Week 3&4: OpenCV

Edge Computing

C. García garsanca@ucm.es

May 10, 2022

- "Mastering OpenCV 4 with Python: a practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7", Villán, Alberto Fernández
- "Hands-On GPU-Accelerated Computer Vision with OpenCV

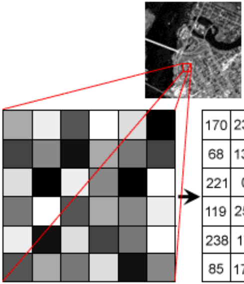
Outline

Images

- Computer vision is everywhere in our daily lives
  - Photos are ubiquitous in our social media feeds, news articles, magazines, books
  - Images are **everywhere**!!!
- Images from a combination of the original image's pixel values

Images as functions

- Each pixel has its own value
- For a gray-scale image, each pixel would have an intensity between 0 and 255
  - 0 means **black**
  - 255 means **white**



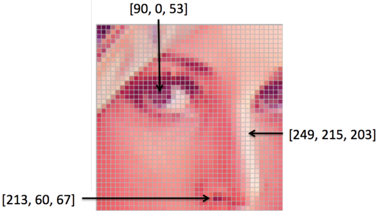
170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Images as functions

- Given a function $f(x, y)$, gives the intensity of the image at pixel position (x, y)
- Colors: combination of Red, Green, and Blue (RGB)
  - Image of width=256 and height=256
  - Total of $256 * 256 * 256 = 16,777,216$ combinations or color choices

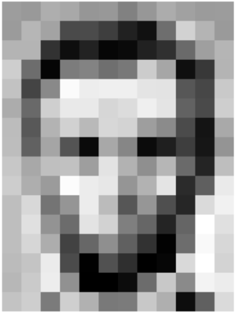
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix} \quad (1)$$

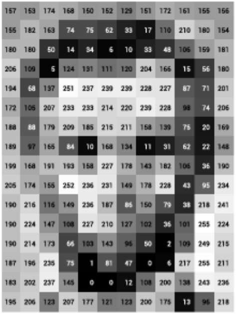
Images as functions



Images

- Images can be represented as a matrix of pixel values.





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Image Processing

- Two main types of image processing:
  - Image filtering & image warping

Image filtering

- Changes the range (i.e. the pixel values) of an image
- Image colors of the are altered **without** changing the **pixel positions**

Image Warping

- Changes the domain (i.e. the pixel positions) of an image
- Points are mapped to other points **without** changing the **colors**

Image Processing

Image Filtering

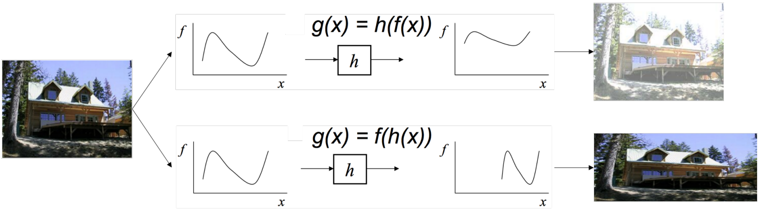


Image Warping

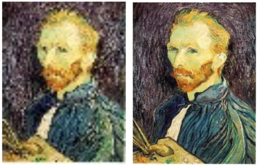
Image Filtering

- The goal of using filters is
  - To modify or enhance image properties
  - To extract valuable information from the pictures such as edges, corners, and blobs

De-noising

Salt and pepper noise

Super-resolution



In-painting

Image Filtering

- Example: image segmentation filter
  - Moving *average filter* replaces each pixel with the average pixel value of it and a neighborhood window of adjacent pixels
  - The effect is a more **smooth image** with sharp features removed

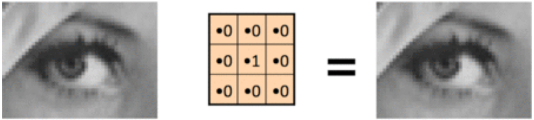
Convolution

- Many filters can be expressed in a principal manner using 2D convolution
  - Smoothing and sharpening images and detecting edges are based on **conv2D**
- Convolution in 2D operates on two images: input image and kernel (acts as a filter)

Convolution

- No change:

$$kernel(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad (2)$$



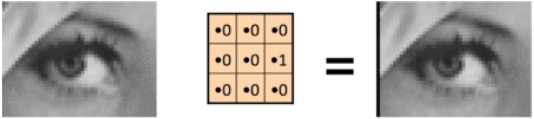
Original

Filtered
(no change)

Convolution

- Shifted right by one pixel:

$$kernel(x, y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (3)$$



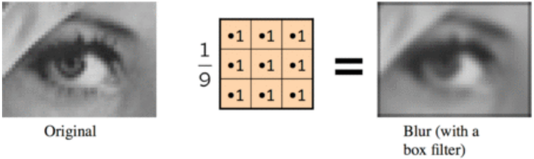
Original

Shifted right
By 1 pixel

Convolution

- Blurred:

$$kernel(x, y) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \quad (4)$$



Convolution

- Sharpening filter (two steps):

1. Step 1: Original - Smoothed = "Details" (see eq. 5)

$$kernelDetail(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5}$$

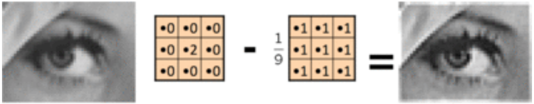
2. Step 2: Original + "Details" = Sharpened (see eq. 6)

$$kernelSharpened(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}\right) \tag{6}$$

- Sharpening filter (two steps)

$$kernelSharpened(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right)$$

$$= 2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Original

Edge Detection

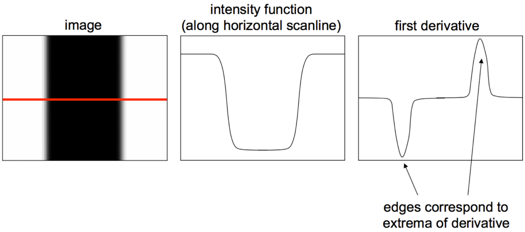
- Edges are sudden discontinuities in an image
  - Arise from surface normal, surface color, depth, illumination, or other discontinuities

Importance

1. Most semantic and shape information can be deduced from them
   - Allows perform object recognition and analyze perspectives and geometry of an image
2. A more compact representation than pixels

Edge Detection

- Edge?
- Rapid change in the intensity function indicates an edge



Edge Detection

- An image gradient
  - A generalization of the concept of derivative to more than one dimension
  - Points in the direction where intensity increases the most
- If gradient is $\nabla f = \left| \frac{\delta f}{\delta x}, \frac{\delta f}{\delta y} \right|$ the gradient direction would be $\Theta = \tan^{-1}\left(\frac{\delta f/\delta y}{\delta f/\delta x}\right)$
- Edge strength would be the gradient magnitude: $\|\nabla f\| = \sqrt{\frac{\delta f}{\delta x}^2 + \frac{\delta f}{\delta y}^2}$

Edge Detection

- Applying a filter (derivative of a Gaussian function), we can eliminate the image noise and effectively locate edges

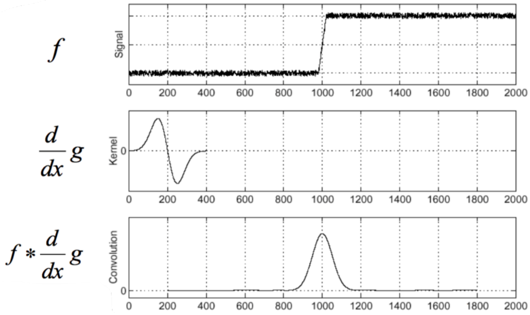
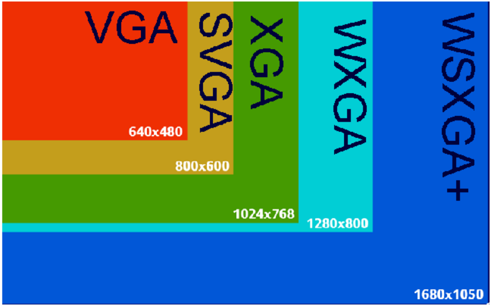


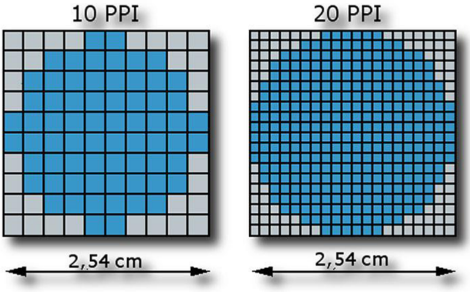
Image resolution

- Graphics display resolution is the **width** and **height** dimension of an electronic visual display device, measured in pixels



PPI

- Pixel per inch: concentration of pixels on a particular display
- For human eyes, 300PPI is enough for us to see comfortably

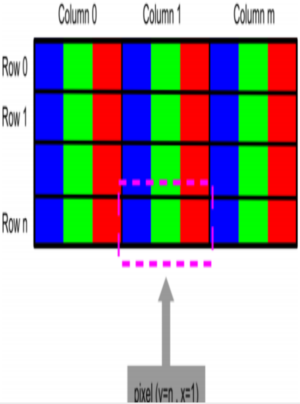


What is OpenCV

- **OpenCV** (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision
  - Supported in most of OS
  - APIs for C/C++, python and Java

BGR color format in OpenCV

- One pixel in OpenCV give us three values corresponding to the blue, green and red channels



Read images

- Read an image from file (using `imread`)
- Display an image in an OpenCV window (using `imshow`)
- Write an image to a file (using `imwrite`)

JupyterNotebook

- Open and follow the file **OpenCV/BasicOperations/Lecture_OpenCV1-Reading_Writing_DisplayingImages.ipynb**

Read/Write images

- Create a python script named **display_image.py**

display_image.py

```
import cv2 as cv
import sys
img = cv.imread('imgs/lena.png')
if img is None:
    sys.exit('Could not read the image.')
cv.imshow("Display window", img)
k = cv.waitKey(0)
if k == ord("s"):
    cv.imwrite('imgs/out_img.png', img)
```

Read/Write images

- Create a python script named **display_image.py**

Terminal #1

```
nano@jetson-nano:$ python3 display_image.py
```

Basic Functions

- **Grayscaling** is process by which an image is converted from a full color to shades of grey (black & white)

JupyterNotebook

- Open and follow the file **OpenCV/BasicOperations/Lecture_OpenCV2-Grayscaling.ipynb**

Basic Functions

- **Grayscaling** is process by which an image is converted from a full color to shades of grey (black & white)

gray_image.py

```
import cv2 as cv
import sys

img = cv.imread('imgs/lena.png')
if img is None:
   sys.exit('Could not read the image.')
gray_image = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow("Gray window", gray_image)
k = cv.waitKey(0)
```

Resizing and Croping

- Resizing means reducing the size of an image will require resampling of the pixels
  - Using the `resize` function
  - You need to interpolate new pixels

resize_image.py

```
import cv2 as cv
import sys

img = cv.imread('imgs/lena.png')
if img is None:
    sys.exit('Could not read the image.')

[w,h,c] = img.shape

# Resize to 100x200
img_resize = cv.resize(img, (100,200), interpolation= cv.INTER_LINEAR)

cv.imshow("Resize window", img_resize)
k = cv.waitKey(0)
```

Resizing and Croping

- Cropping is done to remove all unwanted objects or areas from an image
  - Or even to highlight a particular feature of an image
- There is no specific function for cropping using OpenCV,but as an imaged is stored in a 2D array...
  - Specify the height and width (in pixels) of the area to be cropped

crop_image.py

```
import cv2 as cv
import sys

img = cv.imread('imgs/lena.png')
if img is None:
   sys.exit('Could not read the image.')

[h,w,c] = img.shape

print('height='+str(h)+' width='+str(w))


# Cropping
img_cropped = img[80:280, 150:500]


cv.imshow("Cropped window", img_cropped)
k = cv.waitKey(0)

cv.imwrite('imgs/lena_cropped.png', img_cropped)
k = cv.waitKey(0)
```

Resizing and Croping

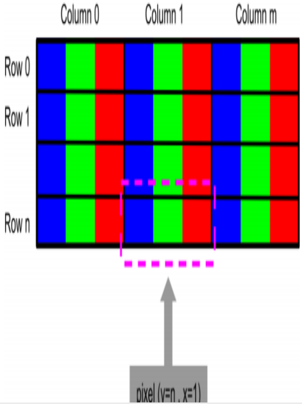
- Croping example

Terminal #1

```
nano@jetson-nano:$ python3 crop_image.py
```

BGR color format in OpenCV

- One pixel in OpenCV give us three values corresponding to the blue, green and red channels



Filtering

- **Remember the example**: image segmentation filter
  - Moving *average filter* replaces each pixel with the average pixel value of it and a neighborhood window of adjacent pixels
  - The effect is a more **smooth image** with sharp features removed

JupyterNotebook

Extra

- Interact with the CSI Camera

JupyterNotebook

- Open and follow the file **OpenCV/BasicOperations/Lecture_OpenCV5-ReadFromWebCam.ipynb**

Aritmetic with images

- Adding an offset: **image looks lighter**
- Subtract an offset: **image looks darker**

JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV6-ArithmeticOperations**

Bitwise operations with images

- Operations (AND, OR, XOR, NOT)
  - Useful while extracting any part of the image (as we will see in coming chapters)
  - Or working with non-rectangular ROI's, and etc

JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV7-BitwiseOperations.ipynb**

Masking with color

- Detect and extract colors present in an image using `inRange()`
  - Sometimes we want to remove or extract color from the image for some reason
- Detect a color using the range of that color (HSV triplet value format)

```
mask_orange = cv2.inRange(imgHSV, lower_orange,
upper_orange) #HSV format
```

JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV7-BitwiseOperations.ipynb**

Morfological operations (erosion, dilation, opening...)

- Morphological operations are a set of operations that process images based on shapes
- **Erosion**:
  - It is useful for removing small white noises.
  - Used to detach two connected objects etc.
- **Dilation**:
  - In cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases.

Erode and Dilate

crop_image.py

```
....
# Taking a matrix of size 5 as the kernel
kernel = np.ones((5,5), np.uint8)

# The first parameter is the original image,
# kernel is the matrix with which image is
# convolved and third parameter is the number
# of iterations, which will determine how much
# you want to erode/dilate a given image.
img_erosion = cv.erode(img, kernel, iterations=1)
img_dilation = cv.dilate(img, kernel, iterations=1)
...
```

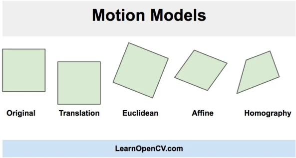
Perspective transformation

- Image Alignment: two images of the same scene are not aligned

C. García garsanca@ucm.es

Image Rotation and Translation

- **Translation:** image can be shifted (translated) by (x, y) to obtain the second image. There are only two parameters x and y that we need to estimate.
- **Rotation:** image is a rotated respect an angle
- More info related to Affine transformation with OpenCV



JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV8-Dilation_Erosion.ipynb**

Edge

- Identify the boundaries (edges) of objects
  - Sobel Edge Detection: one of the most widely used algorithms for edge detection (x & y direction)

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

- Gradient

$$G_x = A * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Thresholding

- The simplest segmentation method
  - Separate out regions of an image corresponding based on the variation of intensity between the object pixels and the background pixels
  - Differentiate the pixels from the rest (which will eventually be rejected)
  - i.e. we can assign them a value of 0 (black), 255 (white)

Thresholding

- Binary: $ifsrc(x, y) > thres, dest(x, y) = maxValue; elsedest(x, y) = minValue$
- Truncate
  $ifsrc(x, y) > thres, dest(x, y) = thres; elsedest(x, y) = src(x, y)$

```
dst = cv.threshold(src_gray, threshold_value, max_binary_va
```

Thresholding

- Simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity $I_{i,j}$ is less than a fixed value called the threshold T

Thresholding & Edge

- Edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images
  - Sobel uses two 3×3 kernels which are convolved
  - Canny edge detector uses a multi-stage algorithm: Gaussian Filtering, Gradient intensity, Gradient magnitude thresholding...

JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV9-Edge.ipynb**

Contours

- Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity
  - Come handy in shape analysis, finding the size of the object of interest, and object detection

JupyterNotebook

- Open and follow the file **OpenCV/ImageProcessing/Lecture_OpenCV10-Contours.ipynb**

Line Detection

- Hough transform is a feature extraction technique within a certain class of shapes by a voting procedure
  - After using edge detection (ie: Sobel, Canny operator)

Line Detection

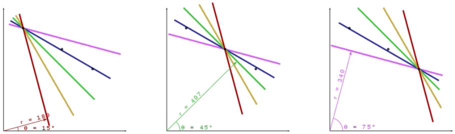
- Hough transform
  - Useful in many line detection

Line

- A line can be seen as: $y = m * x + n$
  - Where (x, y) means a single pixel in a image: **vertical lines are not easy**: $m = \infty$
- Polar representation can be also used: $y = -\frac{\cos\theta}{\sin\theta} * x + \frac{\rho}{\sin\theta}$
- Finally $\rho = x * \cos\theta + y * \sin\theta$ which means that every line can be associated to (ρ, θ) pair

Line Detection

- Consider three data points, shown here as black dots
  - For each data point, a number of lines are plotted
  - The Hough transform accumulates contributions from all pixels in the detected edge
  - The **brightest** accumulated points correspond to the lines



θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Line Detection

JupyterNotebook

- Open and follow the file **OpenCV/BasicOperations/Lecture_OpenCV11-Hough.ipynb**