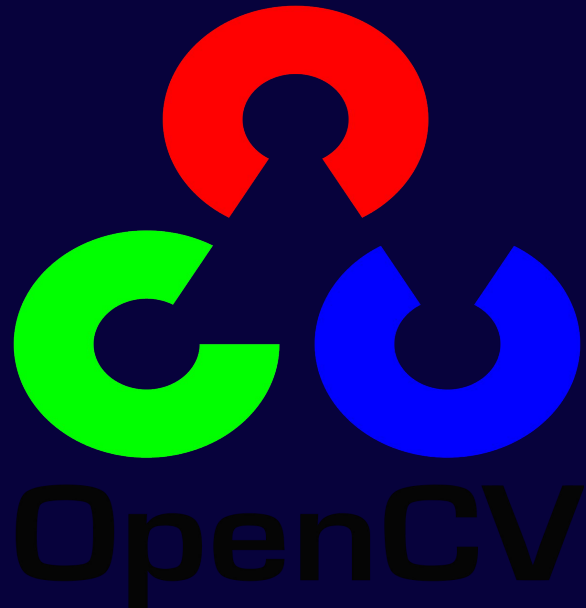


INTRODUCTION TO IMAGE PROCESSING

Electronics and Robotics Club

OVERVIEW

This is an introductory session where we will be basic concepts in basic concepts in Image Processing and a hands-on experience with OpenCV.



WHAT IS AN IMAGE?

Remember we deal with digital images!

An image is a matrix of pixels of size $M \times N$.

Coordinate system is Opencv:

The top left corner is the origin (unlike our usual convention).

PIXELS

The value taken by a pixel is called intensity.

A 8-bit intensity range has 256 possible values.

Number of Channels of an image is the number of primary colours used to make it.

RGB images have 3 channels. A grayscale image has 1 channel.

GRayscale AND RGB IMAGES

In grayscale images, a pixel takes value from 0 to 255. 0 means black and 255 means white.

In RGB images, a pixel takes three values for all RGB from 0 to 255.

Example: (255,0,0) means red and (255,255,255) means white.

The more the value of the pixel, the more the intensity.
Don't get confused!

RGB IMAGES



Notice that the green grass has more intensity in the *G* channel grayscale image, while the red shirt has almost 0 intensity (is almost black).

READING AN IMAGE IN OPENCV

```
import cv2

image = cv2.imread("CM.jpg")
cv2.namedWindow('image', cv2.WINDOW_AUTOSIZE)
cv2.imshow('image', image)
k = cv2.waitKey(0)
if k == 95:           # wait for "a" key to exit
    cv2.destroyAllWindows()
```

Creates a window in which image will be displayed. Function takes a string which is the name of the window and a parameter which describes the window. Here the window can be resized

This function takes an argument in milliseconds and waits for that much time for any keyboard event

CAPTURING A VIDEO

```
cap= cv2.VideoCapture(0)
while True:
    return_val, frame=cap.read()
    cv2.imshow("Showing_video", frame)
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break
```

Creates an object of class VideoCapture.

Cap.read returns an image and a boolean which is True when a non-empty image is captured.

BASIC OPERATIONS

OpenCV uses a BGR format instead of the usual RGB

```
BGR = image[100,30]
B = image[100,30,0]
G = image[100,30,1]
R = image[100,30,2]
blue, green, red = cv2.split(image)
shape = image.shape
```

`image[y, x]` gives pixel value at that point. Notice that it is (y,x) not (x,y).

`Image[y,x,channel_no]` returns the value of intensity of that channel. Example: `channel_no = 2` corresponds to Blue

Returns the shape is the form (rows, cols, channels)

This function splits the image in BGR components respectively

CONVERSION BETWEEN COLOR SPACES


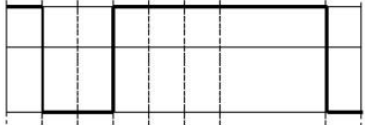
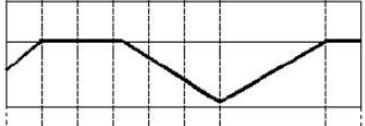
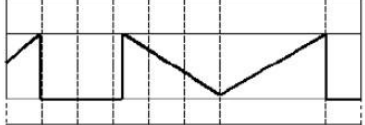
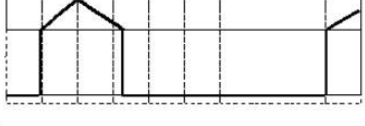
```
image = cv2.imread("CM.jpg", 1)
cv2.namedWindow('image', cv2.WINDOW_AUTOSIZE)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

The second argument
0- reads as grayscale
1- reads as BGR

cvtColor takes two
arguments. First is the
image and the second is
the conversion type

THRESHOLDING IN GRAY SCALE IMAGES

SIMPLE THRESHOLDING

THRESH_BINARY Python: cv.THRESH_BINARY	$\text{dst}(x,y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$		Threshold Binary
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$		Threshold Binary, Inverted
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\text{dst}(x,y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x,y) > \text{thresh} \\ \text{src}(x,y) & \text{otherwise} \end{cases}$		Truncate
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\text{dst}(x,y) = \begin{cases} \text{src}(x,y) & \text{if } \text{src}(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$		Threshold to Zero, Inverted
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{thresh} \\ \text{src}(x,y) & \text{otherwise} \end{cases}$		Threshold to Zero

EXAMPLE:

```
ret, thresh1 = cv2.threshold(img, 127, 255, cv.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 127, 255, cv.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 127, 255, cv.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 127, 255, cv.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 127, 255, cv.THRESH_TOZERO_INV)
```

ADAPTIVE THRESHOLDING

Adaptive thresholding chooses a threshold for a pixel based on its neighboring pixels

There are two ways to calculate this threshold:

- Using mean: Given a window size (let's say 5×5) we calculated the mean for all pixels in that window and then use that as a threshold. Note that you will need to specify what type of thresholding you need (one of the 5 types above)
- Using weighted mean: This is similar as above only that weights are chosen from a normal distribution

```
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,11,2)  
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)
```

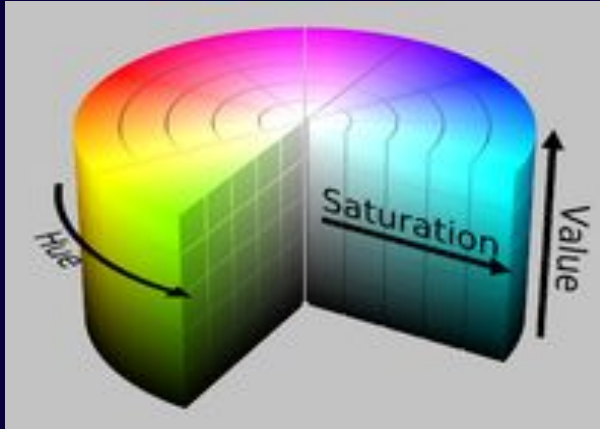
Size of the window

BITWISE OPERATIONS ON BINARY IMAGES

We can perform add, or, xor, not operations on binary images(Consider them as bools)

```
img1_and = cv.bitwise_and(img1, img2)
img1_or = cv.bitwise_or(img1, img2)
img1_not = cv2.bitwise_not(img1)
dst = cv.addWeighted(img1,0.7,img2,0.3,0)
```

HSV COLOR SPACE



Angle

0-60

60-120

120-180

180-240

240-300

300-360

Color

Red

Yellow

Green

Cyan

Blue

Magenta

Saturation indicates the amount of grey in a color. More grey, less saturation. The primary color has a saturation of 1.

Value(or Brightness) amount of white in a color. More black, less value. Black has a value of zero.

To reach a particular color's HSV value on a cylinder, the distance travelled from the circumference to center is saturation and the depth one needs to travel is the value.

DETECTING A COLOR - USING THE `inRange` FUNCTION

This function is used to threshold the hsv image to get a specific color

```
cap= cv2.VideoCapture(0)
while True:
    return_val, frame=cap.read()
    hsv= cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower = np.array([110,50,50])
    upper = np.array([130,255,255])
    mask = cv2.inRange(hsv, lower, upper)
    cv2.imshow("Showing_video", mask)
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break
```

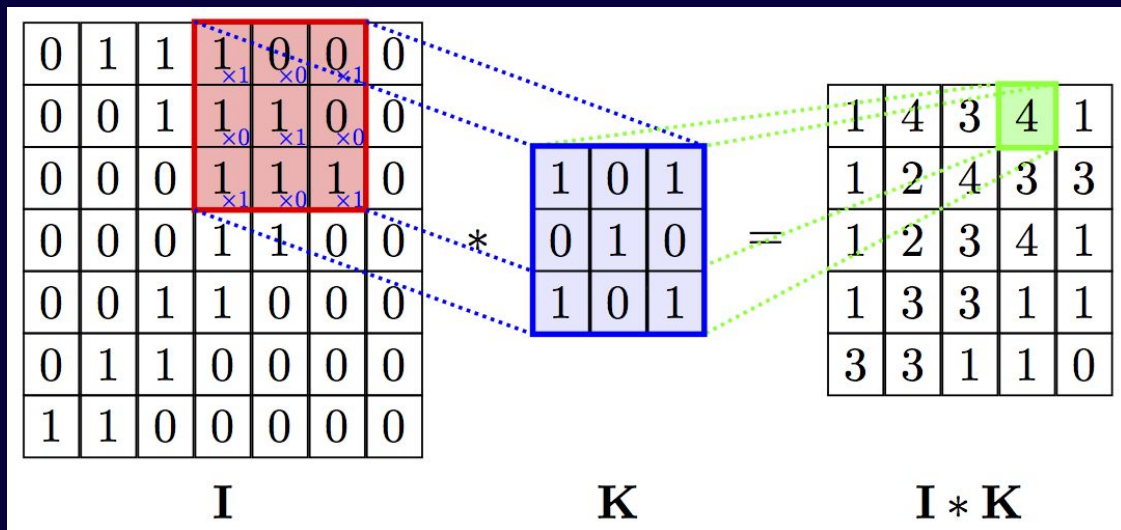
Defining a range for the color. If the Hue value of a color is H , a good range is $(H-10, H+10)$

`inRange` function returns a binary image in which white pixels are the ones which fall in the specified range

KERNEL

A kernel is nothing but a small matrix, also called convolution matrix.

CONVOLUTION



BLURRING IMAGES

Averaging: The center of a $n \times n$ kernel is replaced by the average of the n^2 pixels in it.

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For a 3×3 kernel

```
blurred_image=cv2.blur(image, (3,3))
```

Note that the kernel size is always an odd integer.

Gaussian Blur:

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Notice that if you plot a 3D surface of matrix indices vs matrix element, you'll get a gaussian :D

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

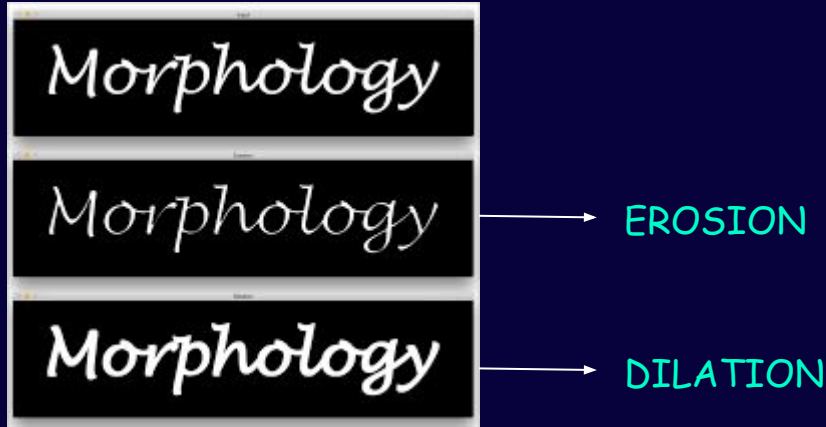
The last parameter is just a number that you want to add to every pixel. Not very important

Median Blur (`cv2.medianBlur`) : The center of a $n \times n$ kernel is replaced by the median of the n^2 pixels in it.

Highly useful in reducing noise from images. In the above filters, a pixel is set to a calculated value that might not belong to any surrounding pixels. In this method, we set a pixel to value which is present in its neighbourhood. This reduces noise very effectively.

MORPHOLOGICAL OPERATIONS

Morphological operations are based on a structuring element or a kernel acting on an image. It is normally performed with binary images.



EROSION, DILATION AND OPENING

EROSION : We choose a window size for the image which slides across the image. If all the pixels in the window are white, then no changes are made. If not, then all the white pixels are converted to black. Hence the boundary gets eroded.

DILATION: The exact opposite of erosion happens. If there is at least one white pixel in the window, the entire window is made white. Hence the size of the object increases

OPENING: It's just erosion followed by dilation. In this way, noise gets removed from the object without disturbing the object size much

EXAMPLE: TRY IT OUT

```
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
dilation = cv2.dilate(img,kernel,iterations = 1)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

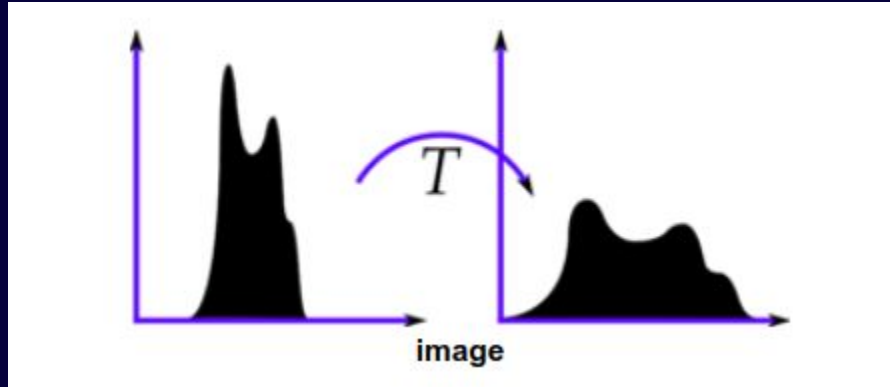
Make sure the image is a binary image. How will get a binary Image from an image? Thresholding!

WHAT IS A HISTOGRAM?

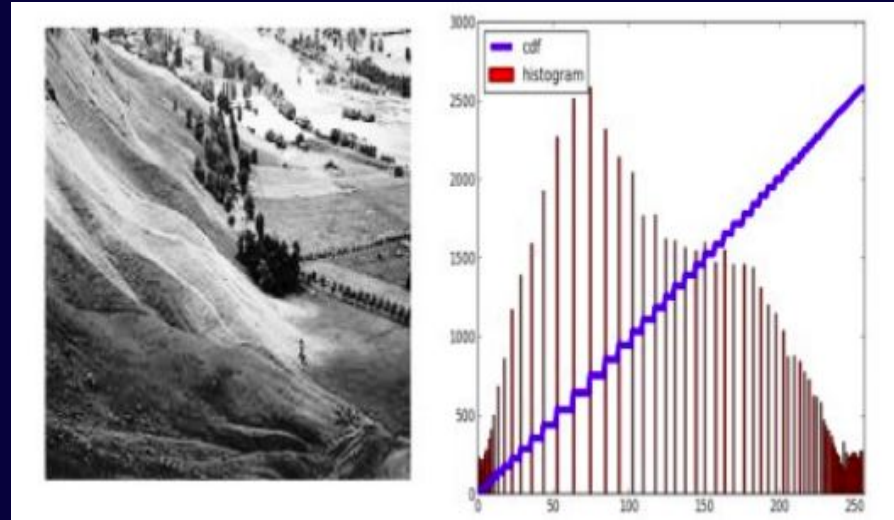
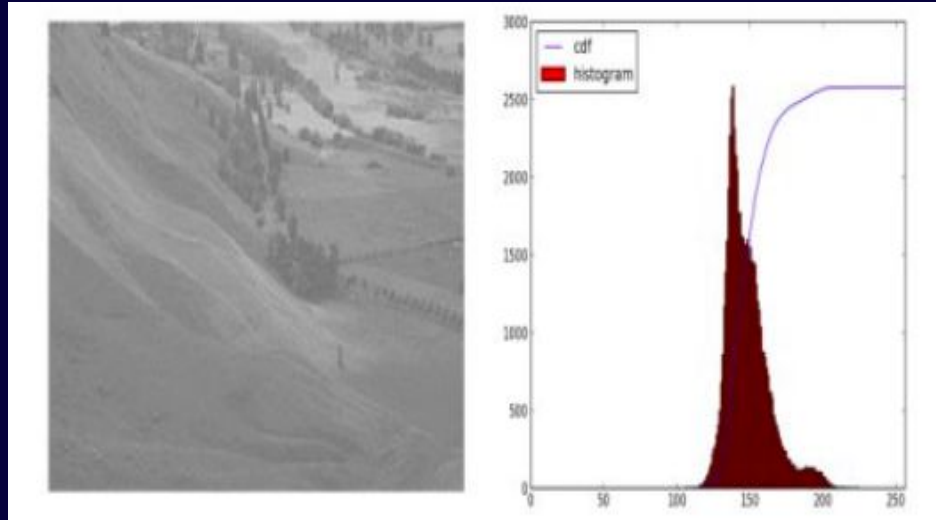
A Histogram is a graph of pixel values on x-axis and corresponding number of pixels on the Y-axis.

Histograms give an overall idea of the intensity distribution of an image.

A histogram with more variance implies better contrast than a histogram where most of the pixels take close values. So.. How to improve contrast.



HISTOGRAM EQUALISATION



Find the function yourself :P

DRAWING FUNCTIONS IN OPENCV

`cv2.line(image, Point1, Point2, Color, thickness)`

`cv2.rectangle(image, top_left_corner, bottom_right_corner, color, thickness)`

`cv2.circle(image, center, radius, color, thickness)`

`cv2.ellipse(image, center, (minor_axis, major_axis), angle1, angle2, angle3, color, thickness)`

angle1= angle of rotation of ellipse. angle2= starting angle of the arc. angle3= ending angle of the arc

```
img = np.zeros((512,512,3), np.uint8)
cv2.line(img, (0,0), (511,511), (255,0,0), 5)
cv2.rectangle(img, (384,0), (510,128), (0,255,0), 3)
cv2.circle(img, (447,63), 63, (0,0,255), -1)
cv2.ellipse(img, (256,256), (100,50), 0,0,180,255, -1)
```

DRAWING FUNCTIONS IN OPENCV

Drawing a polygon:

`cv2.polylines(image, array_of_points of shape ROWS \times 1 \times 2 Boolean_value, color)`

The boolean value is true if the points form a closed polygon.

```
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
cv2.polylines(img,[pts],True,(0,255,255))
```

EDGE DETECTION

The theory of edge detection is out of the scope of this session. So we will use an inbuilt edge detector in opencv which is the **Canny edge detector**.

Canny edge detection works on gray scale images only. It returns a binary image in which white pixels are edges of objects.

It is intuitive that edge detection is sensitive to noise. So, it is a good practice to blur the image first.



EXAMPLE:

```
cap= cv2.VideoCapture(0)
while True:
    return_val, frame=cap.read()
    gray= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray,100,200)
    cv2.imshow("Showing_video", edges)
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break
```

The last two parameters are thresholds for lengths of the edges.

CONTOURS IN AN IMAGE

A Contour is a closed curve joining all continuous points having same color or intensity. This is a important tool in shape analysis and object detection.

Better accuracy is obtained when we use binary images. So before finding contours we can apply thresholding or canny edge detection.

Contours of an image are basically a 2D array of points.

FINDING AND DRAWING CONTOURS IN AN IMAGE

```
cap= cv2.VideoCapture(0)
while True:
    return_val, frame=cap.read()
    gray= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    ret,thresh1 = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
    im2, contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(frame, contours, -1, (0,255,0), 3)
    cv2.imshow("Showing_video", frame)
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break
```

The function **findContours** takes in a binary or grayscale image as input and outputs the modified image and a 2D array of points

The function **drawContours** takes arguments as the image on which contours are to be drawn, contours, the index of the contour to be drawn, the color of the lines and the thickness of the lines

MOMENTS OF A CONTOUR

Used to calculate the centre of a contour.

```
cnt = contours[0]
M = cv2.moments(cnt)
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

- Moments:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

- Corner orientation:

$$c_x = \frac{M_{10}}{M_{00}}, \quad c_y = \frac{M_{01}}{M_{00}}$$

$$C_{ori} = \tan^{-1} \left(\frac{c_y}{c_x} \right)$$

OTHER THINGS TO DO WITH CONTOURS

Finding Area of contours

```
area = cv2.contourArea(contour)
```

Finding Perimeter

```
perimeter = cv2.arcLength(contour, True)
```

If the second argument is false, the contour is just a curve and not closed.

Contour Approximation

```
epsilon = 0.1*cv2.arcLength(contour, True)  
approx = cv2.approxPolyDP(contour, epsilon, True)
```

approx is the array of points which are vertices of contour.
If $\text{len}(\text{approx})=4 \rightarrow$ it is a four sided polygon.

MORE THINGS TO DO WITH CONTOURS

To check symmetry of the contours. Fit a circle or an ellipse

```
ellipse = cv.fitEllipse(cnt)
cv.ellipse(img,ellipse,(0,255,0),2)
```

```
(x,y),radius = cv.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv.circle(img,center,radius,(0,255,0),2)
```

BECAUSE WE ALL HAVE ML ENTHU

When a beginner asks for recommendations
for studying machine learning



YOLO BROLO: YOU ONLY LOOK ONCE

<https://pjreddie.com/darknet/yolo/>

COMMON ARCHITECTURES TO USE:

<https://github.com/puzzledqs/BBox-Label-Tool>

<https://github.com/fizyr/keras-retinanet>

THANK YOU :D