OpenCV

Introduction to OpenCV

- OpenCV is a programming library with real-time computer vision capabilities
- Officially it was launched in 1999, OpenCV from an intel initiative
- It is written in C++
- First major release 1.0 was in 2006, second in 2009 and third in 2015
- It has many algorithms provided for processing
- It is supported in many languages like Python, C, C++, Java etc
- It is a free and open source library

OpenCV applications

- 2D and 3D feature toolkits
- Street view image stitching
- Egomotion estimation
- Facial-recognition system
- Gesture recognition
- Human-computer interaction
- Mobile robotics
- Motion understanding
- Object identification
- Automated inspection and surveillance
- Segmentation and recognition
- Stereopsis stereo vision
- Medical image analysis
- Structure from motion
- Motion tracking
- Augmented reality
- Video/image search and retrieval
- Robot and driverless car navigation and control
- Driver drowsiness and distraction detection

OpenCV modules

- OpenCV is divided into modules to provide image processing capabilities
- **Core**
  - Core functionality is a module defining basic data structures and also basic functions used by all other modules in the library
- **Imgproc**
  - An image-processing module that includes image filtering, geometrical image transformations, color space conversion, and histograms
- **Imgcodecs**
  - Image codecs. Image file reading and writing
- **Videoio**
  - Interface to video capturing and video codecs

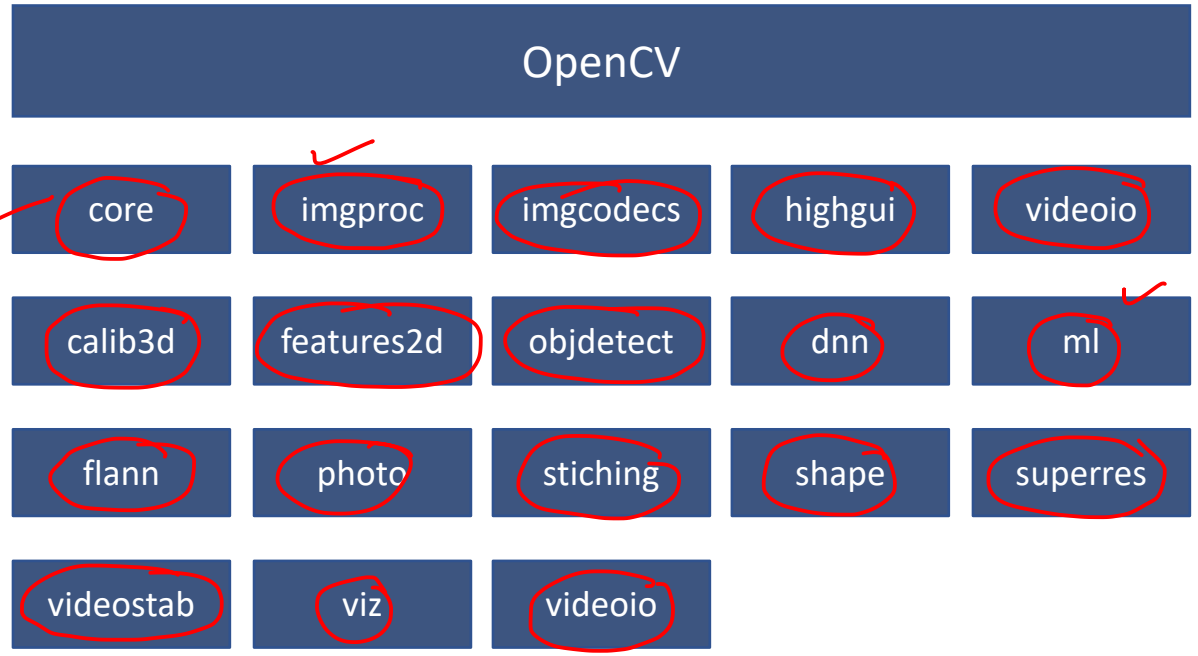


Image Fundamentals

- An image is an artifact that depicts visual perception, such as a photograph or other two-dimensional picture, that resembles a subject—usually a physical object—and thus provides a depiction of it
- It can be seen as a two-dimensional (2D) view of a 3D world
- A digital image is a numeric representation of a 2D image
- It is a finite set of digital values, which are called pixels
- The goal of OpenCV is to transform the 2D data into
  - A new representation (for example, a new image)
  - A decision (for example, perform a concrete task)
  - A new result (for example, correct classification of the image)
  - Some useful information extraction (for example, object detection)

pixel = pix + el

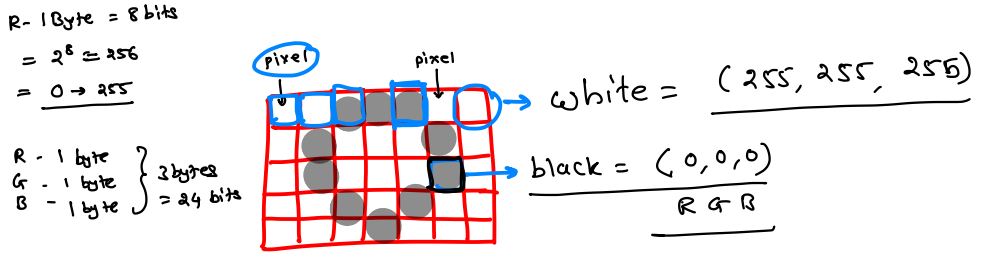
= picture + element

smallest unit of image

pixel information

Color Space

- There are several different color models (also known as color spaces)
- It is used to explain how the image looks like
- **RGB Color Space**
  - The most common one is RGB model which has three basic colors Red, Green and Blue
  - These colors are mixed together to produce broad range of colors
  - Each color (R, G and B) is usually called as a channel, which is commonly represented as an integer value in the range of 0-255
  - Which means each channel produces 256 levels
  - Since there are 3 channels, this is called as 24-bit color depth



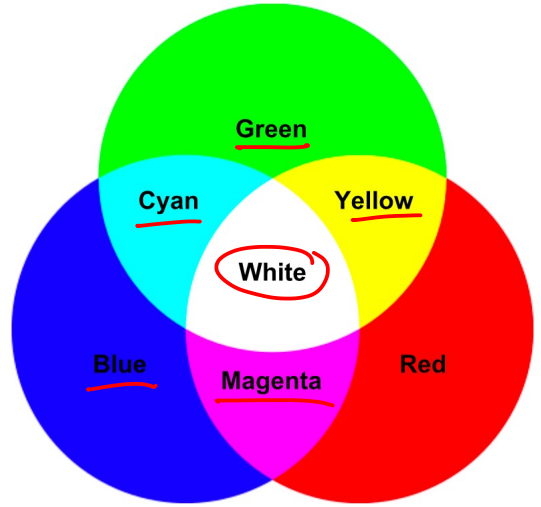


Image file formats

- Image file formats are standardized means of organizing and storing digital images
- An image file format may store data in an uncompressed format, a compressed format (which may be lossless or lossy), or a vector format
- E.g.
  - Bitmap image file (BMP)
  - Device independent bitmap (DIB)
  - Joint Photographic Experts Group (JPEG) → lossy
  - JPEG 2000
  - Graphics Interchange Format (GIF)
  - Portable Network Graphics (PNG)
  - Portable pixmap format (PPM)
  - Portable bitmap format (PBM
  - Portable graymap format (PGM)
  - Tagged Image File Format (TIFF) → fonts

How image is stored on computer?

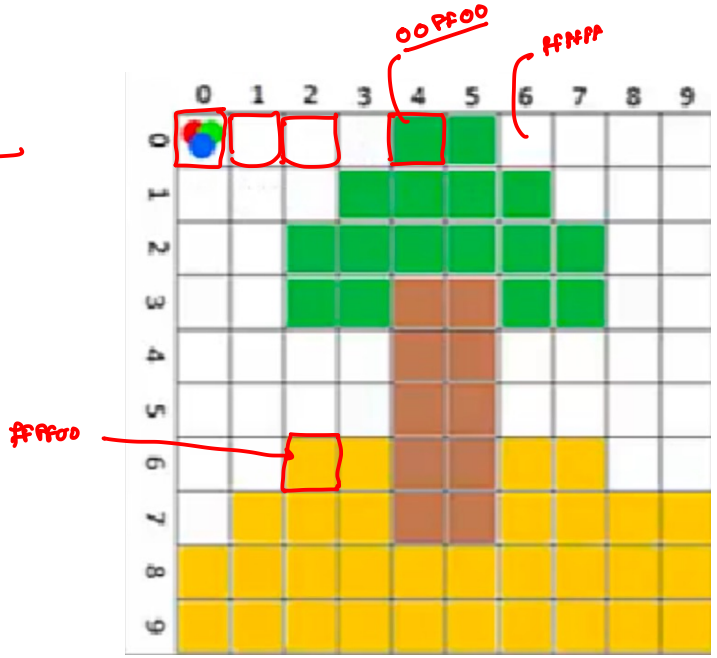
- Every image is stored as binary data (pixel)
- OpenCV uses RGB color space by default → BGR
- Each pixel coordinate (x, y) contains 3 values ranging for intensities of 0 to 255 (8bit)
  - Red
  - Green
  - Blue
- Mixing different intensities of each color gives us the full color spectrum
  - Yellow
    - Red - 255
    - Green - 255
    - Blue - 0

1 byte

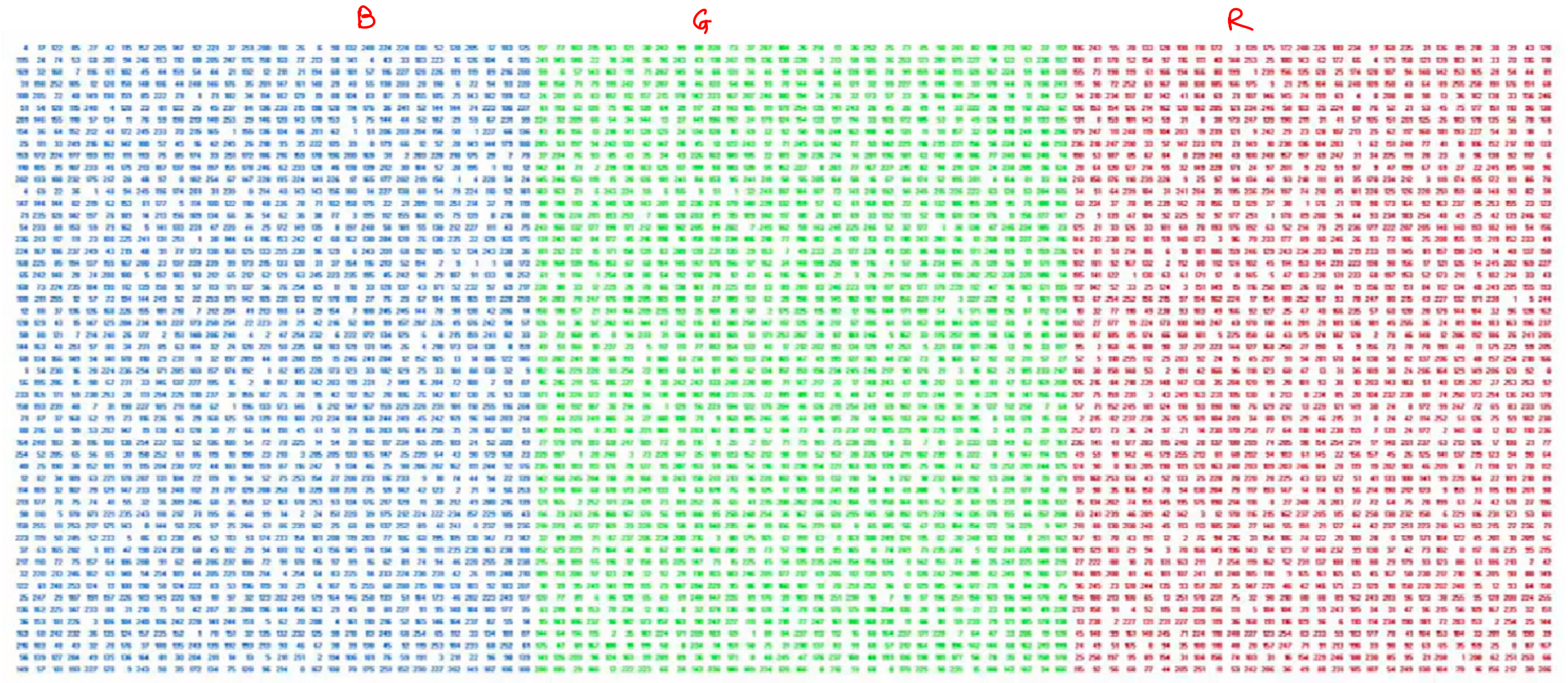
0 - 255
00 - FF

00 → 0
FF → 255

(255, 255, 255)
(FF, FF, FF)



How image is stored on computer?

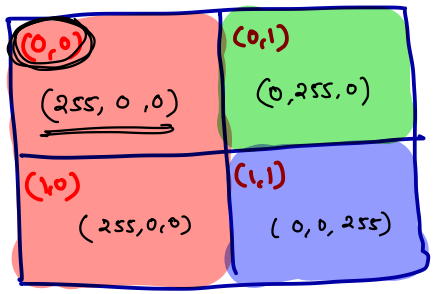


How image is stored on computer?

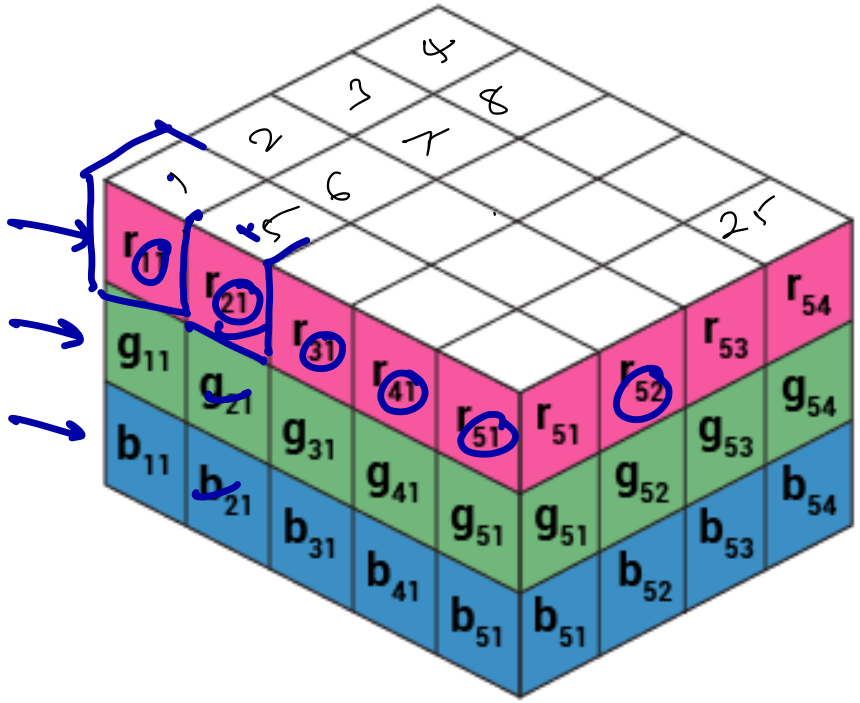
- Image is stored in multi-dimensional arrays

One dimensional array

Two dimensional array

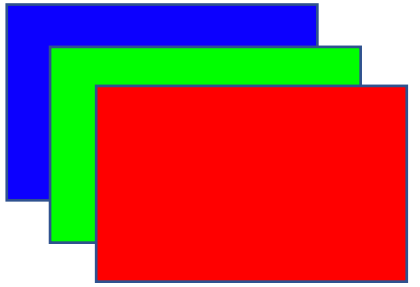


Three dimensional array

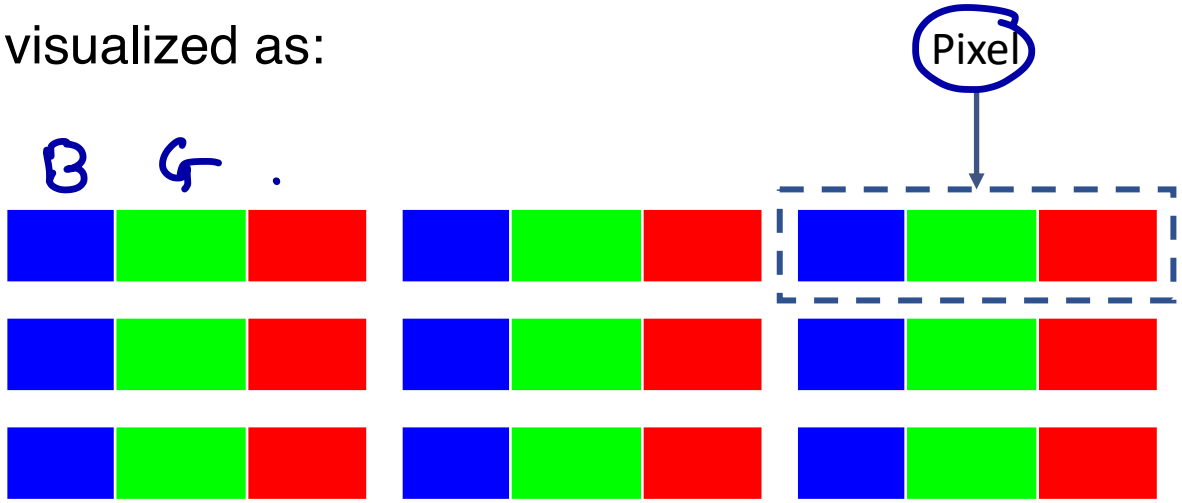


OpenCV Image Representation

- OpenCV uses BGR model instead of RGB model
- The basic colors remain same but they are read in different order



- The pixel structure can be visualized as:



Resolution = # pixels horizontally × # pixel vertically

= width × height

SD

↳ 800 × 640

HD

→ full = 1920 × 1080 = ?

→ half = 1280 × 720 = ?

4K

UHD = 3840 × 2160 = ?

Demo

Coordinate System in OpenCV

- Image is collection of 2D non-zero binary data (pixels)
- The left top corner starts (0, 0)

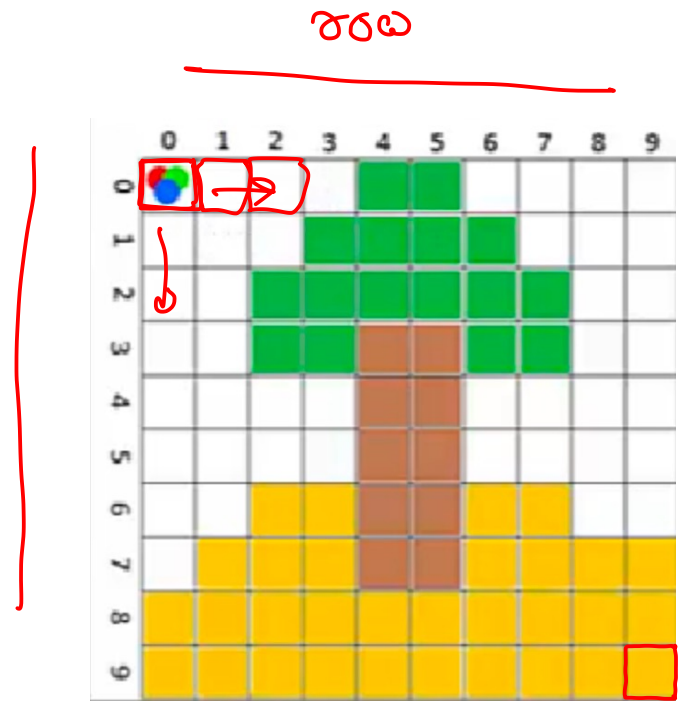


Image Processing steps

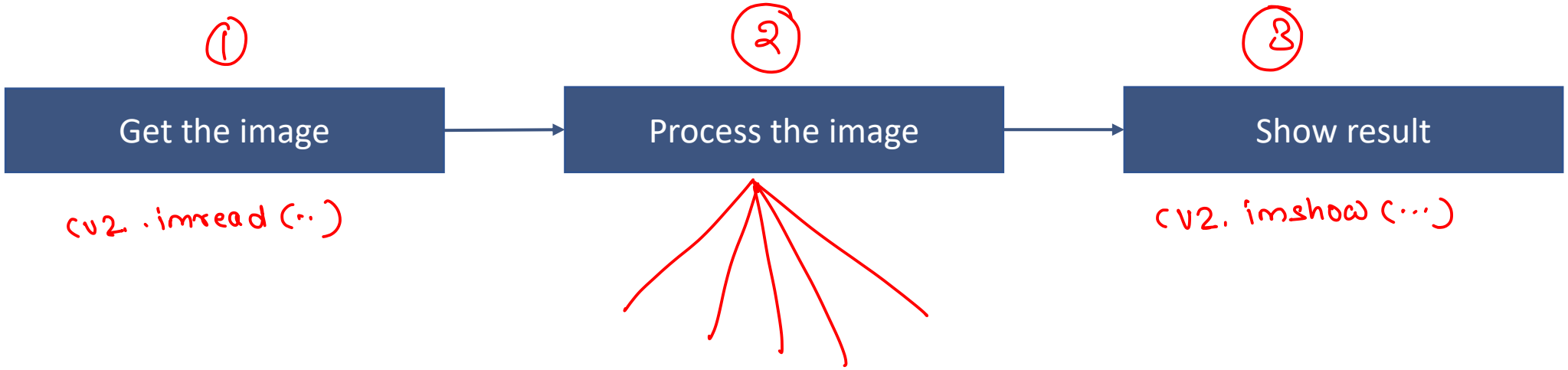


Image Processing Levels

pixel level processing

Low Level Processing

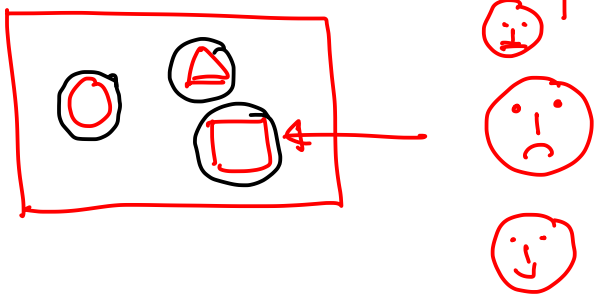
- Noise removal
- Image sharpening
- Illumination normalization
- Perspective correction

Mid Level Processing

- Getting rectangle information
- Getting shapes information

Object detection

Contour detection

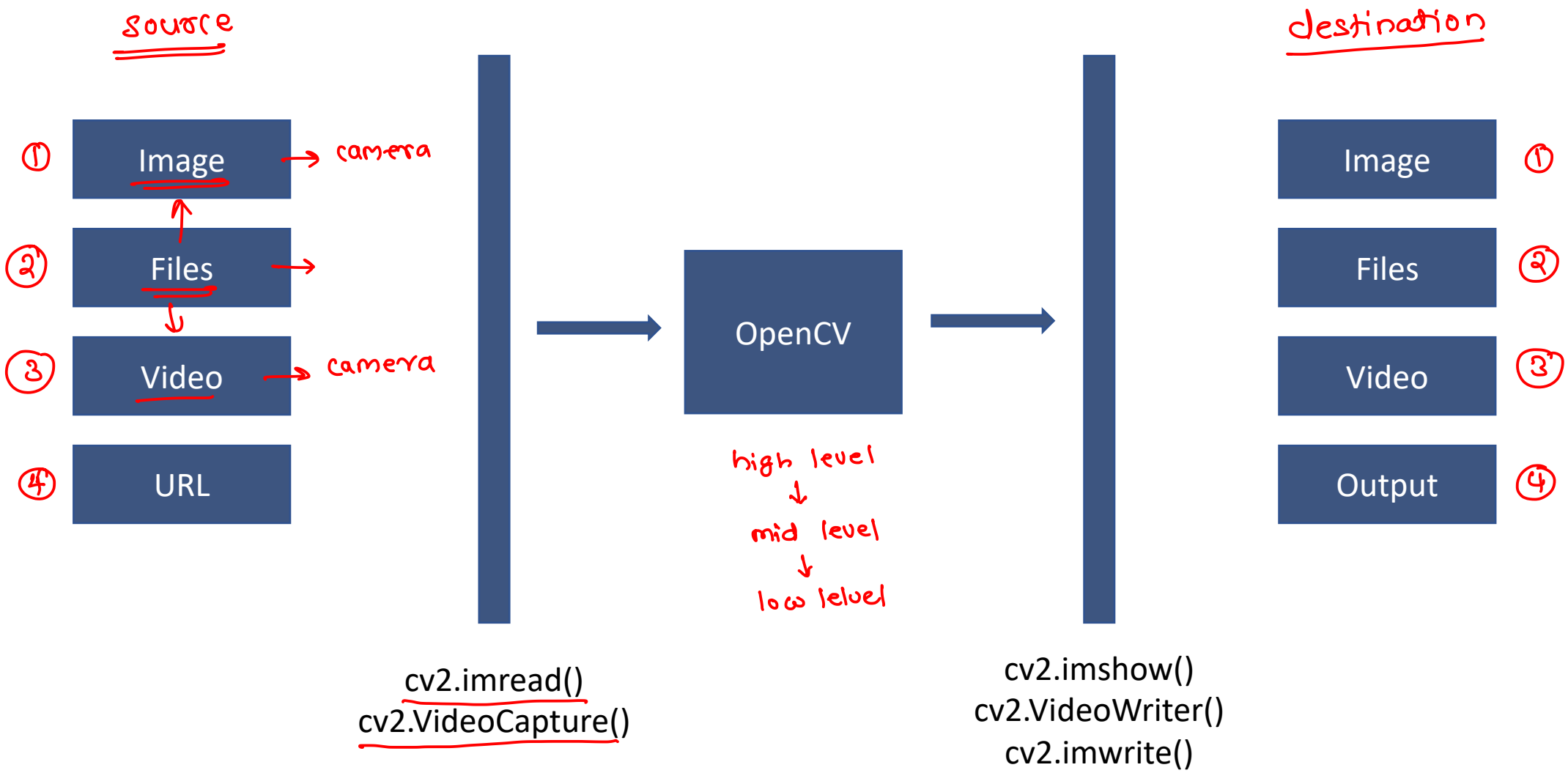


Uses Machine Learning

High Level Processing

- Face recognition
- Emotion recognition
- Drowsiness and distraction detection
- Remote heart rate measurement from the face

Image/File handling in OpenCV



Reading and Writing files

```
import cv2

# read the image
image = cv2.imread("logo.png")

# show the image and wait for user's key input
cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# write image to disk
cv2.imwrite("/tmp/newfile.png", image)
```

Reading Video

```
import cv2

capture = cv2.VideoCapture(0)

while capture.isOpened():
    ret, frame = capture.read()
    cv2.imshow('output', frame)

    if cv2.waitKey(20) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```

camera

take a frame (image) from the source

wait till user presses 'q'

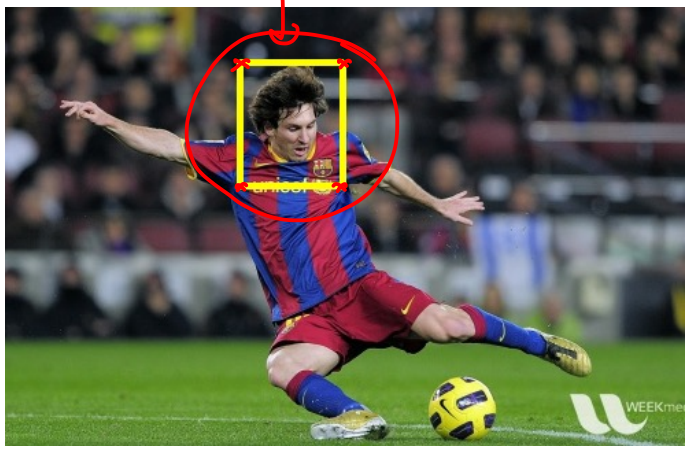
check the keypress

close the source

Drawing Shapes

Introduction to shapes

- OpenCV provides many functions to draw basic shapes
- Common basic shapes include
  - Lines
  - Rectangles
  - Circles
  - Texts
- It is useful in the scenario where the result needs to be highlighted



Creating empty image

- Image is a collection of 2D binary data (pixel)
- To create an empty image, just create an empty array
  - image = np.zeros((400, 400, 3), dtype=np.uint8)

initialized with zeros
black

Terminology

- **img**
  - It is the image where the shape will be drawn.
- **color**
  - It is the color (BGR triplet) used to draw the shape.
- **thickness**
  - If this value is positive, it is the thickness of the shape outline. Otherwise, a filled shape will be drawn.
- **lineType**
  - It is the type of the shape boundary. OpenCV provides three types of line:
    - cv2.LINE_4: This means four-connected lines
    - cv2.LINE_8: This means eight-connected lines
    - cv2.LINE_AA: This means an anti-aliased line
- **shift**
  - This indicates the number of fractional bits in connection with the coordinates of some points defining the shape

Drawing Lines

- To draw line, call line function

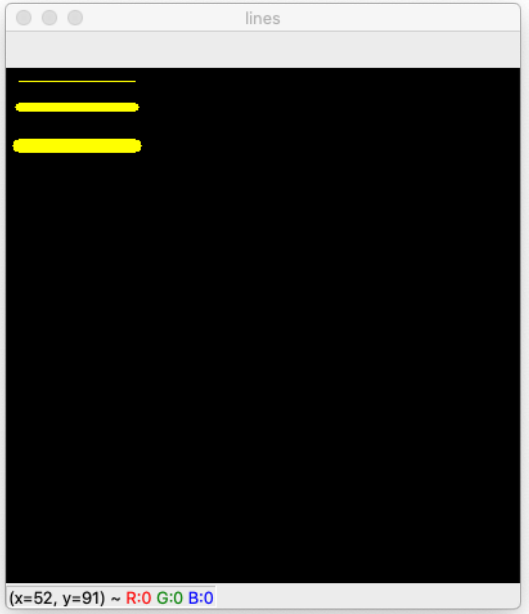
```
cv2.line(img, pt1, pt2, color, thickness=1, lineType=8, shift=0)
```

- E.g.

```
cv2.line(image, (10, 10), (100, 10), (0, 255, 255), 1)

cv2.line(image, (10, 30), (100, 30), (0, 255, 255), 5)

cv2.line(image, (10, 60), (100, 60), (0, 255, 255), 10)
```



Drawing Arrows

- To draw line, call line function

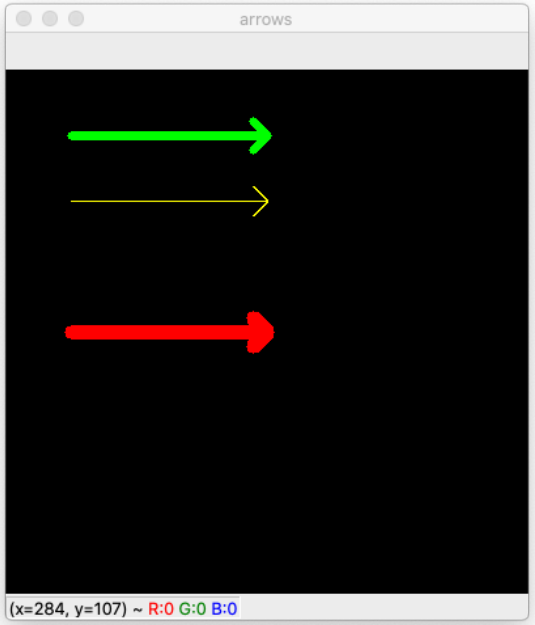
```
cv2.arrowedLine(img, pt1, pt2, color, thickness=1, lineType=8, shift=0, tipLength=0.1)
```

- E.g.

```
cv2.arrowedLine(image, (50, 50), (200, 50), (0, 255, 0), 5, 8, 0)

cv2.arrowedLine(image, (50, 100), (200, 100), (0, 255, 255), 1, 4, 0)

cv2.arrowedLine(image, (50, 200), (200, 200), (0, 0, 255), 10, 8, 0)
```



Drawing Rectangles

- To draw rectangle use following function:

```
cv2.rectangle(img, pt1, pt2, color, thickness=1, lineType=8, shift=0)
```

- E.g.

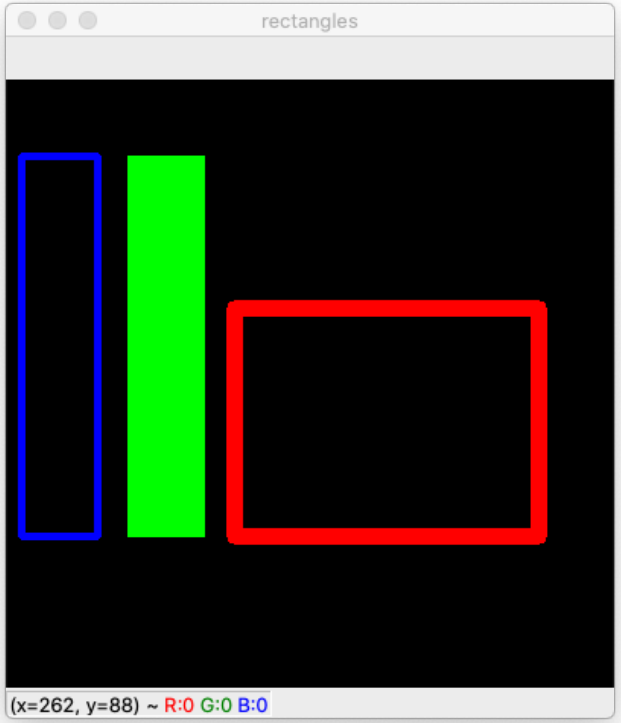
top left

bottom right

```
cv2.rectangle(image, (10, 50), (60, 300), (255, 0, 0), 3)

cv2.rectangle(image, (80, 50), (130, 300), (0, 255, 0), -1)

cv2.rectangle(image, (150, 150), (350, 300), (0, 0, 255), 10)
```



Drawing Circles

- To draw circle use following function:
  - cv2.circle(img, center, radius, color, thickness=1, lineType=8, shift=0)

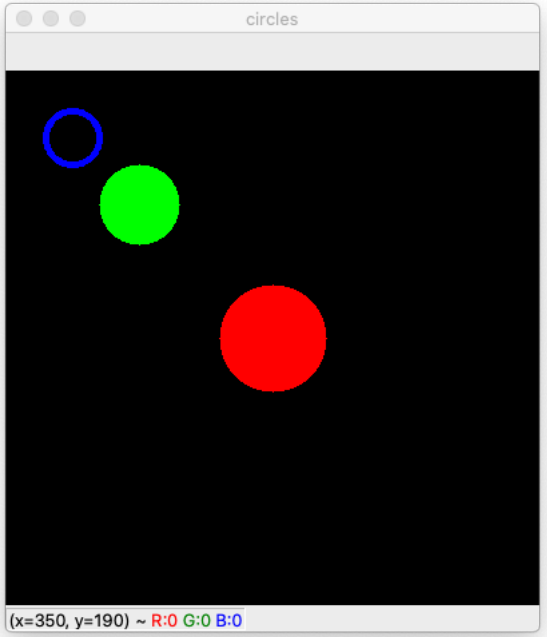
center point

- E.g.

```
cv2.circle(image, (50, 50), 20, (255, 0, 0), 3)

cv2.circle(image, (100, 100), 30, (0, 255, 0), -1)

cv2.circle(image, (200, 200), 40, (0, 0, 255), -1)
```



Drawing Texts

- To draw text in OpenCV, use the following function
  - cv2.putText(img, text, org, fontFace, fontScale, color, thickness=1, lineType= 8)
- E.g.

```
cv2.putText(image, 'OpenCV', (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
    0.9, (0, 255, 0), 2, cv2.LINE_4)

cv2.putText(image, 'OpenCV', (10, 70), cv2.FONT_HERSHEY_DUPLEX,
    0.9, (0, 255, 255), 2, cv2.LINE_8)

cv2.putText(image, 'OpenCV', (10, 110),
    cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 0.9, (0, 0, 255), 2,
    cv2.LINE_AA)
```

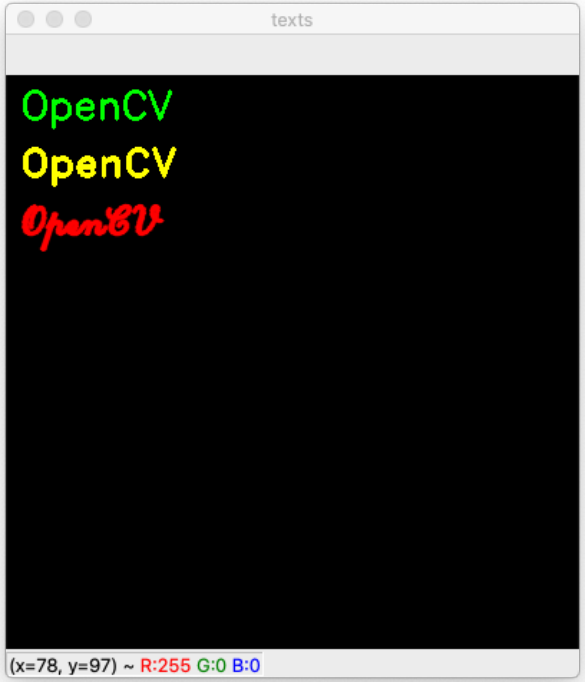


Image Processing Techniques

Introduction

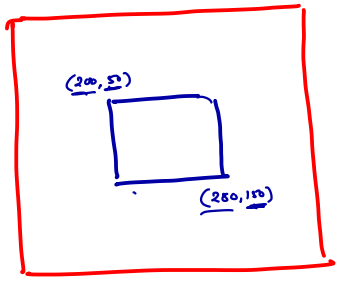
- Image processing is the core of OpenCV
- OpenCV provides various algorithms for image processing
- These algorithms include
  - Splitting and merging channels
  - Geometric transformations of images
    - translation, rotation, scaling, affine transformation, perspective transformation, and cropping
  - Arithmetic with images—bitwise operations (AND, OR, XOR, and NOT) and masking
  - Smoothing and sharpening techniques
  - Morphological operations
  - Color spaces
  - Color maps

Splitting and merging channels

- Sometimes, you have to work with specific channels on multichannel images
- To do this, you have to split the multichannel image into several single-channel images
- E.g.
  - (b, g, r) = cv2.split(image)
- After processing each channel, you can merge them back using merge function
- E.g.
  - Image = cv2.merge((b, g, r))

Image Cropping

- Extracting a segment of an image
- Syntax:
  - img [start_row : end_row, start_col : end_col]



[y1 : y2 , x1 : x2]

```
img = cv2.imread('messi5.jpg')
cropped = img[50:155, 200:271]
cv2.imshow('cropped', cropped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Resizing the image

- Use cv2.resize() to resize the image

```
img = cv2.imread('messi5.jpg')
h, w = img.shape[:2]
new = cv2.resize(img, (w * 2, h * 2))
cv2.imshow('new image', new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Rotations

- Use cv2.warpAffine to implement the translations
- Matrix

$$T = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

- Use cv2.getRotationMatrix2D() to create the matrix

```
img = cv2.imread('messi5.jpg')
h, w = img.shape[:2]
center = (w//2, h//2)
t = cv2.getRotationMatrix2D(center, -90, 1)
new = cv2.warpAffine(img, t, (w, h))
cv2.imshow('new image', new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Translation

- Use cv2.warpAffine to implement the translations
- Matrix

$$T = \begin{bmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \end{bmatrix}$$

```
img = cv2.imread('messi5.jpg')
h, w = img.shape[:2]
t = np.float32([[1, 0, 10], [0, 1, 10]])
new = cv2.warpAffine(img, t, (w, h))
cv2.imshow('new image', new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```