Computer Vision
algorithms for R users

useR.2017, Brussels, July 2017

Jan Wijffels

BNOSAC - jwijffels@bnosac.be

bnosac

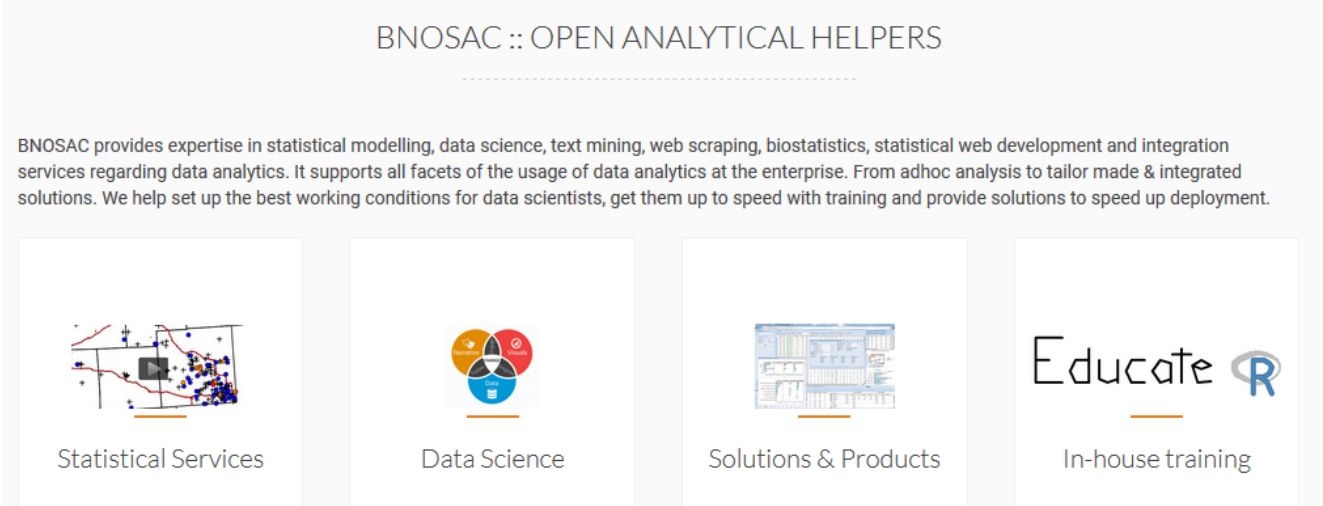
Overzicht

BNOSAC

www.bnosac.be: R & open source analytics experts

Providing consultancy services in **open source analytical engineering**

- Support for R / Oracle R Enterprise / Microsoft R / PostgreSQL / Python / ExtJS / Hadoop / …
- Expertise in predictive data mining, biostatistics, geostats, R + Python programming, GUI building, artificial intelligence, process automation, analytical web development
- R implementations, application maintenance & training/consulting
- Server Pro and Shiny Pro reseller
- Organise & support RBelgium
- Hosting CRAN at www.datatailor.be
- Contributing to the R community with R packages / R training



R training by BNOSAC: www.bnosac.be/training

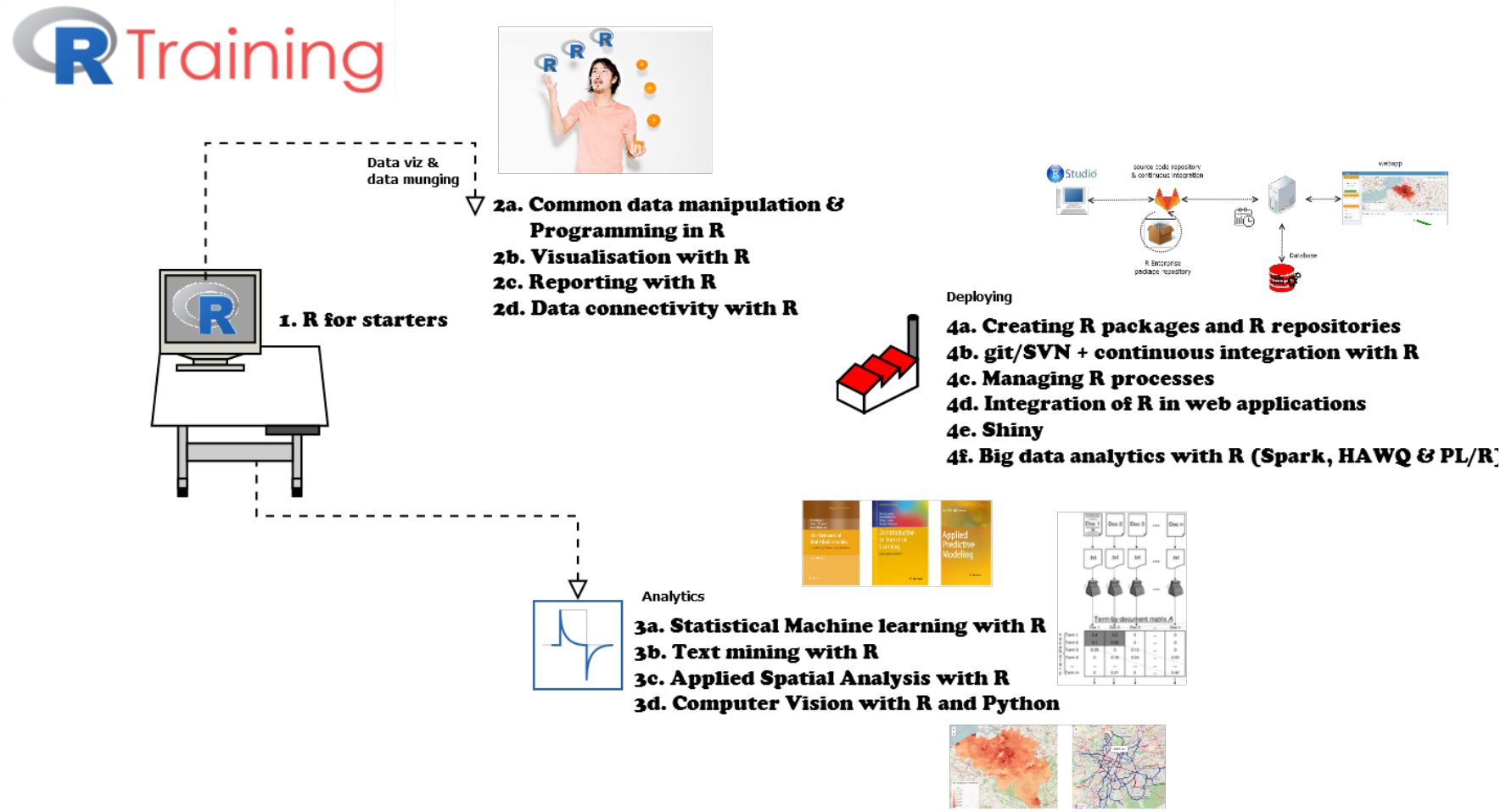


Figure: Training on R / Data Science

bnosac

open analytical helpers

Computer Vision - R toolset

Computer Vision with R - existing R tools

Quite some packages *exist* already

General manipulation and algorithms

magick (importing & converting to/from all formats / basic image manipulation); **imager** (interpolation, resizing, warping, filtering, fourier transforms, haar wavelets, morphological operations, denoising, segmentation, gradients, blurring); **EBImage**, used mainly for biological applications; **OpenImageR** (hashing, edge detection, manipulation)

Domain specific processing

R packages like **adimpro** (smoothing), **radiomics** (texture analysis), **fftw** (fourier transforms), **oro.dicom/oro.nifti** (brain images), **zooimage** (plankton analysis), **wvtool** (wood identification / filters), **Thermimage** (thermal images), **colordistance** (image clustering, color distances), **CRImage** (tumor detection), **spatstat** (Spatial Point Patterns).

R is good at interfacing

Rvision/**ROpenCVLite** (OpenCV from R). API's exist for traditional computer vision (Google Vision API, Microsoft Cognitive Services) or on top of deep learning tools like Keras (**kerasR**), Tensorflow (**tensorflow** R package).

Typical areas where bnosac.be used image recognition

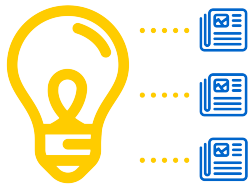


Figure: Use cases

- Identify products based on images
- Quality control of products in factories
- As input for further processing (predictive models / data enrichment)

6 new R packages by bnosac.be

Available at `https://github.com/bnosac/image`
Containing image algorithms lacking in other R packages.

- **image.CornerDetectionF9**: FAST-9 corner detection (BSD-2).
- **image.CannyEdges**: Canny Edge Detector (GPL-3).
- **image.LineSegmentDetector**: Line Segment Detector (LSD) (AGPL-3).
- **image.ContourDetector**: Unsupervised Smooth Contour Line Detection (AGPL-3).
- **image.dlib**: Speeded up robust features (SURF) and histogram of oriented gradients (FHOG) features (AGPL-3).
- **image.darknet**: Image classification using darknet with deep learning models AlexNet, Darknet, VGG-16, GoogleNet and Darknet19. As well object detection using the state-of-the art YOLO detection system (MIT).

More packages and extensions are under development.

bnosac

open analytical helpers

Detect Corners

New R package (1): image.CornerDetectionF9

- An implementation of the 'FAST-9' **corner detection** algorithm.
- Finds feature points (corners) in digital images. **These blobs can e.g. be used to track and map objects.**
- Idea of FAST-9
  - If a certain number of pixels around that pixel are all brighter or darker than the pixel itself, the pixel is a corner
  - brighter/darker is defined by a **threshold**

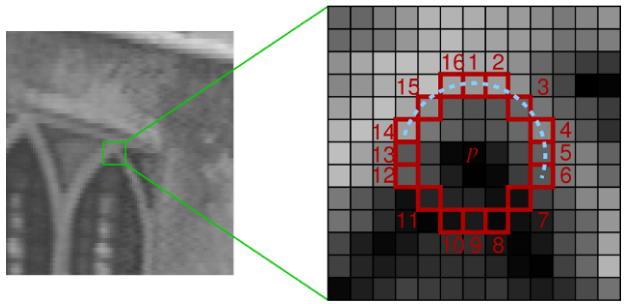


Figure: Fast-9 logic - see `https://www.edwardrosten.com/work/fast.html`

- R package is based on C code at `https://github.com/jcayzac/F9-Corner-Detection-Library`

- R function image_detect_corners requires as input
  - grayscale matrix with values in 0-255 range + set the threshold
  - suppress_non_max: if pixels are not part of the local maxima they are set to zero (to avoid 2 points in neighbourhood)

Figure: Fast-9 corner detection example

```
library(pixmap)
library(image.CornerDetectionF9)

## Read in a PGM grey scale image
image <- read.pnm(file = system.file("extdata", "chairs.pgm", package="image.CornerDetectionF9"),
                  cellres = 1)

## Detect corners
corners <- image_detect_corners(image@grey * 255,
                                threshold = 100,
                                suppress_non_max = TRUE)

## Plot the image and the corners
plot(image)
points(corners$x, corners$y, col = "red", pch = 20, lwd = 0.5)
```

bnosac

open analytical helpers

Detect Edges

New R package (2): image.CannyEdges

- Canny edge detector `https://en.wikipedia.org/wiki/Canny_edge_detector`,
- Detects edges in images. Logic:
  - Remove noise > Find out gradients > Keep only maximum gradient intensities (suppress non-max) > threshold to identify edges > remove edges which are very weak and not connected to strong edges

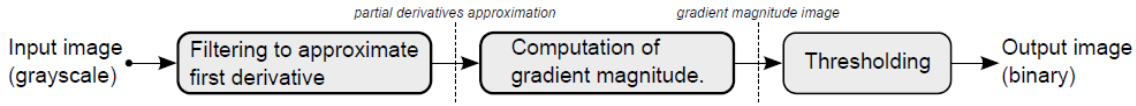
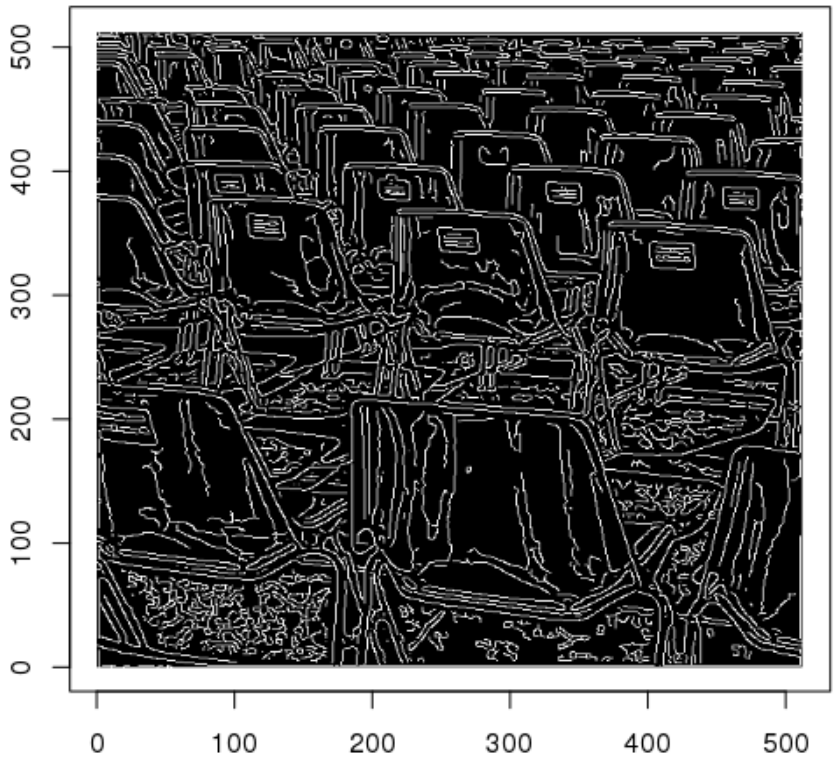


Figure: Edge detection logic - 1st derivative

- Based on C code from `https://github.com/Neseb/canny`, requiring libpng and fftw3 to be installed (as in sudo apt-get install libpng-dev fftw3 fftw3-dev pkg-config)
- For details on the math: `http://www.ipol.im/pub/art/2015/35`. Other edge detectors in R package OpenImageR.

Example below reads in a greyscale image and detects edges in the chairs.

```
library(pixmap)
library(image.CannyEdges)

## Read in a PGM grey scale image
image <- read.pnm(file = system.file("extdata", "chairs.pgm", package="image.CannyEdges"),
                  cellres = 1)

## Detect edges
edges <- image_canny_edge_detector(image@grey * 255, s = 2, low_thr = 3, high_thr = 10)

## Plot the edges
plot(edges)
```



bnosac

open analytical helpers

Detect Lines & Contours

New R package (3): image.LineSegmentDetector

- **Detection of lines** with the Line Segment Detector (LSD). Idea:
  - Contour lines are zones of the image where the gray level is changing fast enough from dark to light or the opposite.
  - Line support regions are being defined by the gradient and grouped into regions with the same direction.
  - Regions are put in rectangles which define segments

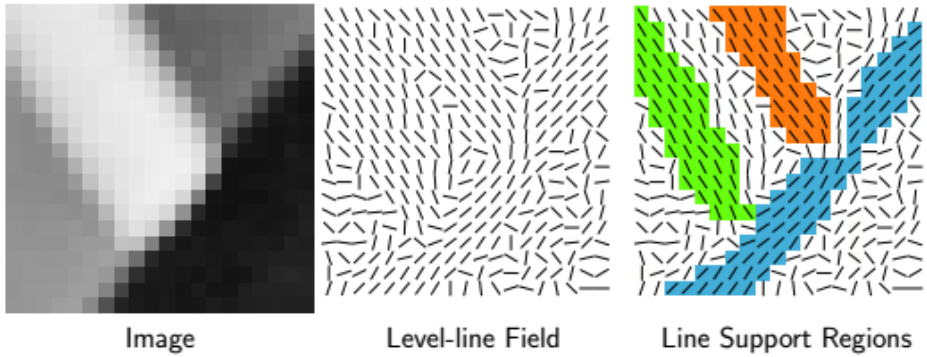


Figure: LSD - line support regions

- The default arguments provide very good line detections for any image
- Based on C code available at `https://doi.org/10.5201/ipol.2012.gjmr-lsd`

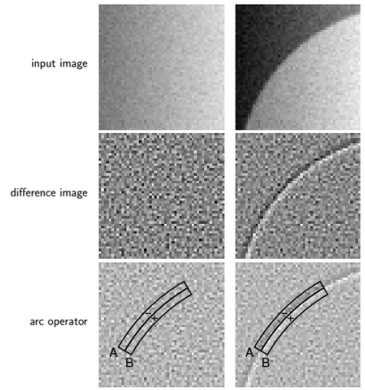
Example below reads in a greyscale image and detects lines in houses.

Figure: Detect Lines with LSD

```
library(pixmap)
library(image.LineSegmentDetector)

## Read in the PGM file
image <- read.pnm(file = system.file("extdata", "le-piree.pgm", package="image.LineSegmentDetector"),
                  cellres = 1)

## Detect the lines
linesegments <- image_line_segment_detector(image@grey * 255)
linesegments

## Plot the image + add the lines in red
plot(image)
plot(linesegments, add = TRUE, col = "red")
```

New R package (4): image.ContourDetector

- Detection of contour lines based on paper (Unsupervised Smooth Contour Detection, 2016-11-18, IPOL, `http://www.ipol.im/pub/art/2016/175`). Idea:
  - remove low frequencies (noise) from the input image.
  - contours are frontiers separating two adjacent regions, one with significantly larger values than the other.
  - based on existing edge detector curve candidates are proposed, the curves are piecewise approximated by circular arcs
  - significance based on non-parametric Mann-Whitney U test to determine whether the samples were drawn from the same distribution or not.
- The default arguments provide very good contour detections for any image. No need to set detection thresholds like in the Canny edge detection where noise has a big impact on.
- Based on C code available at `https://doi.org/10.5201/ipol.2016.175`



Example below reads in a greyscale image and detects contour lines in a car.

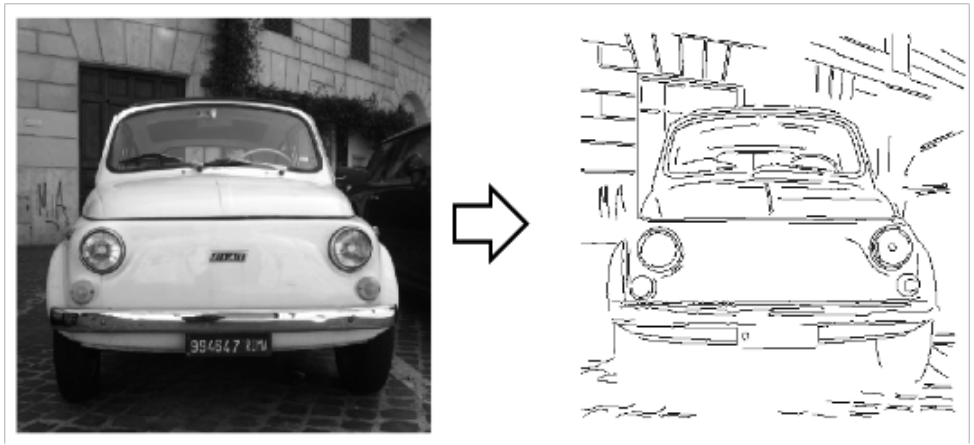


Figure: Detect ContourLines on car

```
library(pixmap)
library(image.ContourDetector)

## Read in the PGM file
image <- read.pnm(file = system.file("extdata", "image.pgm", package="image.ContourDetector"),
                  cellres = 1)

## Detect the contours
contourlines <- image_contour_detector(image@grey * 255)
contourlines

Contour Lines Detector
  found 192 contour lines

## Plot the contour lines
plot(image)
plot(contourlines)
```

Example below reads in a jpg image, converts it to a greyscale image and detects contour lines in the atomium.

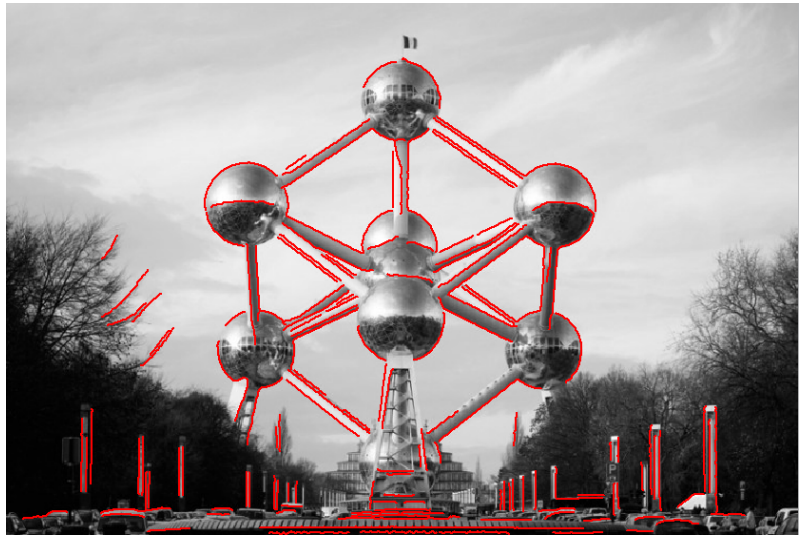


Figure: Detect ContourLines on the Atomium

```
library(image.ContourDetector)
library(magick)
## Convert jpg to PGM file using the magick package
x <- image_read(path = system.file("extdata", "atomium.jpg", package="image.LineSegmentDetector"))
x <- image_convert(x, format = "pgm", depth = 8)

## Save the PGM file
f <- tempfile(fileext = ".pgm")
image_write(x, path = f, format = "pgm")

## Read in the PGM file, detect the lines
image <- read.pnm(file = f, cellres = 1)
linesegments <- image_line_segment_detector(image@grey * 255)

## Overlay the contour lines on top of the plot
plot(image)
plot(linesegments, add = TRUE, col = "red")
```

bnosac

open analytical helpers

Identification / track points / predictive image features

New R package (5): image.dlib - SURF

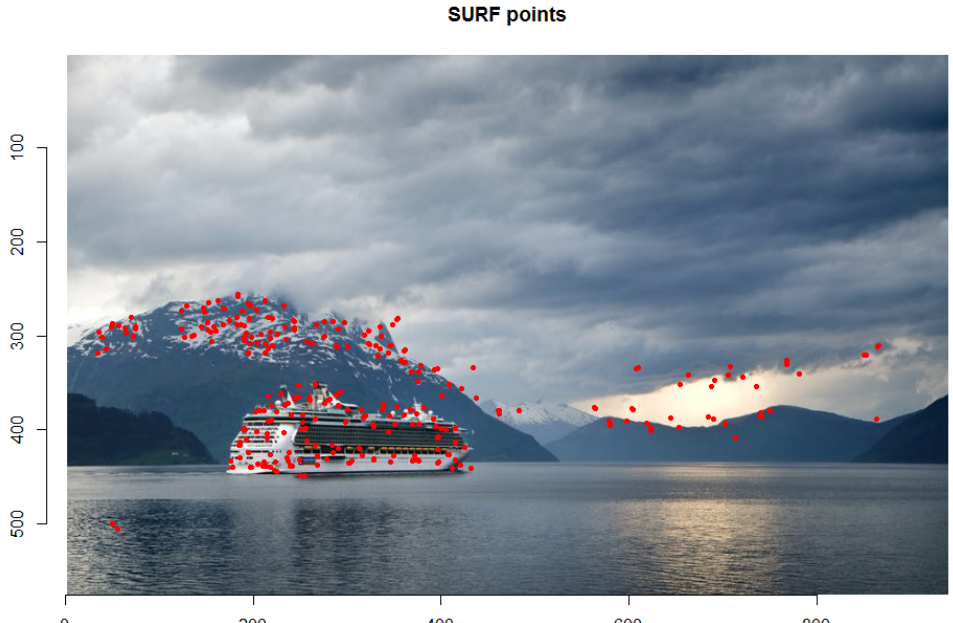
- Speeded up robust features (**SURF**)
  - Identifies points in images
  - Gives a **64-dimensional description of each of the points**
- SURF descriptors have been used to locate and recognize objects, people or faces, to reconstruct 3D scenes, to track objects and to extract points of interest.
- The SURF feature descriptor is based on the sum of the Haar wavelet response around the point of interest. Algorithm details at

  `http://www.ipol.im/pub/art/2015/69`
- Want to use it to do object matching/object recognition?
  - Run SURF descriptor (image.dlib::image_surf) on 2 images
  - Compare the SURF descriptors based on k-nearest-neighbours e.g. with the rflann R package (`https://CRAN.R-project.org/package=rflann`).

Figure: SURF for matching

Example below finds the blows. The output shows the points, and the 64-dimensional SURF descriptor for each of the points. If we plot it, we see it finds the boat in the below image and the mountain.

```
library(image.dlib)
f <- system.file("extdata", "cruise_boat.bmp", package="image.dlib")
surf_blobs <- image_surf(f, max_points = 10000, detection_threshold = 50)
str(surf_blobs)

List of 8
 $ points       : num 296
 $ x            : num [1:296] 232 237 282 374 186 ...
 $ y            : num [1:296] 402 371 367 382 416 ...
 $ angle        : num [1:296] -2.99 2.31 2.14 -1.43 1.42 ...
 $ pyramid_scale: num [1:296] 5.27 2.76 2.97 2.94 2.96 ...
 $ score        : num [1:296] 959 630 596 549 526 ...
 $ laplacian    : num [1:296] -1 -1 -1 -1 -1 -1 -1 -1 -1 1 ...
 $ surf         : num [1:296, 1:64] -0.0635 0.1435 0.1229 0.0496 -0.0501 ...

## Plot the points
library(imager)
library(magick)
img <- image_read(path = f)
plot(magick2cimg(img), main = "SURF points")
points(surf_blobs$x, surf_blobs$y, col = "red", pch = 20)
```



New R package (5): image.dlib - FHOG

- Histogram of oriented gradients (**HOG**) features
- On top of C++ library dlib (http://dlib.net) which works with bmp files as input
- HOG: http://dlib.net/imaging.html#extract_fhog_features - popular pedestrian detection algorithm commonly used in the automotive industry
  - input image is broken into cells that are (cell size) x (cell size) pixels
  - within each cell we compute a **31 dimensional FHOG vector**
  - this vector describes the gradient structure within the cell and which **can be used in traditional supervised learning**
  - finds features even in case of changes in illumination and viewpoint, but also due to non-rigid deformations, and intraclass variability in shape and other visual properties.
  - Object Detection with Discriminatively Trained Part Based Models (P. Felzenszwalb) http://people.cs.uchicago.edu/~pff/papers/lsvm-pami.pdf

```
library(image.dlib)
f <- system.file("extdata", "cruise_boat.bmp", package="image.dlib")
x <- image_fhog(f, cell_size = 8)
str(x)

List of 6
 $ hog_height         : num 70
 $ hog_width          : num 116
 $ fhog               : num [1:70, 1:116, 1:31] 0.4 0.4 0.311 0.275 0.399 ...
 $ hog_cell_size      : int 8
 $ filter_rows_padding: int 1
 $ filter_cols_padding: int 1
```

bnosac

open analytical helpers

Object detection

New R package (6): image.darknet

- Interface to darknet (`https://pjreddie.com/darknet/yolo`)
- Applications
  - Detect locations of objects in an image based on YOLO - You Only Look Once
  - Classify an image with existing deep learning models (AlexNet, Darknet, VGG-16, Extraction, Darknet19)
- Darknet: single neural network to the full image.
  - Network divides the image into regions and predicts bounding boxes and probabilities of objects inside each region.
  - These bounding boxes are weighted by the predicted probabilities

```
library(image.darknet)

## Define the model
yolo_tiny_voc <- image_darknet_model(
  type = 'detect',
  model = "tiny-yolo-voc.cfg",
  weights = system.file(package="image.darknet", "models", "tiny-yolo-voc.weights"),
  labels = c("aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
             "car", "cat", "chair", "cow", "diningtable", "dog",
             "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"))

## Find objects inside the image
image_darknet_detect(file = "img/clairvoyance.jpg", object = yolo_tiny_voc)
```

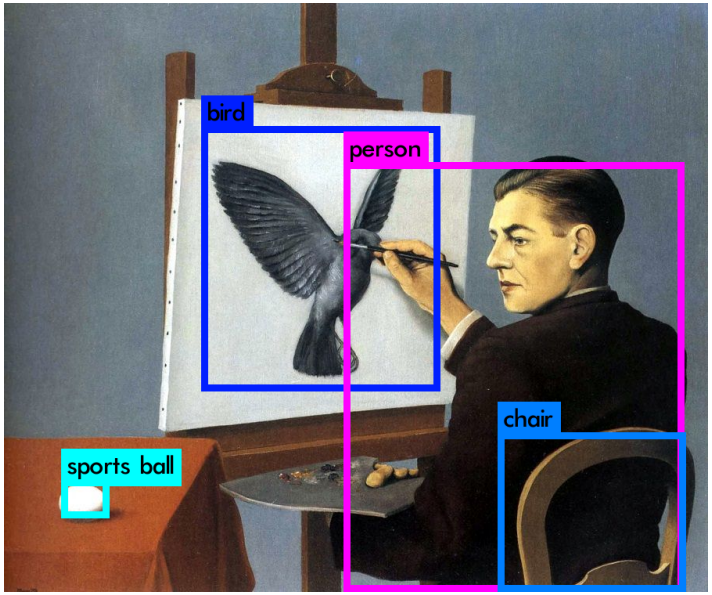


Figure: YOLO - Detection

- Example was based on YOLO. For other models (AlexNet, Darknet, VGG-16, Extraction, Darknet19), just download the deep learning weights and off you go. Examples at ?image_darknet_model

```
weights <- file.path(system.file(package="image.darknet", "models"), "yolo.weights")
download.file(url = "http://pjreddie.com/media/files/yolo.weights", destfile = weights)
```

- For classification of an image, use **image_darknet_classify**

```
library(image.darknet)
##
## Define model
##
model <- system.file(package="image.darknet", "include", "darknet", "cfg", "tiny.cfg")
weights <- system.file(package="image.darknet", "models", "tiny.weights")
labels <- system.file(package="image.darknet", "include", "darknet", "data", "imagenet.shortnames.list")
labels <- readLines(labels)

darknet_tiny <- image_darknet_model(type = 'classify',
                                    model = model, weights = weights, labels = labels)

##
## Classify new images alongside the model
##
f <- system.file("include", "darknet", "data", "dog.jpg", package="image.darknet")
x <- image_darknet_classify(file = f, object = darknet_tiny)
x

$file
[1] "C:/Users/Jan/Documents/R/win-library/3.3/image.darknet/include/darknet/data/dog.jpg"

$type
           label probability
1       malamute  0.18206641
2        dogsled  0.12255822
3     Eskimo dog  0.06975935
4         collie  0.04245654
5 Siberian husky  0.03541765
```

Figure: YOLO - for classification

open analytical helpers

Contact

www.bnosac.be

www.bnosac.be

Need more information, send a message at www.bnosac.be/index.php/contact/get-in-touch

Figure: Contact www.bnosac.be