1 0.0000000
2 2 1 0.2035926
3 3 1 0.2051887
4 4 1 0.2115169
5 5 1 0.2207213
6 6 1 0.2260739
```



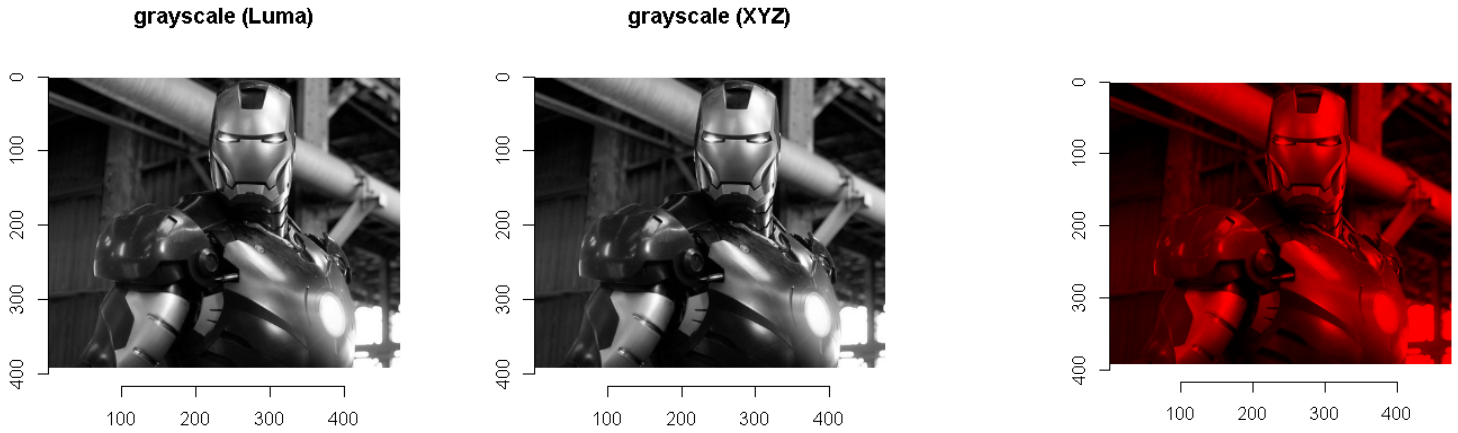






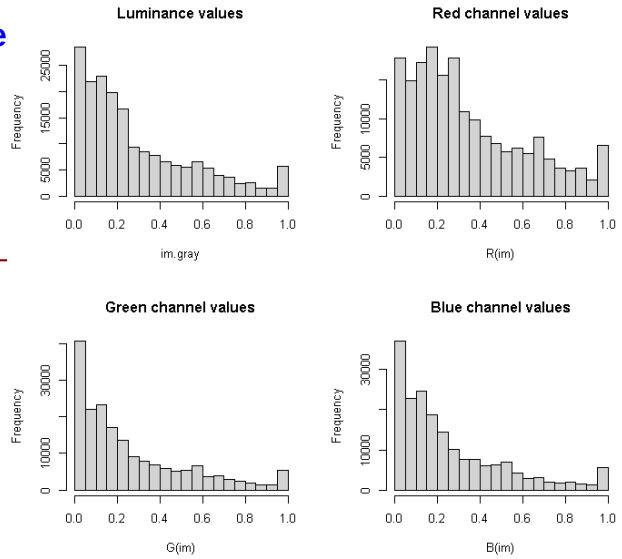
gray scale, colour channel

```
> par(mfrow=c(1, 2))
> plot(grayscale(im, method = "Luma"), main="grayscale (Luma)")
> plot(grayscale(im, method = "XYZ"), main="grayscale (XYZ)")
```



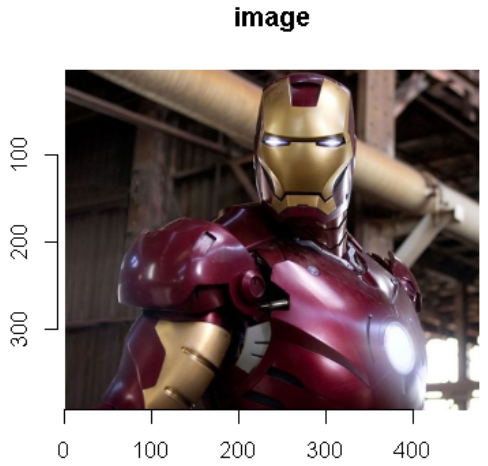

```
> R(im)  # same as channel(im, 1) ,G(im) ,B(im)
Image. Width: 476 pix Height: 392 pix Depth: 1 Colour channe
>
> im.temp <- im
> G(im.temp) <- 0
> B(im.temp) <- 0
> plot(im.temp)
```

```
> im.gray <- grayscale(im)
> par(mfrow=c(2, 2))
> hist(im.gray, main="Luminance values")
> hist(R(im), main="Red channel values")
> hist(G(im), main="Green channel values")
> hist(B(im), main="Blue channel values")
```

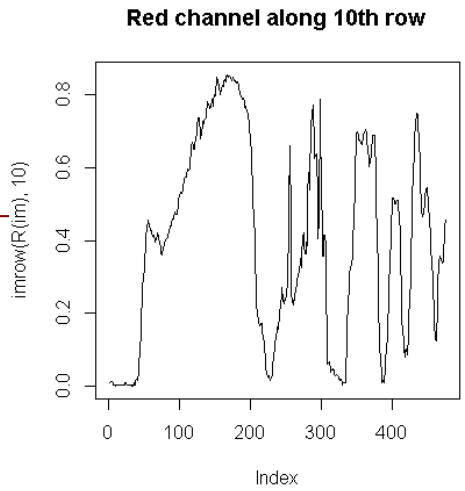


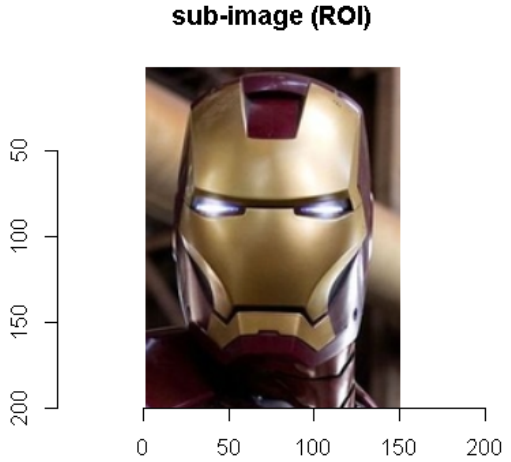
Sub-images

```
> imsub(im, x < 30) #Only the first 30 columns
Image. Width: 29 pix Height: 392 pix Depth: 1 Colour channels: 3
> imsub(im, y < 30) #Only the first 30 rows
> imsub(im, x < 30, y < 30) #First 30 columns and rows
> imsub(im, sqrt(x) > 8) #Can use arbitrary expressions
> imsub(im, x > height/2, y > width/2)
> imsub(im, cc==1) # Colour axis is "cc"
>
> par(mfrow=c(1, 2))
> plot(im, main="image")
> im.ROI <- imsub(im, 200 < x & x <= 350, 0 < y & y <= 200)
> plot(im.ROI, main="sub-image (ROI)")
```



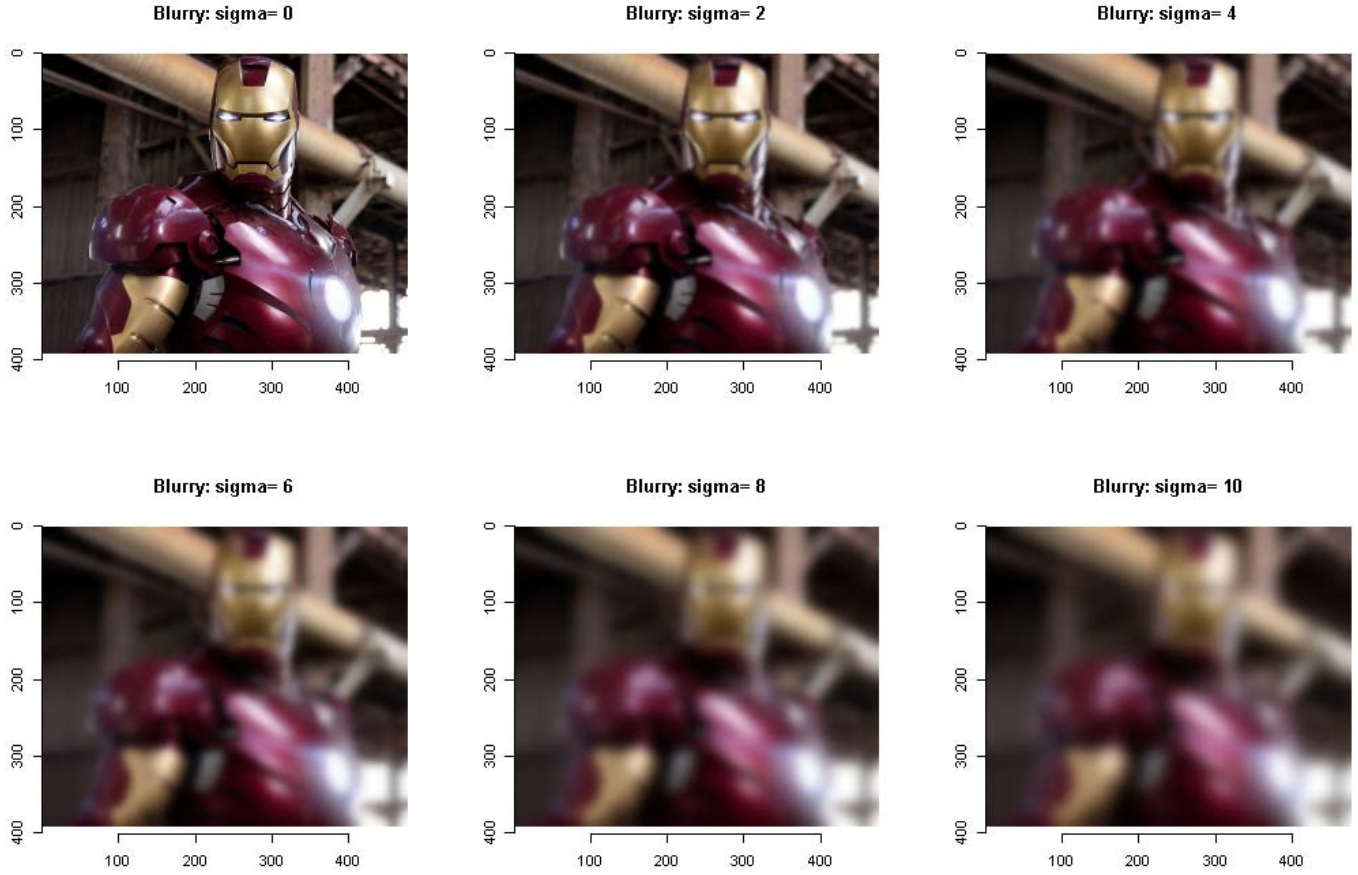

```
> # Rows, columns
> plot(imrow(R(im), 10), main="Red channel along 10th row",
type="l")
>
> # individual pixels
> at(im, x=20, y=20, cc=1:3)
[1] 0.011764706 0.003921569 0.007843137
> color.at(im, x=20, y=20)
[1] 0.011764706 0.003921569 0.007843137
```





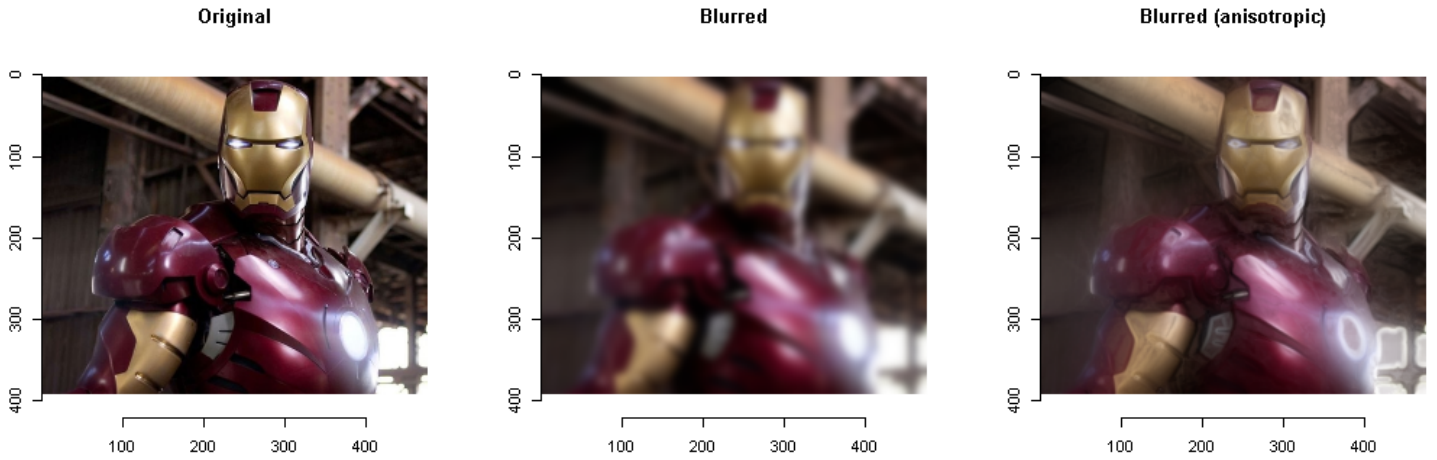
Blurry

```
> par(mfrow=c(2, 3))
> for(i in seq(0, 10, 2)){
+   im.blurry <- isoblur(im, sigma=i)
+   plot(im.blurry, main=paste("Blurry: sigma=", i))
+ }
```



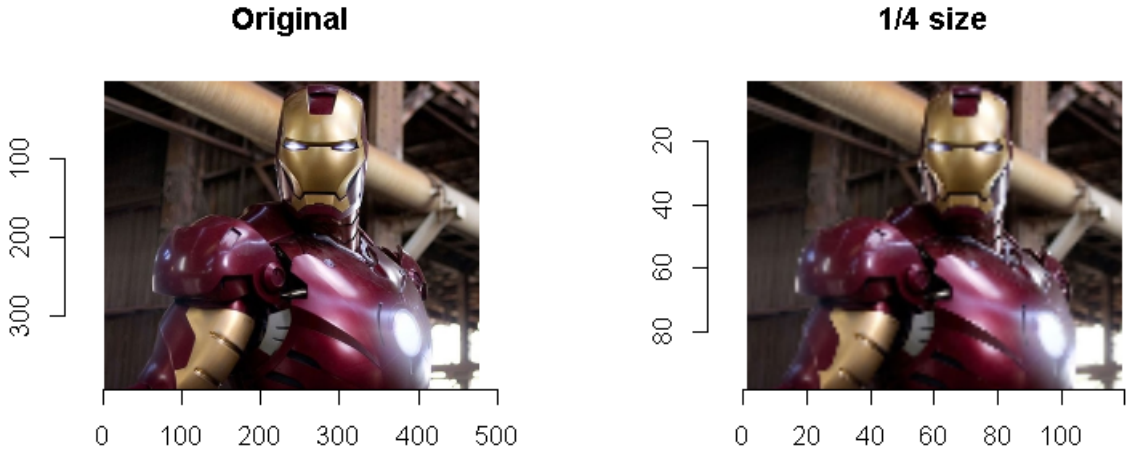
Denoising

```
> par(mfrow=c(1, 3))
> plot(im, main="Original")
> plot(isoblur(im, 5), main="Blurred")
> im.blur.ani <- blur_anisotropic(im, ampl=1e3, sharp=0.3)
> plot(im.blur.ani, main="Blurred (anisotropic)")
```



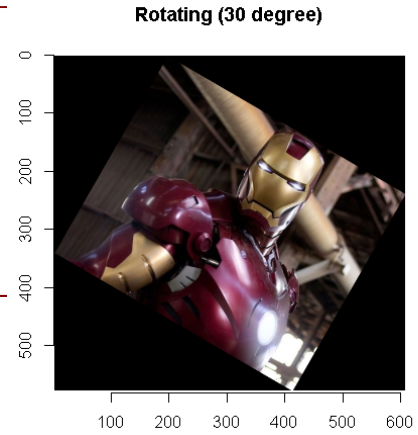
Resizing

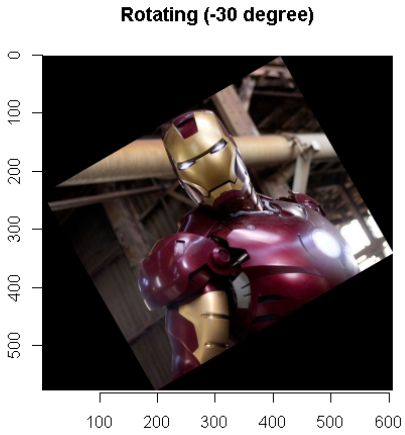
```
> par(mfrow=c(1, 2))
> plot(im, main="Original")
> im.thumb <- resize(im, round(width(im)/4),round(height(im)/4))
> plot(im.thumb, main="1/4 size")
>
> #Same as above: negative arguments are interpreted as percentages
> im.thumb <- resize(im, -25, -25)
```



Rotation, Shifting

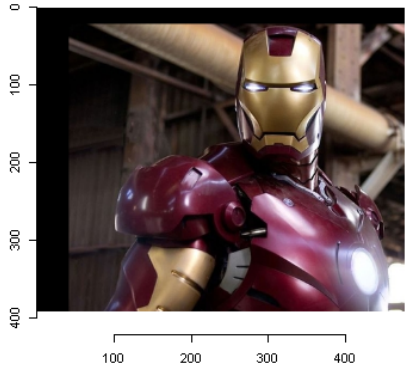
```
> par(mfrow=c(1, 2))
> plot(imrotate(im, 30), main="Rotating (30 degree)")
> plot(imrotate(im, -30), main="Rotating (-30 degree)")
```

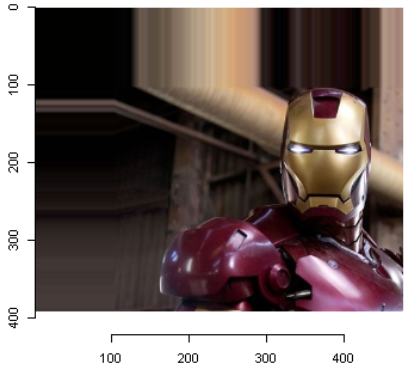


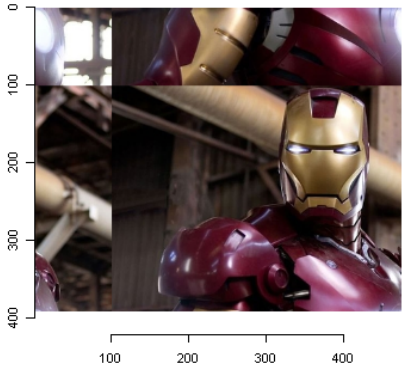


Question?
How to do the flipping?
Hint: `dim(img)`

```
> im.shift1 <- imshift(im, 40, 20)
> im.shift2 <- imshift(im, 100, 100, boundary=1)
> im.shift3 <- imshift(im, 100, 100, boundary=2)
> par(mfrow=c(1, 3))
> plot(im.shift1, main="Shifting")
> plot(im.shift2, main="Shifting (Neumann boundaries)")
> plot(im.shift3, main="Shifting (circular)")
```





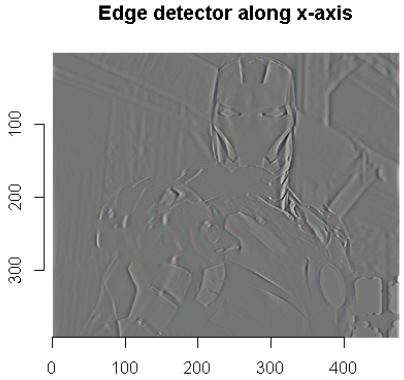


Edge detection: apply recursive Deriche filter

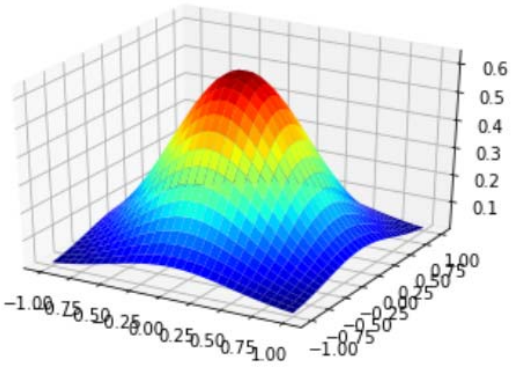
Gaussian (and derivative-of-Gaussian) filters

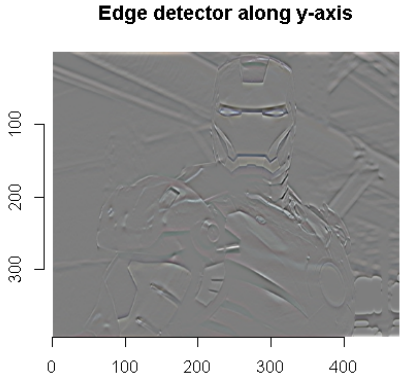
- The Deriche filter (**deriche**): a fast approximation to a Gaussian filter (order = 0), or Gaussian derivatives (order = 1 or 2).
- The Young-van Vliet filter (**vanvliet**):a fast approximation to a Gaussian filter (order = 0), or Gaussian derivatives (order = 1 or 2).

```
> par(mfrow=c(1, 2))
> im.edges.x <- deriche(im, sigma=2, order=2, axis="x")
> plot(im.edges.x, main="Edge detector along x-axis")
> im.edges.y <- deriche(im, sigma=2, order=2, axis="y")
> plot(im.edges.y, main="Edge detector along y-axis")
```



$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$



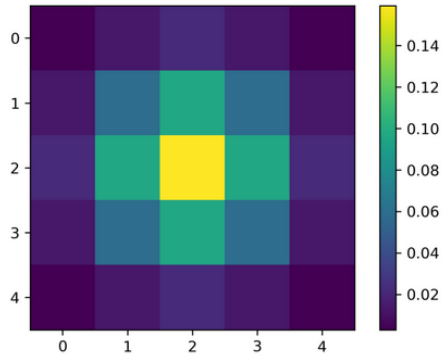


$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Gaussian filter

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$H(u,v)$

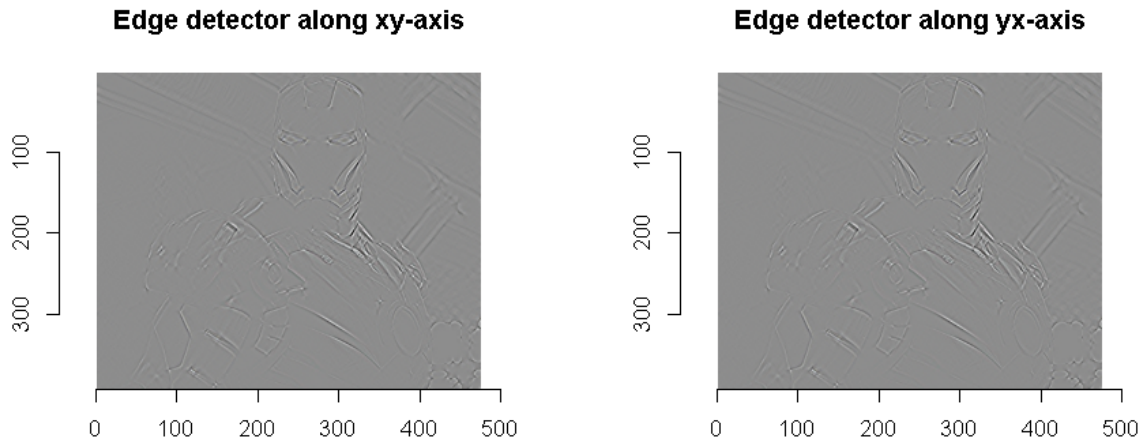


http://datahacker.rs/opencv-average-and-gaussian-filter/

https://en.wikipedia.org/wiki/Deriche_edge_detector

Edge detection: apply recursive Deriche filter

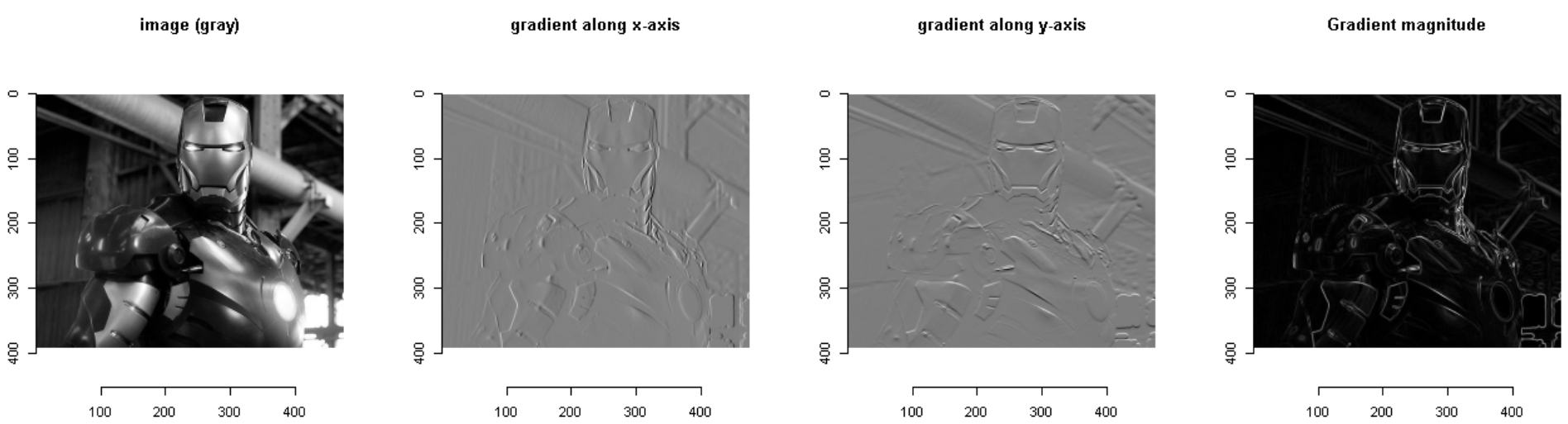
```
> par(mfrow=c(1, 2))
> im.edges.xy <- deriche(im.edges.x, 2, order=2, axis="y")
> im.edges.yx <- deriche(im.edges.y, 2, order=2, axis="x")
> plot(im.edges.xy, main="Edge detector along xy-axis")
> plot(im.edges.yx, main="Edge detector along yx-axis")
```



https://en.wikipedia.org/wiki/Deriche_edge_detector

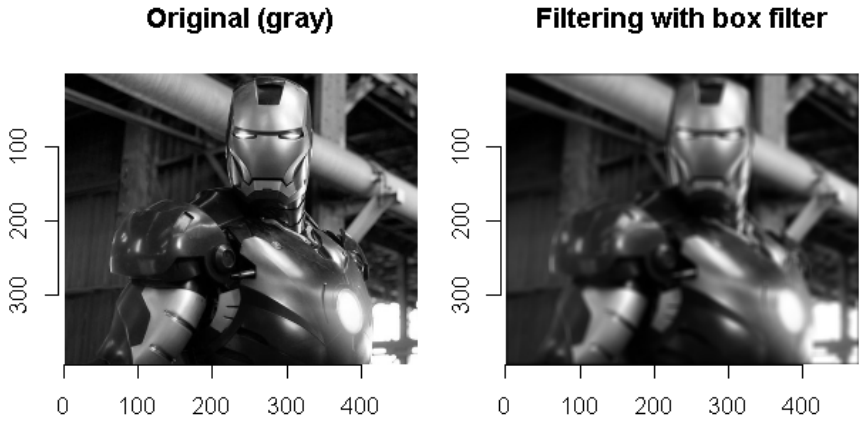
Edge detection: gradient

```
> par(mfrow=c(1, 4))
> im.gray <- grayscale(im)
> im.dx <- imgradient(im.gray, "x")
> im.dy <- imgradient(im.gray, "y")
> # same as im.dxy <- imgradient(im.gray, "xy")
>
> plot(im.gray, main="image (gray)")
> plot(im.dx, main="gradient along x-axis")
> plot(im.dy, main="gradient along y-axis")
>
> im.grad.mag <- sqrt(im.dx^2+im.dy^2)
> plot(im.grad.mag, main="Gradient magnitude")
```

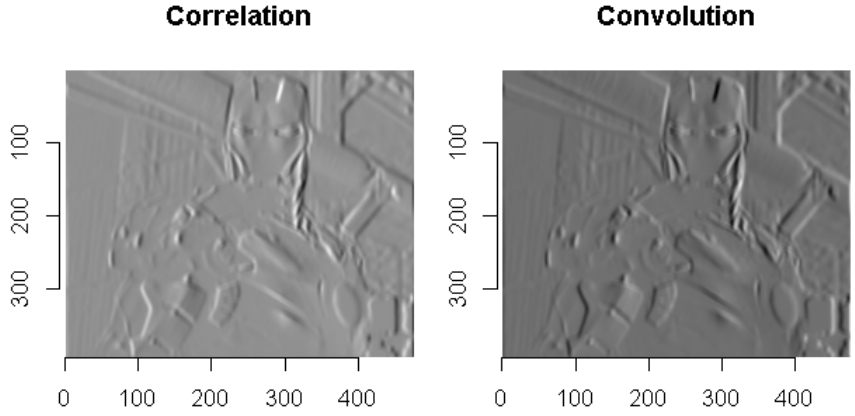


Box filter, Edge filter

```
> # 8x8 Box filter
> box.8x8 <- as.cimg(matrix(1, 8, 8))
> box.8x8[]
, , 1, 1
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    1    1    1    1    1    1
...
[8,]    1    1    1    1    1    1    1    1
> im.bf <- correlate(im.gray, box.8x8)
> par(mfrow=c(1, 2))
> plot(im.gray, main="Original (gray)")
> plot(im.bf, main="Filtering with box filter")
```

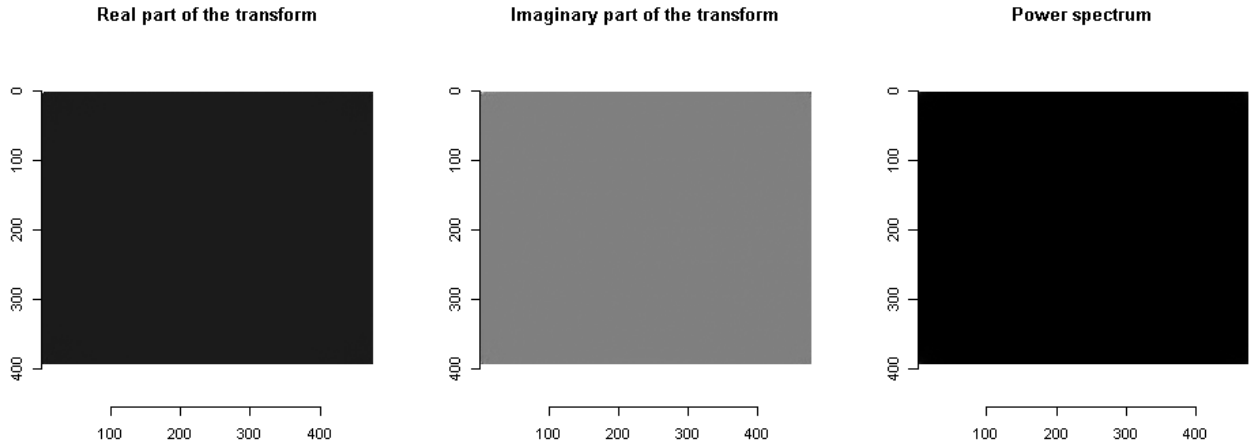



```
> edge.filter <- as.cimg(function(x,y) sign(x-5), 9, 9)
> edge.filter[]
, , 1, 1
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   -1   -1   -1   -1   -1   -1   -1   -1   -1
...
 [4,]   -1   -1   -1   -1   -1   -1   -1   -1   -1
 [5,]    0    0    0    0    0    0    0    0    0
 [6,]    1    1    1    1    1    1    1    1    1
...
 [9,]    1    1    1    1    1    1    1    1    1
> im.corr <- correlate(im.gray, edge.filter)
> im.conv <- convolve(im.gray, edge.filter)
> par(mfrow=c(1, 2))
> plot(im.corr, main="Correlation")
> plot(im.conv, main="Convolution")
```

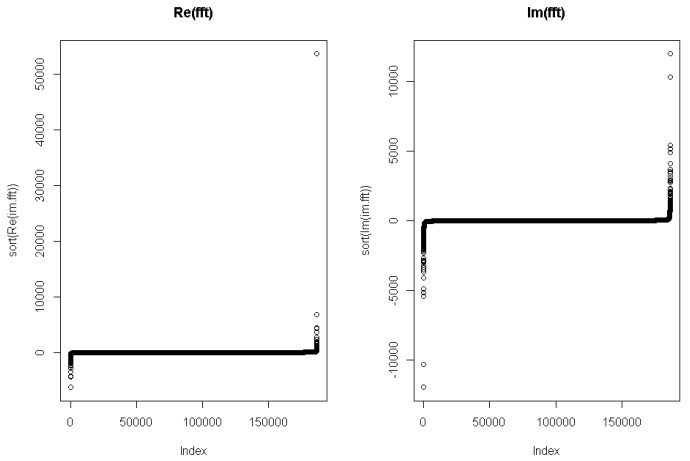


FFTs and the periodic decomposition

```
> im.fft <- fft(as.matrix(im.gray))
> par(mfrow=c(1, 3))
> plot(as.cimg(Re(im.fft)), main="Real part of the transform")
> plot(as.cimg(Im(im.fft)), main="Imaginary part of the transform")
> im.fft.ps <- as.cimg(sqrt(Re(im.fft)^2+Im(im.fft)^2))
> plot(im.fft.ps, main="Power spectrum")
```



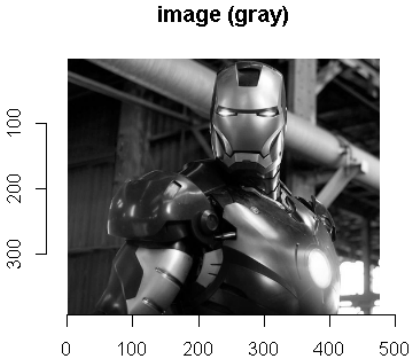

```
> par(mfrow=c(1, 2))
> plot(sort(Re(im.fft)), main="Re(fft)")
> plot(sort(Im(im.fft)), main="Im(fft)")
```

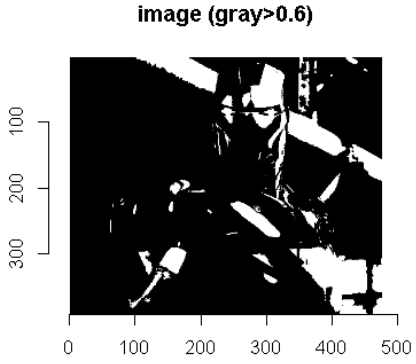


Pixel sets (pixsets)

https://cran.r-project.org/web/packages/imager/vignettes/pixsets.html
A pixset is a set of pixels, represented as a binary image.
Pixsets represent sets of pixels, or equivalently binary images (masks)

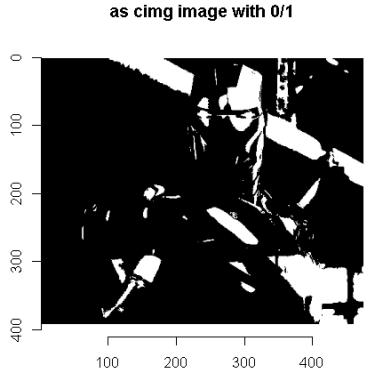
```
> im.gray.px <- im.gray > 0.6 #Select pixels with high luminance
> str(im.gray.px)
 'pixset' logi [1:476, 1:392, 1, 1] FALSE FALSE FALSE FALSE FALSE FALSE ..
> # The  "TRUE" (white) values correspond to pixels in the set,
> # and  "FALSE" (black) to pixels not in the set.
> par(mfrow=c(1, 2))
> plot(im.gray, main="image (gray)")
> plot(im.gray.px, main="image (gray>0.6)")
```





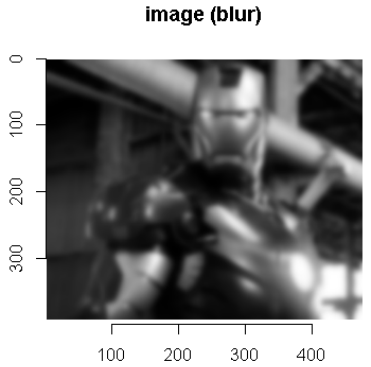

```
> # Coordinates for pixels
> pixsets <- where(im.gray.px)
> head(pixsets)
    x y
1 101 1
2 102 1
...
6 106 1
> # Number of pixels in set
> sum(im.gray.px)
[1] 26288
> mean(im.gray.px) # Proportion
[1] 0.1408849
> # Indexing using pixsets
> mean(im.gray[im.gray.px])
[1] 0.7870543
```

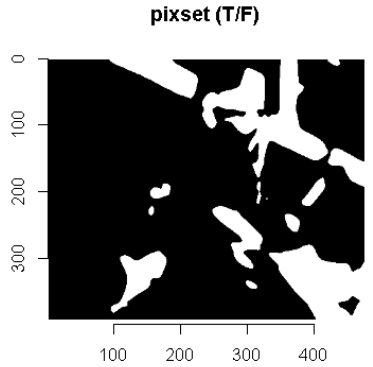
```
> # coordinates of subset of pixels
> im.cond <- get.locations(im,
        condition=function(x) x < 0.5)
> head(im.cond)
  x y z cc
1 1 1 1  1
2 2 1 1  1
..
6 6 1 1  1
> im.tmp <- as.cimg(im.gray.px)
> im.tmp
Image. Width: 476 pix Height: 392 pix
Depth: 1 Colour channels: 1
> plot(im.tmp, main="as cimg image
with 0/1")
```



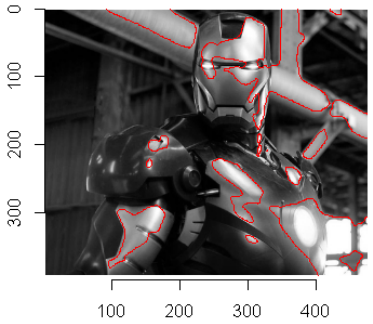
Segmentation using pixsets (EX1)

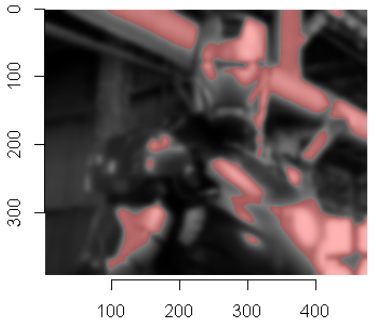
```
> par(mfrow=c(1, 2))
> im.blur <- isoblur(im.gray, 4)
> plot(im.blur, main="image (blur)")
> im.blur.px <- im.blur  > 0.5
> plot(im.blur.px, main="pixset (T/F)")
```



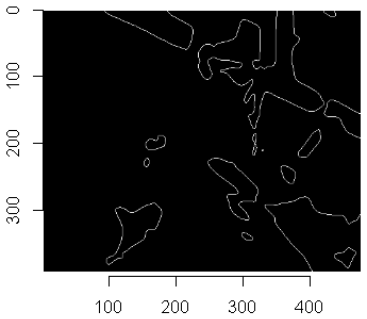


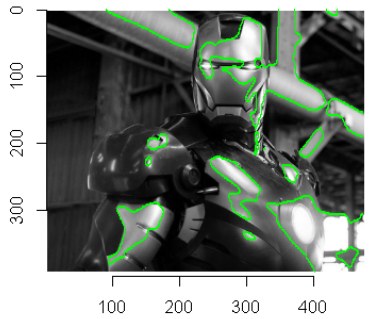

```
> plot(im.gray, main="segmentation (highlight)")
> highlight(im.blur.px)
>
> im.seg.color <- colorise(im.blur, im.blur.px,
                           "red", alpha=0.3)
> plot(im.seg.color, main="segmentation (color)")
```






```
> # Boundaries
> im.seg.bound <- boundary(im.blur.px)
> plot(im.seg.bound, main="pixset (boundary: T/F)")
>
> plot(im.gray, main="segmentation (boundary)")
> points(where(im.seg.bound), cex=0.1, col="green")
```

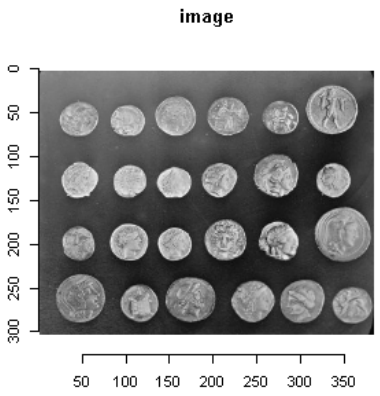


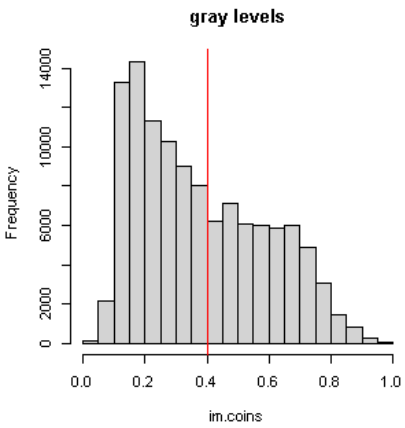


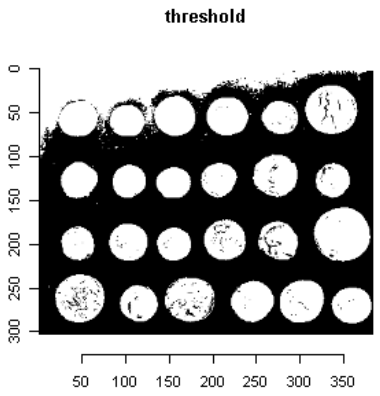
Segmentation using pixsets (EX2)

See https://cran.r-project.org/web/packages/imager/vignettes/pixsets.html

```
> im.coins <- load.example("coins")
> par(mfrow=c(1, 3))
> plot(im.coins, main="image")
> str(im.coins)
 'cimg' num [1:384, 1:303, 1, 1] 0.184 0.482 0.522 0.506 0.537 ...
> hist(im.coins, main="gray levels")
> lambda <- 0.4
> abline(v=lambda, col="red")
> plot(threshold(im.coins, thr=lambda), main="threshold")
>
> # correct the illumination using a linear model by removing the trend
> par(mfrow=c(2, 3))
> im.coins.df <- as.data.frame(im.coins)
> str(im.coins.df)
'data.frame':       116352 obs. of  3 variables:
 $ x    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ y    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ value: num  0.184 0.482 0.522 0.506 0.537 ...
> head(im.coins.df)
  x y     value
1 1 1 0.1843137
2 2 1 0.4823529
3 3 1 0.5215686
4 4 1 0.5058824
5 5 1 0.5372549
6 6 1 0.5176471
```

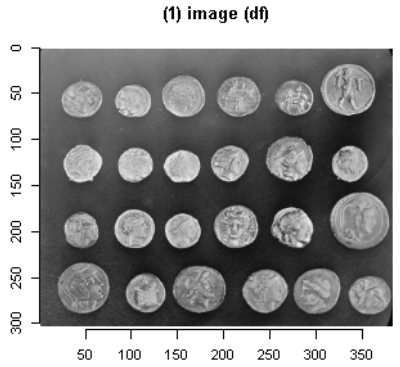


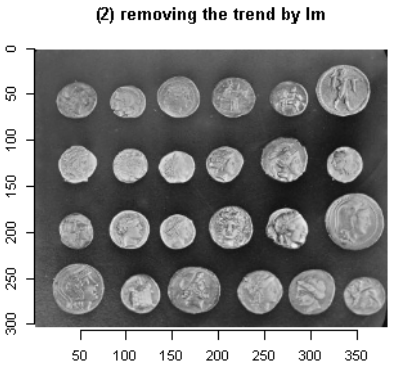


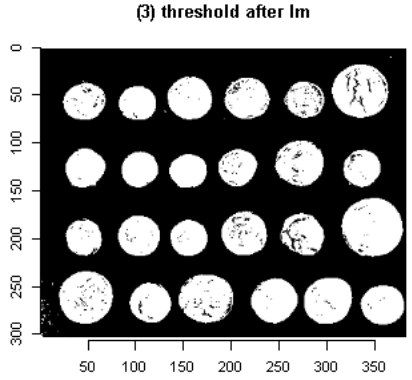


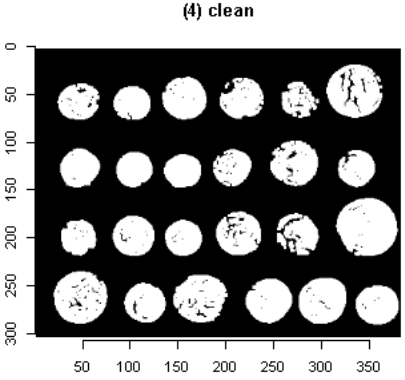
Segmentation using pixsets (EX2)

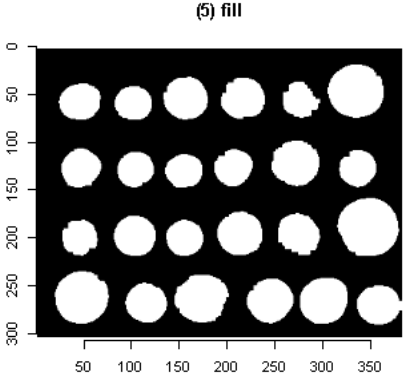
```
> plot(as.cimg(im.coins.df, dims=dim(im.coins)), main="(1) image (df)")
>
> im.coins.df.sub <- im.coins.df[sample(nrow(im.coins.df), 10000), ]
> mylm <- lm(value ~ x * y, data=im.coins.df.sub)
> im.coins.lm <- im.coins.df
> im.coins.lm$value <- im.coins.df$value - predict(mylm, im.coins.df)
> im.coins.lm.cimg <- as.cimg(im.coins.lm, dims=dim(im.coins))
> plot(im.coins.lm.cimg, main="(2) removing the trend by lm")
> im.coins.lm.cimg.thr <- threshold(im.coins.lm.cimg, thr="auto")
> plot(im.coins.lm.cimg.thr, main="(3) threshold after lm")
>
> im.coins.lm.cimg.thr.clean <- clean(im.coins.lm.cimg.thr, 3)
> im.coins.lm.cimg.thr.clean.fill <- fill(im.coins.lm.cimg.thr.clean, 7)
> plot(im.coins.lm.cimg.thr.clean, main="(4) clean")
> plot(im.coins.lm.cimg.thr.clean.fill, main="(5) fill")
> plot(im.coins, main="(6) segmentation result")
> highlight(im.coins.lm.cimg.thr.clean.fill)
```

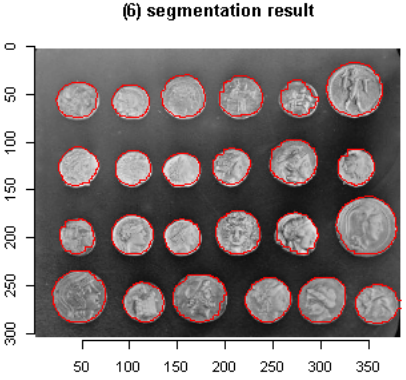










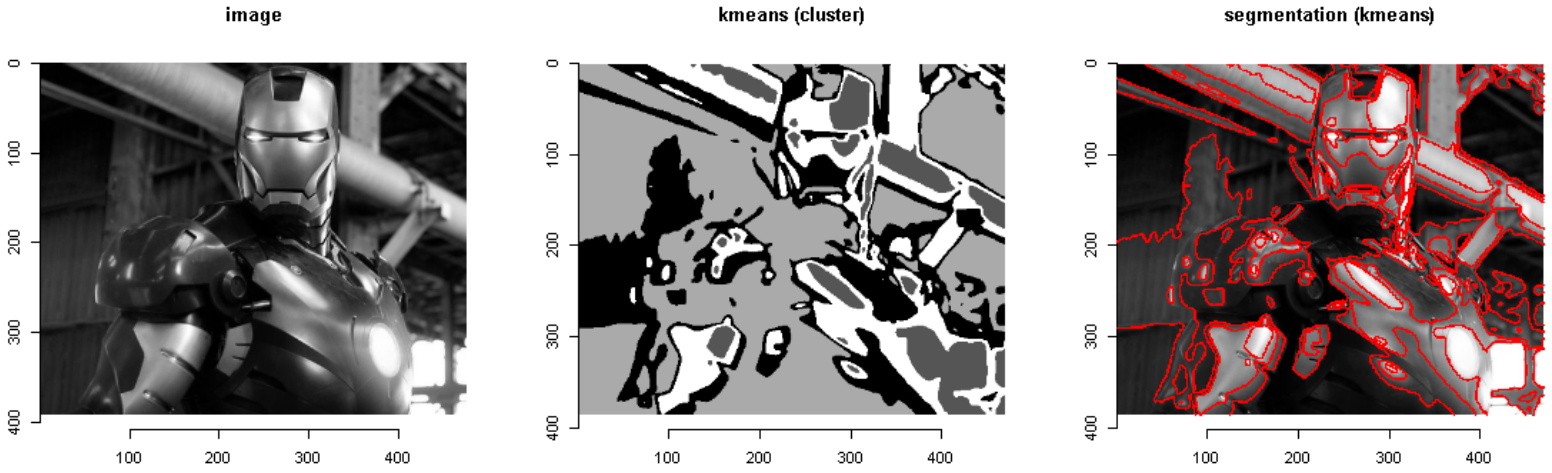


Segmentation (using clustering)


```
> dim(im.gray)
[1] 476 392   1   1
> im.nrow <- width(im.gray)
> im.ncol <- height(im.gray)
>
> block.length <- 7
> st.x <- (block.length+1)/2
> en.x <- width(im)-(block.length-1)/2
> st.y <- (block.length+1)/2
> en.y <- height(im)-(block.length-1)/2
> cx <- rep(st.x:en.x, height(im)-block.length+1)
> cy <- rep(st.y:en.y, each=width(im)-block.length+1)
> out <- extract_patches(im.gray, cx, cy, wx=block.length, wy=block.length)
> out
Image list of size 181420
> image.X.4x4 <- data.frame(matrix(unlist(out), nrow=length(out), byrow=T))
> dim(image.X.4x4)
[1] 181420     49
> head(image.X.4x4)
           X1          X2          X3          X4          X5          X6          X7
1 0.003490196 0.006980392 0.007843137 0.007843137 0.015686275 0.015686275 0.007843137
2 0.006980392 0.007843137 0.007843137 0.015686275 0.015686275 0.007843137 0.011764706
..
          X43         X44         X45         X46         X47         X48         X49
1 0.007843137 0.007843137 0.011764706 0.015686275 0.015686275 0.015686275 0.007843137
...
6 0.015686275 0.007843137 0.007843137 0.003921569 0.003921569 0.003921569 0.003921569
> seg.nrow <- im.nrow-block.length+1
> seg.ncol <- im.ncol-block.length+1
```

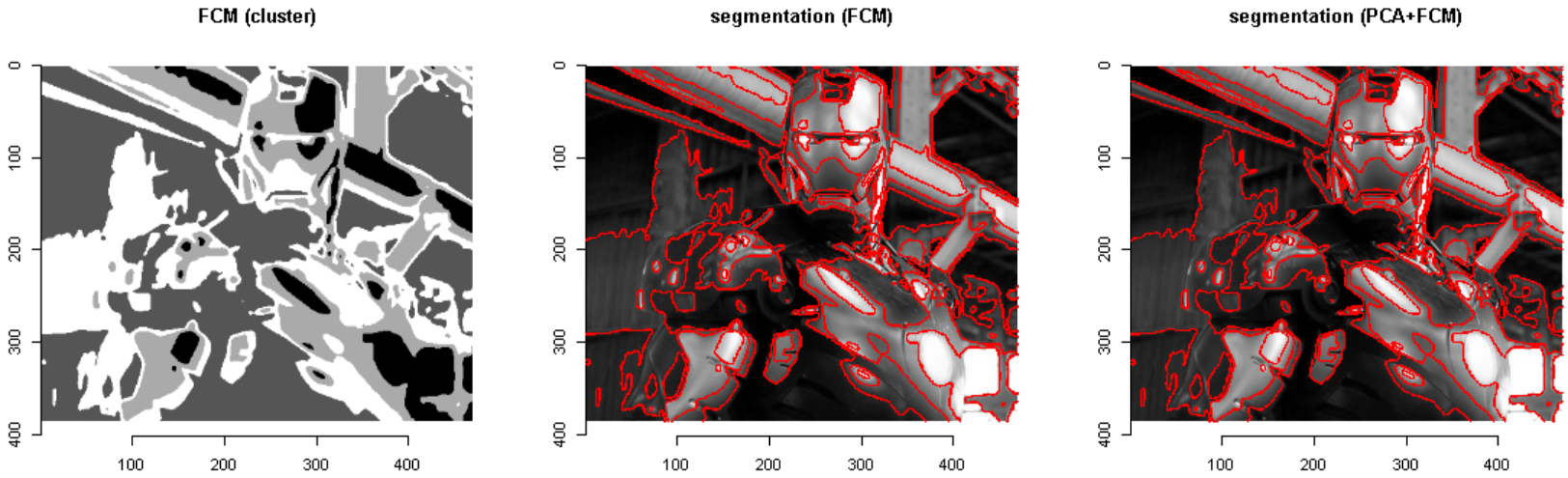
Segmentation (using clustering)

```
> # K-means
> no.class <- 4
> kmeans.seg <- kmeans(image.X.4x4, no.class)$cluster
>
> # K-means: reconstruction
> im.kmeans.seg <- as.cimg(matrix(kmeans.seg, nrow=seg.nrow, ncol=seg.ncol))
> dim(im.kmeans.seg)
[1] 470 386   1   1
> par(mfrow=c(1, 3))
> plot(im.gray, main="image")
> plot(im.kmeans.seg, main="kmeans (cluster)")
> im.gray.sub <- imsub(im.gray, st.x <= x & x <= en.x, st.y <= y & y <= en.y)
> dim(im.gray.sub)
[1] 470 386   1   1
> plot(im.gray.sub, main="segmentation (kmeans)")
> highlight(im.kmeans.seg)
```



Segmentation (using clustering)

```
> par(mfrow=c(1, 3))
> library(e1071)
> cmeans.seg <- cmeans(image.X.4x4, no.class)$cluster
> im.cmeans.seg <- as.cimg(matrix(cmeans.seg, nrow=seg.nrow, ncol=seg.ncol))
> par(mfrow=c(1, 3))
> plot(im.cmeans.seg, main="FCM (cluster)")
> plot(im.gray.sub, main="segmentation (FCM)")
> highlight(im.cmeans.seg)
>
> ## PCA+FCM
> pca.cmeans.seg <- cmeans(princomp(image.X.4x4, 3)$scores, no.class)$cluster
> im.pca.cmeans.seg <- as.cimg(matrix(pca.cmeans.seg, nrow=seg.nrow, ncol=seg.ncol))
> plot(im.gray.sub, main="segmentation (PCA+FCM)")
> highlight(im.pca.cmeans.seg)
```



More examples: http://dahtah.github.io/imager/foreground_background.html