Image processing and computer vision with R

eRum 2020

Lubomír Štěpánek[1, 2, 3] Jiří Novák[3]

[1]Institute of Biophysics and Informatics
First Faculty of Medicine
Charles University in Prague

[2]Department of Biomedical Informatics
Faculty of Biomedical Engineering
Czech Technical University in Prague

[3]Department of Probability and Statistics
Faculty of Informatics and Statistics
University of Economics, Prague

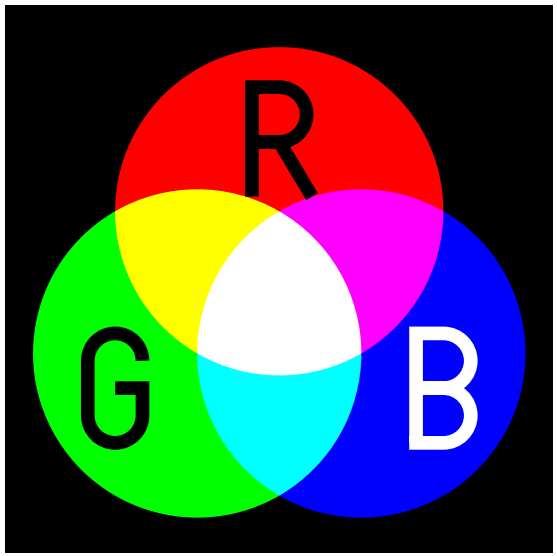
Content

Motivation

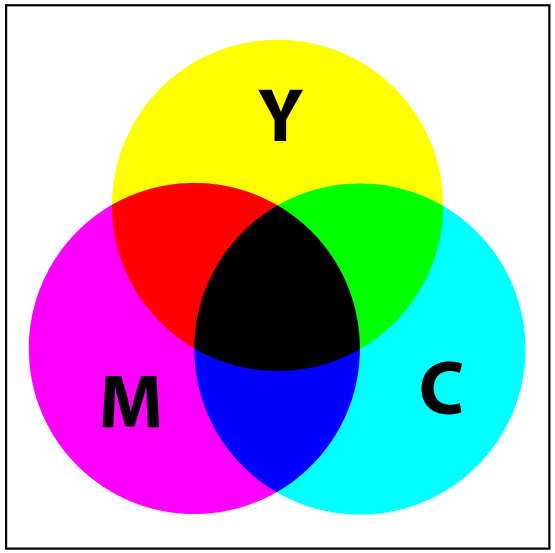
- image processing and computer vision task are top-class problems changing today's world
- options for effectively handling image processing tasks in R environment
  - although R is not considered as a number-one language for image processing
- R speaking community would like to combine many analyses by keeping all their code "under one roof" within R

Colour representation models

- RGB model
  - additive colour model
  - Red, Green, Blue colours



- CMYK model
  - subtractive colour model
  - Cyan, Magenta, Yellow, Key (black) colours



More on RGB colour model

- an image following the RGB colour model is represented as an array $m \times n \times 3$, where m is a number of pixels of image's height, n is a number of pixels of image's width
- three matrices below stand for red, green and blue components of each pixels of the image

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \quad \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \quad \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix}$$

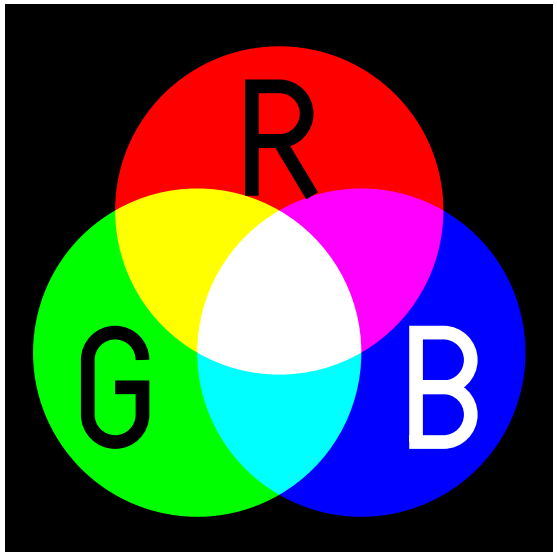
More on RGB colour model (con't)

- a pixel with coordinates $[i, j]$ is coded by a tuple of $(x_{i,j}, x_{i,j}, x_{i,j})^T$, such that $(x_{i,j}, x_{i,j}, x_{i,j})^T \in \{0, 1, 2, \ldots, 2^k - 1\}^3$, where k stands for bit depth of the image, i. e. $k = 8$ for 8-bit rasters, $k = 1$ for black-noir rasters, etc.
- sometimes, and typically in R (!), the intensities $(x_{i,j}, x_{i,j}, x_{i,j})^T$ are scaled, i. e.

$$(x_{i,j}, x_{i,j}, x_{i,j})^T \in \left\{ \frac{0}{2^k - 1}, \frac{1}{2^k - 1}, \frac{2}{2^k - 1}, \ldots, \frac{2^k - 1}{2^k - 1} \right\}^3 \in \langle 0, 1 \rangle^3$$

More on RGB colour model (con't con't)

- so that a pixel with coordinates $[i, j]$ equal to a tuple of
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (0, 0, 0)^T$ is black,
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (1, 0, 0)^T$ is (perfectly) red,
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (0, 1, 0)^T$ is (perfectly) green,
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (0, 0, 1)^T$ is (perfectly) blue,
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (1, 1, 1)^T$ is white,
  - $(x_{i,j}, x_{i,j}, x_{i,j})^T = (0.972, 0.905, 0.147)^T$ is somewhat yellow, etc.



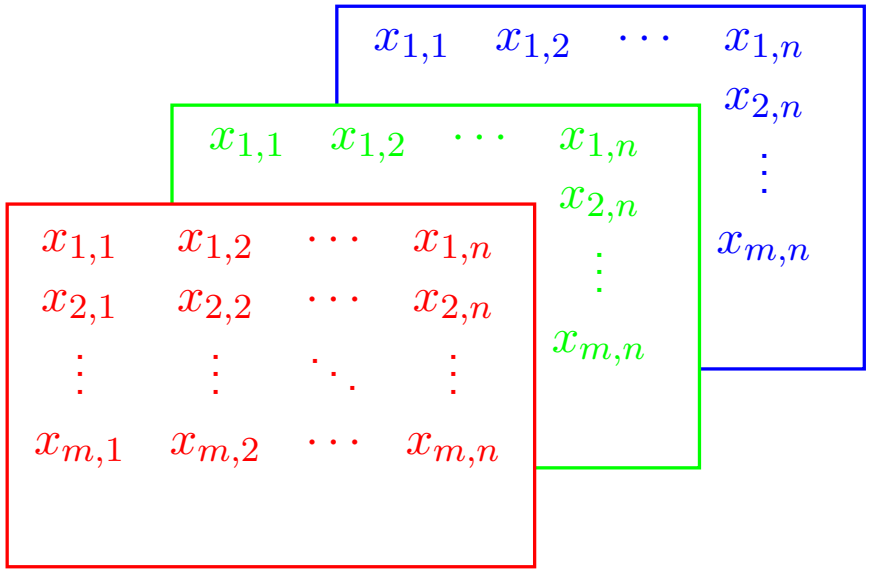


Image data formats

- vector formats
  - describe geometric elements that compose the image using an appropriate language
  - optimal for basic shapes, curves, functions, math plots, etc.
  - `.pdf`, `.eps`, `.svg`
- raster formats
  - describe complex pixels positions and colours
  - optimal for real world photographies
  - `.tiff`, `.jpeg`, `.png`, `.bmp`

List of R packages for image processing

- R packages handling images mostly using low-level code
  - `jpeg`, `png`, `bmp`, `grImport`
- R packages for general image processing tasks
  - `magick`, `imager`, `EBImage`, `OpenImageR`
- domain- and task-specific R packages
  - `CRImage` for tumor detection, `fftw` for Fourier transformation, `radiomics` for texture analysis, and many others
- API's, bridging and interfacing R packages
  - `ROpenCVLite` bridging the `openCV` python library, `dlib` bridging C++ `dlib` library, and others
- Bnosac's family of computer vision R packages
  - both for advanced image processing and computer vision tasks

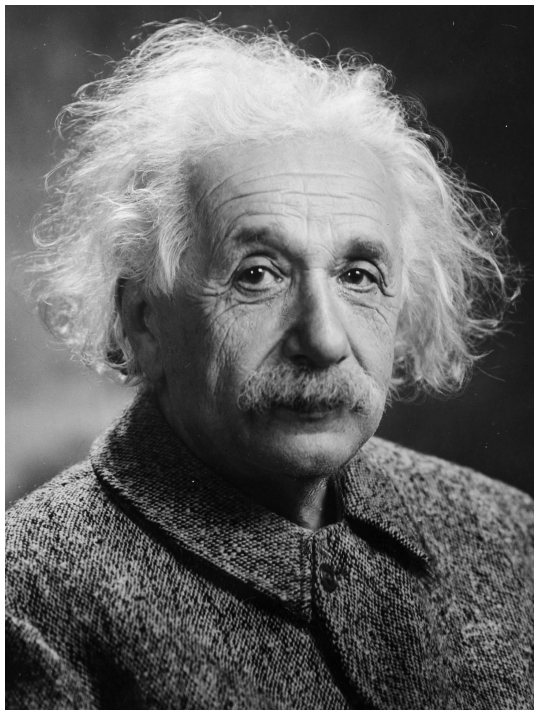
Reading and writing raster images

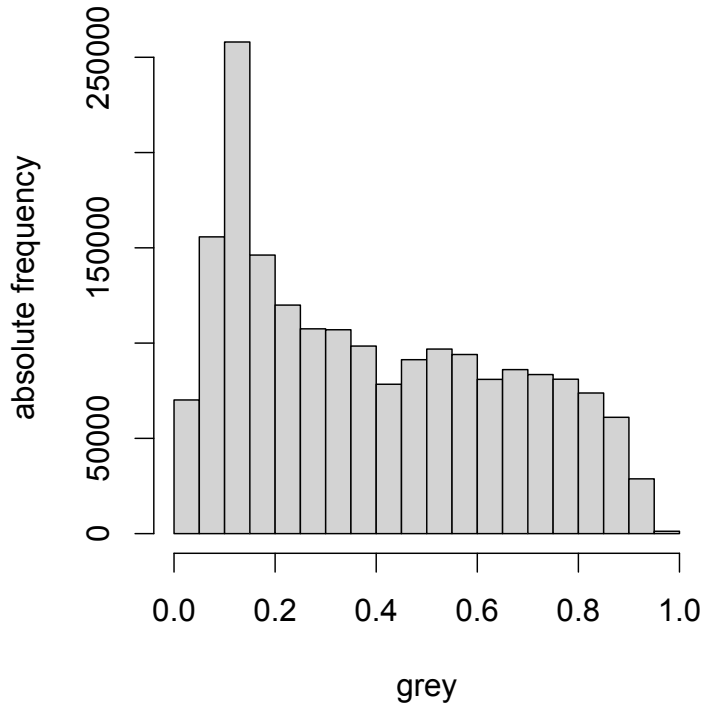
- R packages for image processing tasks use their own I/O functions
- considering the basic R packages `jpeg`, `png`, or `bmp`, it follows

```
1  library(jpeg)
2
3  my_image <- jpeg::readJPEG(
4    source = "my_image.jpg"
5  )
6
7  my_image[, , 1:2] <- 0 # all red and green pixels
8                         # are reduced to zero
9
10 jpeg::writeJPEG(
11   image = my_image,
12   target = "my_image_in_blue.jpg",
13   quality = 0.8
14 )
```

Histogram of pixel intensities

- histogram depicts counts of pixels of a given intensity
- could be plotted for grey pixels in greyscaled images, or for each colour channel (R, G, B, respectively) in coloured images





Contrast and brightness of an image

- brightness is an overall level of pixel intensities of an image
- contrast is a subjective difference between the darker and more intense pixels
- considering a grayscale image, for $\forall i \in \{1, 2, \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$ is $x_{i,j} \in \langle 0, 1 \rangle$ intensity of grey colour of a pixel with coordinates $[i, j]$
- adjusted intensity $x'_{i,j}$ of the pixel follows

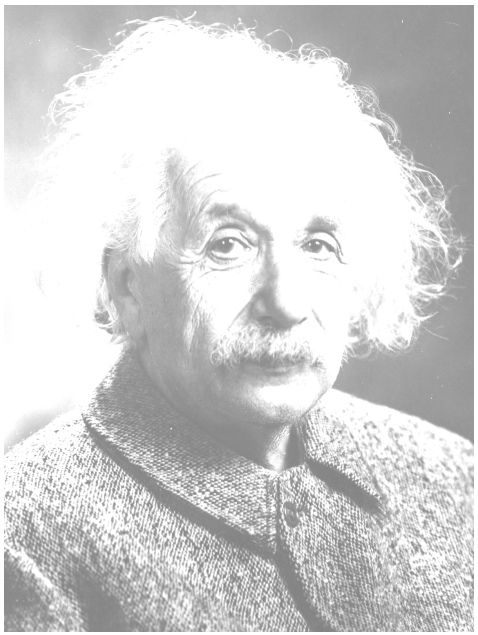
$$x'_{i,j} = \alpha \cdot (x_{i,j} - 0.5) + (0.5 + \beta),$$

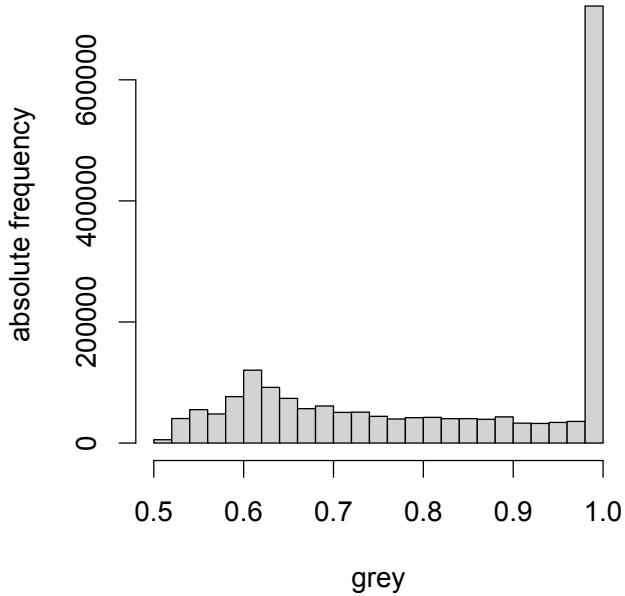
where α controls contrast (*gain*) and β controls brightness (*bias*)

  - $\alpha > 0$ means more contrast, $0 < \alpha < 1$ means less contrast
  - $\beta > 0$ means more brightness, $\beta < 0$ means less brightness

Increasing the brightness of an image with R

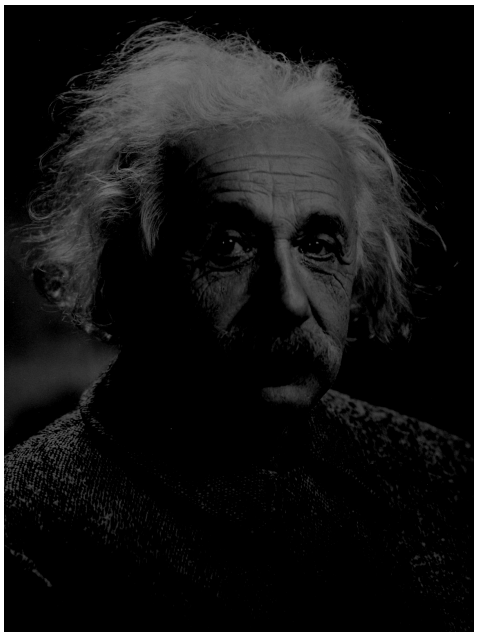
```
1 my_beta <- +0.5
2
3 ifelse(my_image + my_beta > 1,
4        1,
5        my_image + my_beta)
```

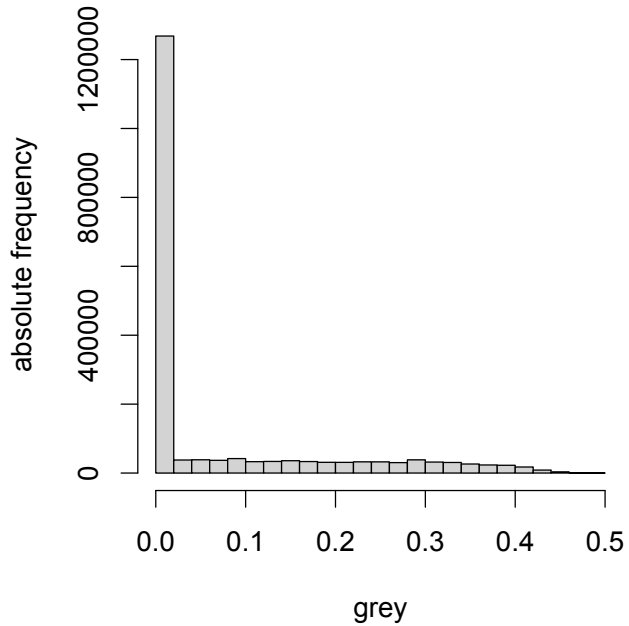




Decreasing the brightness of an image with R

```
1 my_beta <- -0.5
2
3 ifelse(my_image + my_beta < 0,
4        0,
5        my_image + my_beta)
```

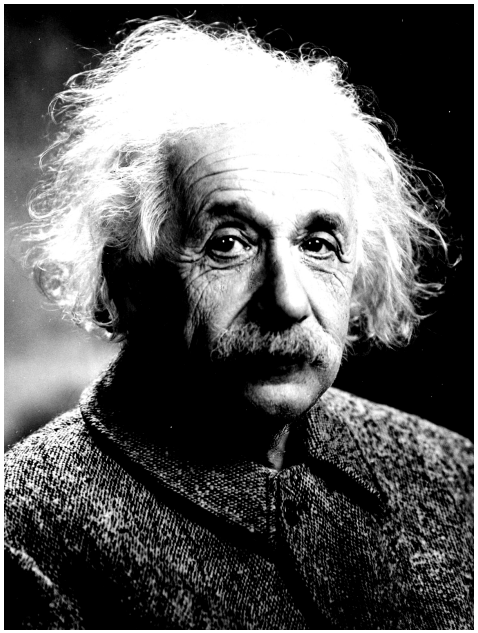


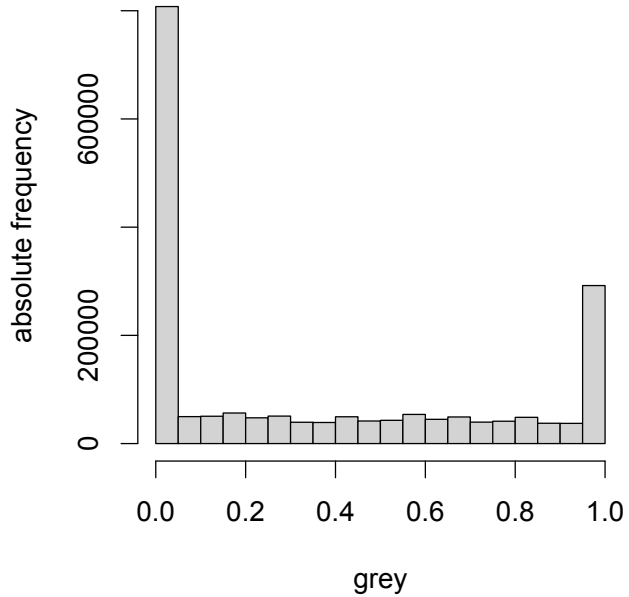


Increasing the contrast of an image with R

```
my_alpha <- 2.0

ifelse(my_alpha * (my_image - 0.5) + 0.5 > 1,
   1,
   ifelse(my_alpha * (my_image - 0.5) + 0.5 < 0,
          0,
          my_alpha * (my_image - 0.5) + 0.5))
```

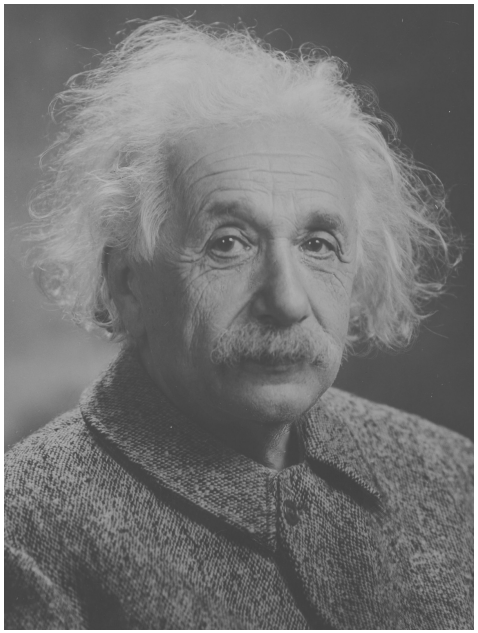


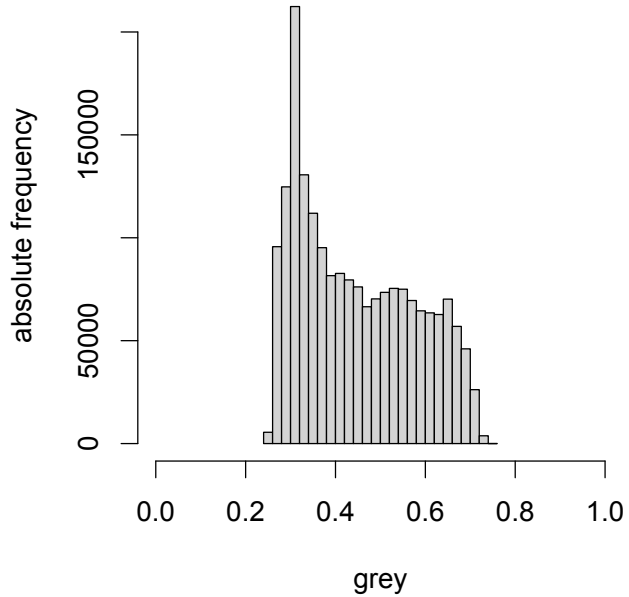


Decreasing the contrast of an image with R

```
my_alpha <- 0.5

ifelse(my_alpha * (my_image - 0.5) + 0.5 > 1,
   1,
   ifelse(my_alpha * (my_image - 0.5) + 0.5 < 0,
          0,
          my_alpha * (my_image - 0.5) + 0.5))
```





Hands-on!

(i) try to simplify the `ifelse()` condition for contrast decreasing

```
my_alpha <- 0.5

ifelse(my_alpha * (my_image - 0.5) + 0.5 > 1,
  1,
  ifelse(my_alpha * (my_image - 0.5) + 0.5 < 0,
         0,
         my_alpha * (my_image - 0.5) + 0.5))
```

(ii) try to convert the greyscaled image `einstein.jpg` into a black-and-white image
HINT: think about an appropriate threshold

Contrast and brightness enhancements

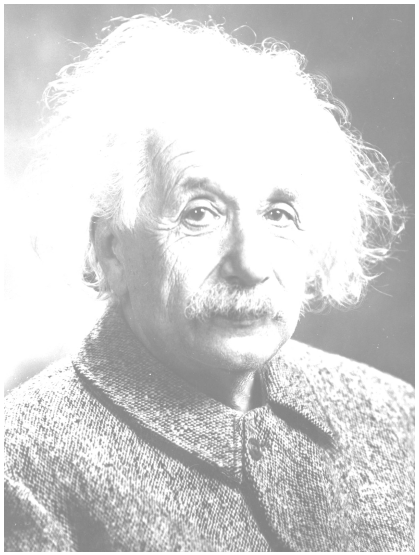
- brightness is an overall level of pixel intensities of an image
- contrast is a subjective difference between the darker and more intense pixels
- considering a image defined by an array of red, green, blue matrices (channels), for $\forall i \in \{1, 2, \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$ is $x_{i,j} \in \langle 0, 1 \rangle$ intensity of a colour of a pixel with coordinates $[i, j]$ given the matrix
- adjusted intensity $x'_{i,j}$ of the pixel follows

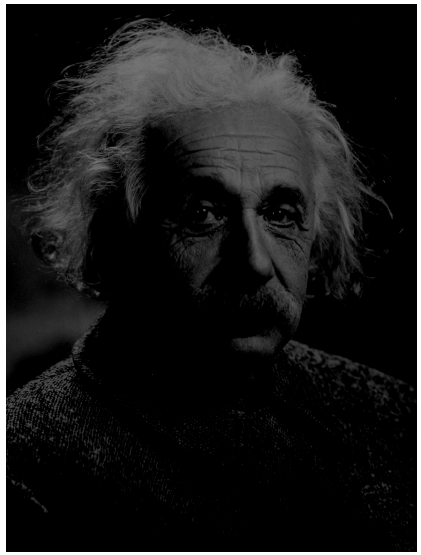
$$x'_{i,j} = \alpha \cdot (x_{i,j} - 0.5) + (0.5 + \beta),$$

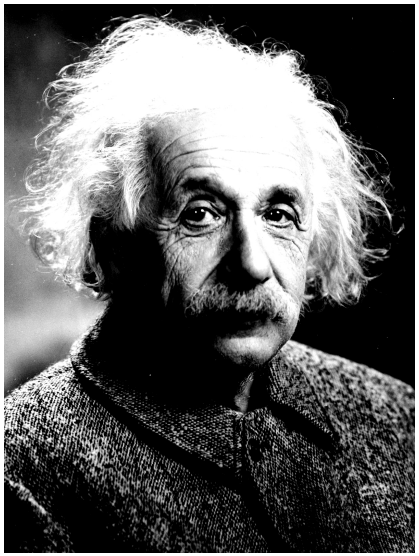
where α controls contrast (*gain*) and β controls brightness (*bias*)

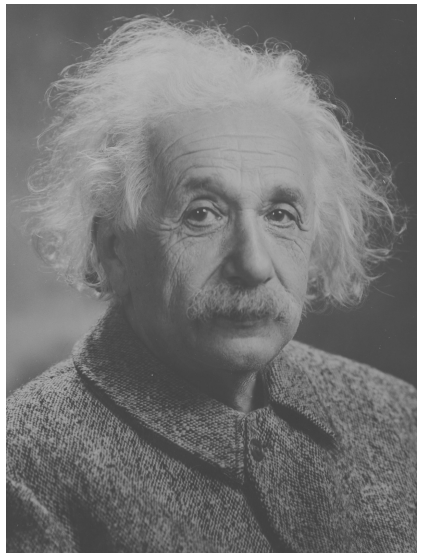
  - $\alpha > 0$ means more contrast, $0 < \alpha < 1$ means less contrast
  - $\beta > 0$ means more brightness, $\beta < 0$ means less brightness

Contrast and brightness enhancements (con't)









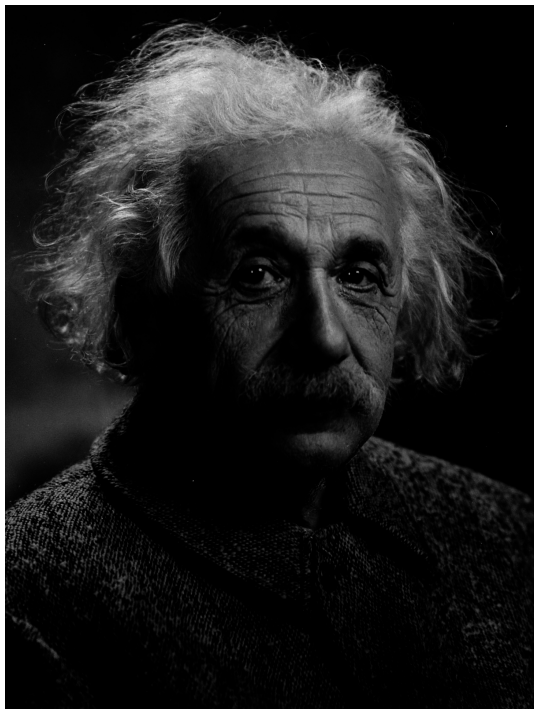
Gamma correction

- gamma correction is a non-linear used to adjust pixel intensities
- considering a image defined by an array of red, green, blue matrices (channels), for $\forall i \in \{1, 2, \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$ is $x_{i,j} \in \langle 0, 1 \rangle$ intensity of a colour of a pixel with coordinates $[i, j]$ given the matrix
- adjusted intensity $x'_{i,j}$ of the pixel follows

$$x'_{i,j} = x^{\gamma}_{i,j},$$

  where γ is an argument of gamma correction
- usually, $\gamma > 1$ make the shadows darker, while $\gamma < 1$ make dark regions lighter, keeping other pixel intensities almost intact

Gamma correction (con't)



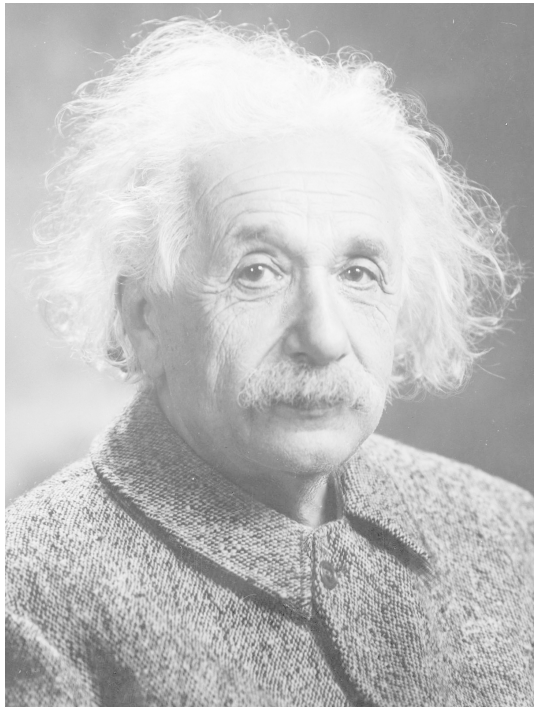
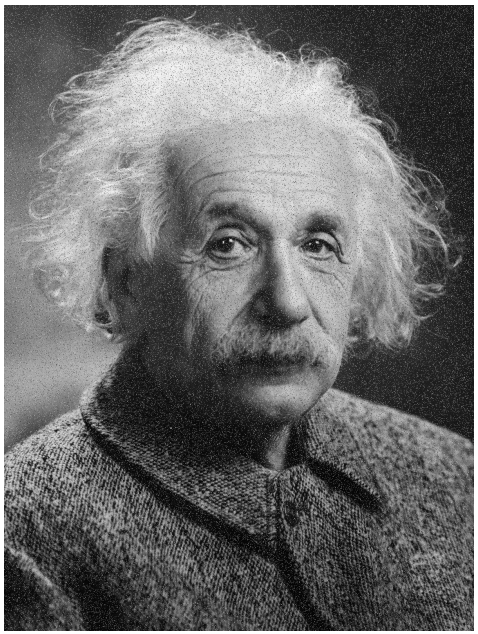


Image denoising

- there are many types of noise
  - salt-and-pepper (random), uniform, gaussian, etc.
- a noise can be reduced by denoising
  - linear filters, smoothing, statistical denoising, non-linear filters (median filter), etc.



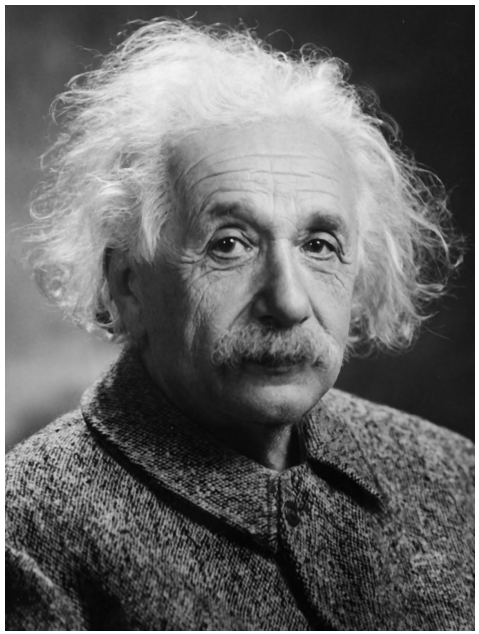
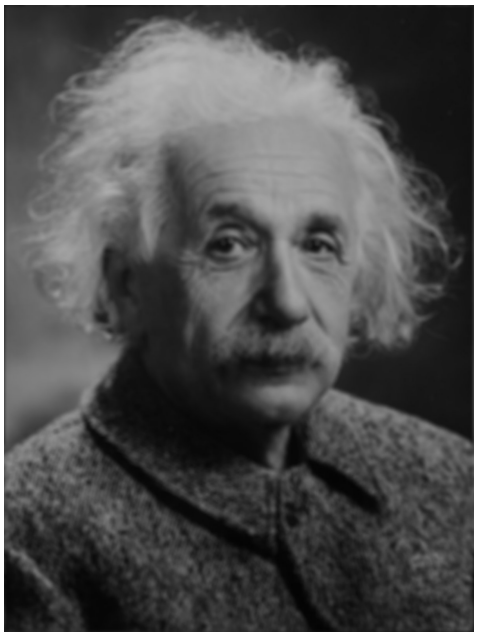


Image filtering

- low-pass filtering is used to perform blur image, remove noise, etc.
  - Gaussian blur (smoothing) as a convolution the image with a Gaussian function; it reduces image noise and details
- high-pass filtering is used to perform detect edges, sharpen images, etc.



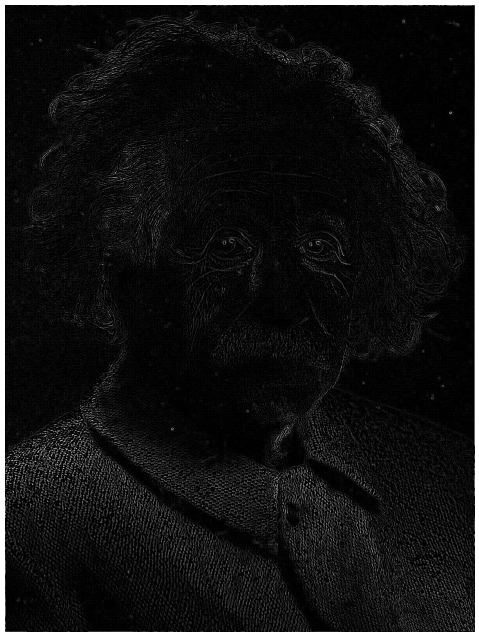


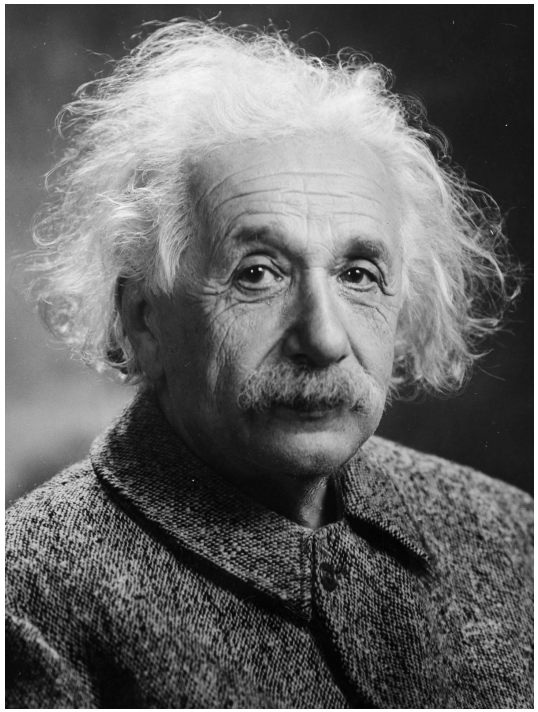
Image thresholding

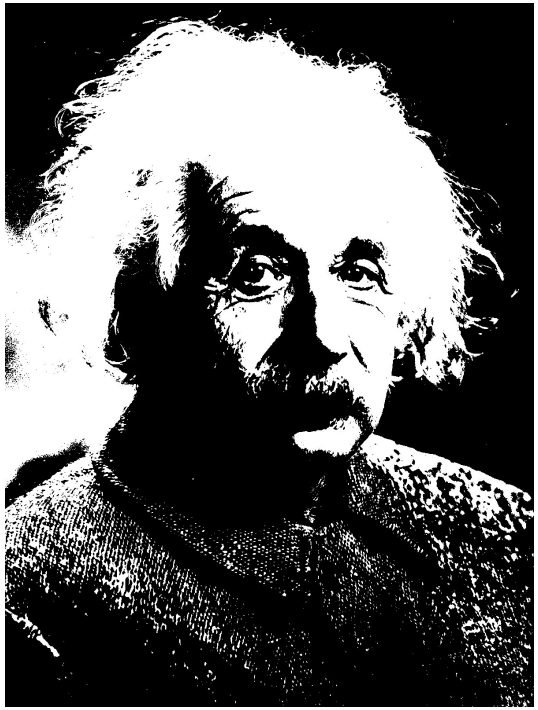
- a simple method for segmenting images
- it can create a binary (black-and-white) image from a grayscale one
- considering a grayscale image, for $\forall i \in \{1, 2, \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$ is $x_{i,j} \in \langle 0, 1 \rangle$ intensity of grey colour of a pixel with coordinates $[i, j]$
- adjusted intensity $x'_{i,j}$ of the pixel follows

$$x'_{i,j} = \begin{cases} 1, & x_{i,j} \geq t \\ 0, & x_{i,j} < t, \end{cases}$$

where $t \in \langle 0, 1 \rangle$ is a given threshold

Image thresholding (con't)





Blob detection and counting

- blob counting detection of regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions
- usually based on convolution
  - (i) differential methods based on derivatives of the function with respect to position
  - (ii) methods based on local extrema, finding the local maxima and minima of the function

Computer vision with R

- using Bnosac's family of packages for computer vision[1], these task are comfortable to be solved
  - face detection using `image.libfacedetection`
  - image recognition using `image.darknet`
  - image classification using `image.darknet`
  - significant points (landmarks) detection using `image.dlib`

[1] Jan Wijffels and BNOSAC. *image: Computer Vision and Image Recognition algorithms for R users*. R package version 0.1.0. 2017. URL: https://github.com/bnosac/image

Face detection

- using Bnosac's package `image.libfacedetection`
- it employs the weights of a Single Shot Detector (SSD) Convolutional Neural Network that was trained with the Caffe Deep Learning kit

Significant points (landmarks) detection

- using Bnosac's package `image.dlib` which bridges C++ library `dlib`[2] by `Rcpp` package
- it uses Speeded Up Robust Features (SURF) descriptors to locate and recognize objects, people or faces

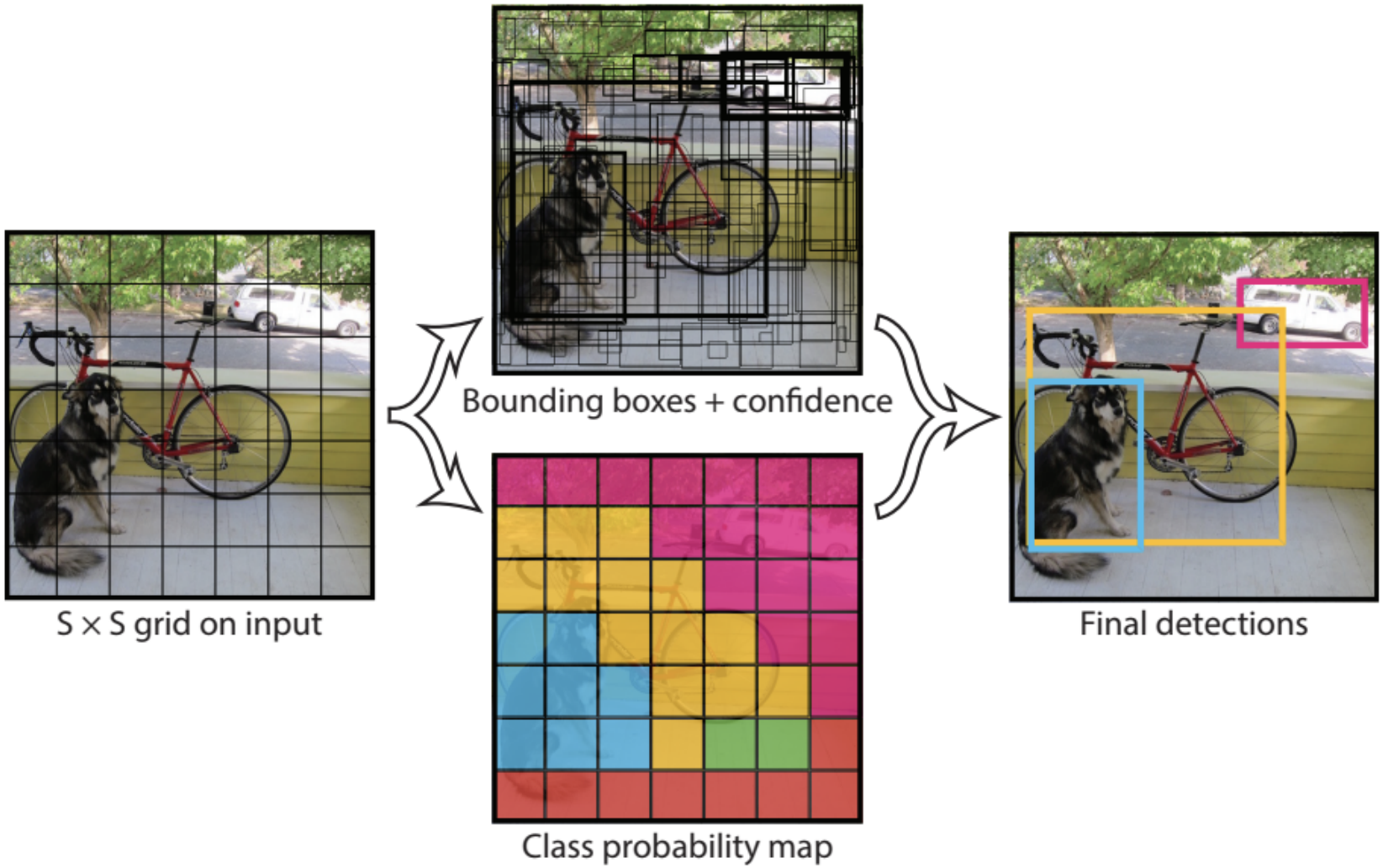
[2] Davis E. King. "Dlib-ml: A Machine Learning Toolkit". In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758

Image detection using YOLO approach

- YOLO (You Only Look Once) treats an object detection as a regression problem to spatially separated bounding boxes and associated class probabilities[3]
- predictions done directly from full images in one evaluation

[3] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV]

Image detection using YOLO approach (con't)



Thank you for your attention!

lubomir.stepanek@{vse, lf1.cuni, fbmi.cvut}.cz
jiri.novak@vse.cz