Image classification

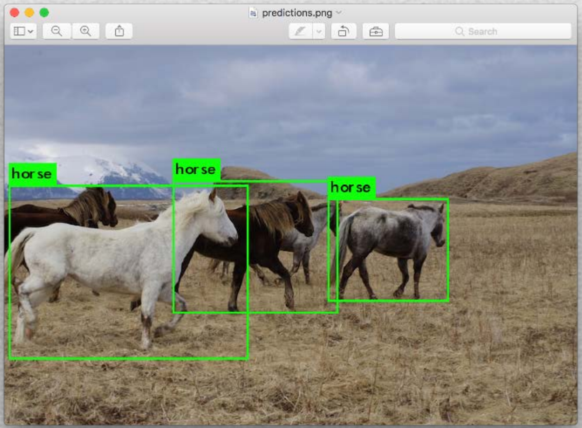
Raúl Alonso Calvo

Introduction

- A possible simple definition of image classification:
  - Assign a label to an image or to parts of an image from list of predefined labels (classes)

- The goal is allowing the computer to distinguish objects inside one image

- Easy task for humans
  - In every domain?



Source: https://pjreddie.com/darknet/yolo/

Introduction

- Well-known algorithms for data classification have to be adapted for this task
- Image manipulation usually requires different techniques than numerical or textual data
- Data in images is locally related

Introduction

- Classification could be binary or multiclass

- Some examples

  - Helping to diagnose bi-lateral pneumonia using x-ray

  - Classifying objects inside one image

  - Classifying images from handwritten characters or digits

Required steps

- Image Preprocessing
- Detection of ROI or objects
- Feature definition and extraction
- Training and Classification

Image pre-processing

- Usually, the great forgotten step
  - Simplifies the subsequent tasks
  - Problem-driven

- ‘Classical’ filters and techniques are very useful to simplify the problem

- Basic operations as color transformation

Basic commands

- We will see some examples with python using OpenCV
  - Most of these methods are available in several libraries

```
import cv2
```

- Opening an image

```
 # Load a color image in grayscale
img = cv2.imread('messi5.jpg’,flag)

#cv2.IMREAD_COLOR (1)
#cv2.IMREAD_GRAYSCALE (0)
#cv2.IMREAD_UNCHANGED (-1)
```

Basic commands

- Displaying an image

```
window_name = 'image'
# Displaying the image
cv2.imshow(window_name, img)
#waits for user to press any key  this is necessary for user interaction)
cv2.waitKey(0)
```

- Free resources

```
#closing all open windows
cv2.destroyAllWindows()
```

Basic commands

- Copy the image

```
copy = img.copy()
```

- Get image size

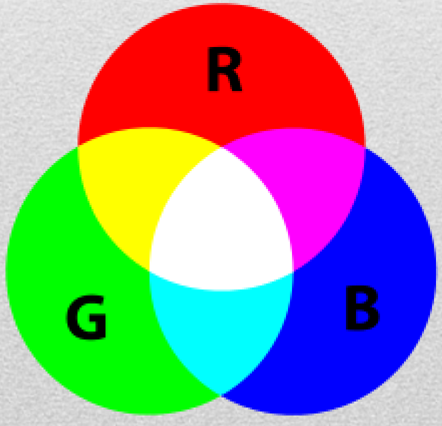
```
height, width, channels = img.shape
```

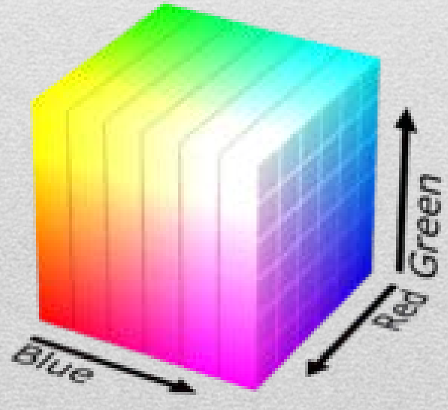
- Write an image

```
cv2.imwrite(filename, img)
# filename: must include the extension for format :  .jpg, .png, etc.
# img: It is the image that is to be saved.
# Returns true if image is saved successfully.
```

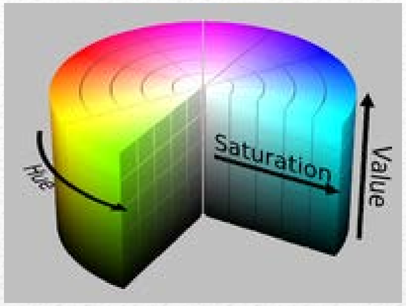
Color space

- RGB
  - RGB build the colord based in the combination of three light beams (one red, one green, and one blue)
  - They are superimposed on a black screen
  - Each of them can have an arbitrary intensity, from fully off to fully on, thus the RGB color model is *additive*





Color space



- HSL or HLS - hue, saturation, and lightness
- HSV or HSB - hue, saturation, and value/ brightness
- HIS - hue, saturation, and intensity
- Are the most common cylindrical-coordinate representations of points in an RGB color model
  - Hue indicates what color it is, usually 0-360 (a circle
  - Saturation tells how rich the color is (0-100%) (from white)
  - Value/Intensity indicates how bright the color is (0-100%)

Image pre-processing

- Colorspace conversion

```
 # Convert BGR to HSV
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
hsl = cv2.cvtColor(image, cv2.COLOR_BGR2HLS) # equal to HSL
luv = cv2.cvtColor(image, cv2.COLOR_BGR2LUV)
```

- Resize image

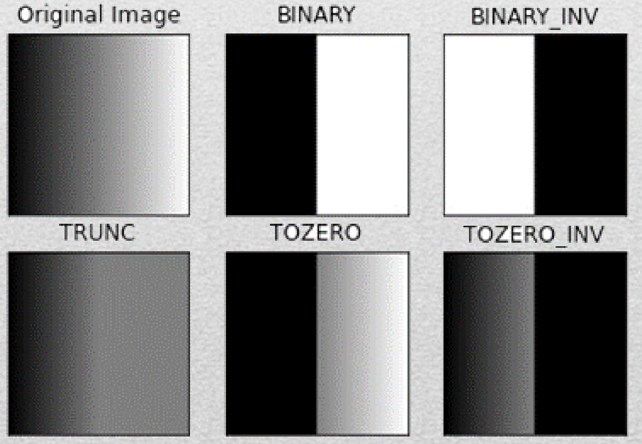
```
scale_percent = 60 # percent of original size
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)

# resize image
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
```

Image pre-processing

- Image thresholding

```
img = cv.imread('gradient.png',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
```



Source: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

Image pre-processing

- Canny edge detector (1986)
  - Based in finding directional (horizontal and vertica) intensity gradients
    - Uses convolution masks (matrices):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

  - It uses low and high thresholds:

```
edges = cv2.Canny(img,100,200)
```

- https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

Image pre-processing

- Histogram Equalization
  - To increase contrast in image (increase global contrast of an image using the image intensity histogram)

```
img_to_yuv = cv2.cvtColor(img,cv2.COLOR_BGR2YUV)
img_to_yuv[:,:,0] = cv2.equalizeHist(img_to_yuv[:,:,0])
hist_equalization_result = cv2.cvtColor(img_to_yuv,
        cv2.COLOR_YUV2BGR)
```

Image pre-processing

- Previously used methods
  - Erosion, dilation

```
element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3,3))
eroded = cv2.erode(img, element)
dilated = cv2.dilate(img, element)
```

  - Image substraction

```
res = cv2.subtract(img, eroded)
```

Practice 1

- Apply histogram equalization to Lena image (from test images) and display it

- Extract different channels (BGR and display before and after equalization)

  - You could try online using Jupyter Notebook. i.e.: https://jupyter.org/try

Required steps

- Image Preprocessing
- Detection of ROI or objects
- Feature definition and extraction
- Training and Classification

Detection of ROI or objects

- Depending on our problem we can decide to describe each image by:
  - Describe the whole image
  - Describe contents of the image (ROI)

- We have to define which solution is the best for our problem

Feature extraction

- We are going to see to typical features commonly used as image descriptors
- Color histogram
- Keypoints selection (SIFT, ORB, FAST, BRIEF, …)

Color histogram

- A histogram of the number of pixels of each

```
cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
```

Practice 2

- Create the Lena Histogram and paint it graphically

  https://docs.opencv.org/master/d1/db7/tutorial_py_histogram_begins.html