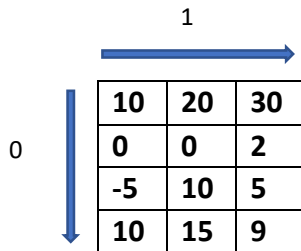


## NUMPY

El axis en numpy indica una dimensión sobre el cual se quiere realizar alguna operación matemática.

```
A=np.array( [ [10,20,30] , [0 ,0 ,2] , [ -5, 10 , 5 ], [10 , 15, 9] ] )
```



10	20	30
0	0	2
-5	10	5
10	15	9

**APLANAMIENTO (flatten):**

Si se desea obtener un arreglo plano o de una sola dimensión a partir de un matriz de mayor dimensión se hará uso de la función `flatten()`

```
B=np.flatten(A,order='C')
```

**RESULTADO:**

[10 , 20, 30 , 0, 0, 2, -5, 10, 5, 10, 15, 9] del tipo `numpy.ndarray`

```
B=np.flatten(A,order='F')
```

**RESULTADO:**

[10 , 0,-5 ,10 ,20 ,0 ,10 ,15 ,30 ,2 ,5 ,9] del tipo `numpy.ndarray`

**MAXIMO:**

Para calcular el máximo valor del arreglo bidimensional se hará uso del método `max`

```
B=np.max(A,axis=0)
```

Resultado: [10,20,30] del tipo `numpy.ndarray`

```
B=np.max(A,axis=1)
```

Resultado: [30,2,10,15] del tipo `numpy.ndarray`

```
B=np.max(A)
```

Resultado: 30

### **ORDENAMIENTO:**

Para ordenar un array de menor a mayor se hará uso de la función `sort()`

Tomemos como ejemplo la matriz anterior

```
B= np.sort(A,axis=0)
```

**RESULTADO:**

```
B=np.sort(A,axis=1)
```

**RESULTADO:**

## SEÑAL DISCRETA

Una señal discreta esta representando por una secuencia de valores.

$x = [1, 2, 10, 20, -1, 0, 5.4]$

## GRAFICAS EN PYTHON Y MATPLOTLIB

Para realizar gráficos de secuencias discretas se hará uso del paquete matplotlib con el fin de importar el módulo pyplot .

**from matplotlib import pyplot as plt**

## OPERACIONES MATEMATICAS

### SUMA:

Se puede sumar dos señales discretas con la condición de que ambas tengan las mismas longitudes

$x_1 = [1, 2, 10, 20, -1, 0, 5.4]$

$x_2 = [1, -2, 0, 20, 1, 0, 0]$

$x_3[n] = x_1[n] + x_2[n]$

### RESTA

Se puede restar dos señales discretas con la condición de que ambas tengan las mismas longitudes

$x_1 = [1, 2, 10, 20, -1, 0, 5.4]$

$x_2 = [1, -2, 0, 20, 1, 0, 0]$

$x_3[n] = x_1[n] - x_2[n]$

## MULTIPLICACION

Se puede multiplicar elemento a elemento dos señales discretas con la condición de que ambas tengan las mismas longitudes.

$x_1[n] = [1, 2, 10, 20, -1, 0, 5.4]$

$x_2[n] = [1, -2, 0, 20, 1, 0, 0]$

$x_3[n] = x_1[n] \cdot x_2[n]$

$x_3 = [1, -4, 0, 400, -1, 0, 0]$

### PLEGADO:

La operación de plegado es un cambio en el orden de la secuencia de los valores.

$$x1[n] = [1, 2, 10, 20, -1, 0, 5.4]$$

se realizará el plegado (folding) sobre la secuencia x1

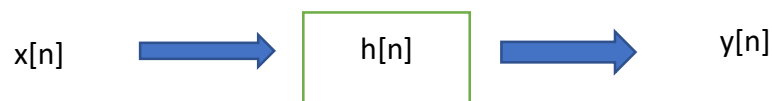
$$x2[n] = x1[-n]$$

### CONVOLUCION:

Operación matemática que involucra suma, multiplicación y cambios

#### SISTEMA DISCRETO

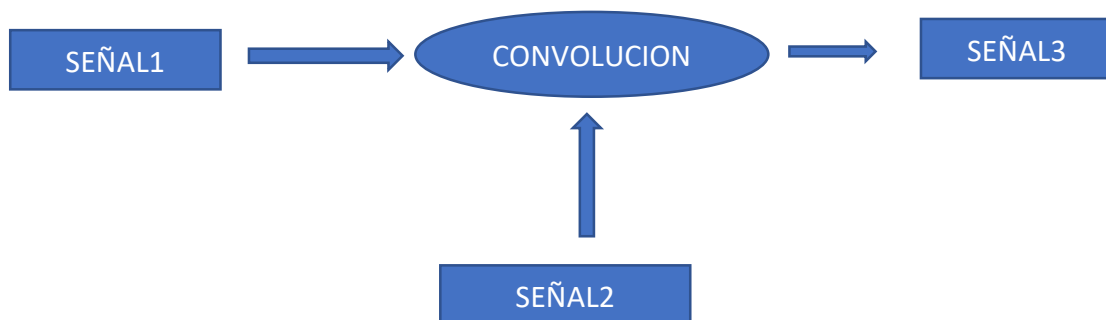
Un sistema discreto está representado por  $T[\cdot]$  que no es más que un operador matemático que recibe como excitación una secuencia o señal discreta y produce una respuesta otra secuencia o señal discreta.



#### CONVOLUCION DISCRETA

Operación matemática de extenso uso en el tratamiento digital de señales, sistemas de control, procesamiento de imágenes entre otras áreas.

La convolución discreta es aplicada cuando se asumen sistemas que incorporan información temporal o de otra variable en tiempo discretos, para tiempo continuo esto se denomina convolución continua.



## OPENCV

### Escalamiento

El scaling o escalamiento se refiere al proceso de redimensionar una imagen. En opencv podemos utilizar el método `resize` para tal propósito.

#### CODIGO:

```
import numpy as np
import cv2
img=cv2.imread('imagen1.png')
#definir 2 variables para ejemplo practico
fx,fy=300,300
"""
el metodo resize recibe 3 argumentos
imagen de origen: puede ser RGB o escala de grises u otro espacio de color
tupla : representa las nuevas dimensiones en filas y columnas
interpolation: representa el metodo de interpolacion usado para lograr el escalamiento
"""
img_new=cv2.resize(img,(fx,fy),interpolation=cv2.INTER_AREA)
#mostrar imagen original
cv2.imshow("IMAGEN ORIGINAL",img)
#mostrar nueva
cv2.imshow("IMAGEN MODIFICADA",img_new)
#esperar a que se presione un boton para cerrar las ventanas
cv2.waitKey(0)
#cerrar ventanas despues de presionar una tecla
cv2.destroyAllWindows()
```

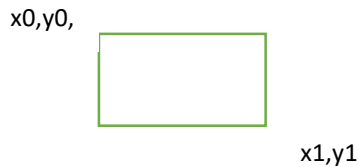
#### RESULTADO:



### DIBUJAR RECTANGULOS EN OPENCV

Con el fin de dibujar un rectángulo sobre una imagen se hará uso de la función rectangle de opencv de la siguiente manera:

`cv2.rectangle(imagen,(x0,y0),(x1,y1),(0,255,0),2)`



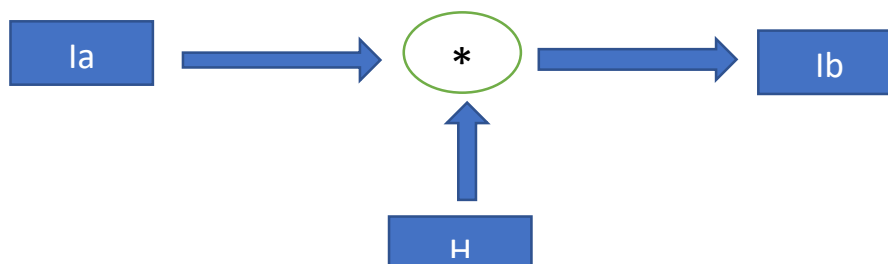
la tupla (x0,y0) representa la coordenada superior e izquierda del rectángulo.

la tupla (x1,y1) representa la coordenada inferior y derecha del rectángulo .

la tupla (0,255,0) representa el color de los bordes del rectángulo .

el valor entero 2 representa el grosor de los bordes del rectángulo.

## CONVOLUCION DISCRETA EN IMÁGENES



Ia :imagen original con dimensiones RxQ

H: kernel o mascara con dimensiones MxN

Ib:Imagen resultante con dimensiones (R-M+1) x (Q-N+1)

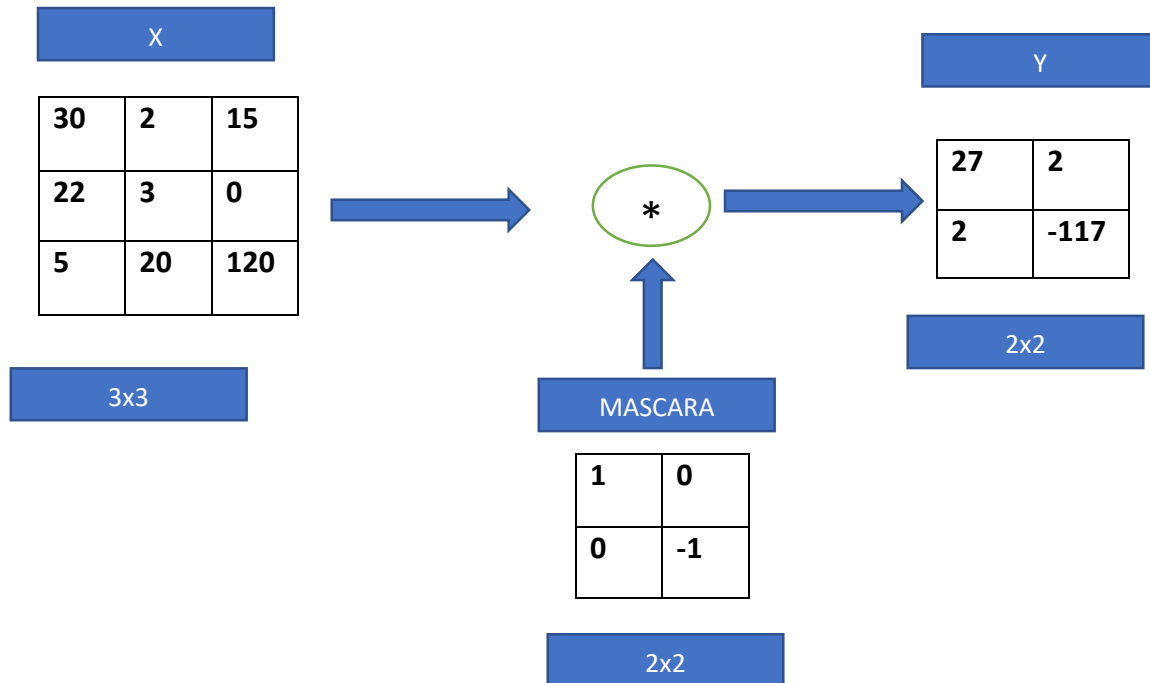
## FORMULA DE CONVOLUCION

Primera forma

$$Ib [x, y] = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} H[i, j] \cdot Ia[x - i, y - j]$$

Segunda forma

$$Ib [x, y] = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} H[i, j] \cdot Ia[x + i, y + j]$$



### FILTRADO DE IMÁGENES DIGITALES

El filtrado de imágenes digitales es una operación de convolución entre una imagen y un kernel que producirá una nueva imagen con características extraídas gracias al kernel .

En opencv podemos utilizar el método `filter2D()`.

**`Ib=cv2.filter2D(Ia,-1,kernel)`**

**Ia=imagen original**

**Kernel=mascara**

## Template Matching

Método para realizar búsqueda y localización de una imagen dentro de otra imagen mas grande.

Metodo de template matching

`cv2.TM_CCOEFF_NORMED`

`cv2.TM_CCORR_NORMED`

`res = cv2.matchTemplate(img,template,method)`



## DETECCION DE ROSTROS MEDIANTE HAAR CASCADE

La detección de objetos usando Haar cascade es un método que fue propuesto por Paul Viola y Michael Jones en el paper “**Rapid Object Detection using a Boosted Cascade of Simple Features**” en el año 2001.

Opencv viene con algoritmos ya entrenados para detección de rostros , ojos , sonrisas en otros . para el caso de detección de rostros lo que se debe de hacer es crear o instanciar un objeto mediante de la clase

**CascadeClassifier**('haarcascade\_frontalface\_default.xml') en la cual se le tiene que pasar como argumento lo que se quiere detectar , para el caso de rostros de forma frontal se hara uso del archivo con extensión (.xml )  
'haarcascade\_frontalface\_default.xml' .

### Ejemplo:

Crear objeto para luego detectar caras , esto puede cambiar dependiente que se desea localizar .

```
Caras= CascadeClassifier('haarcascade_frontalface_default.xml')
```

Con el objeto ya creado ahora podemos hacer uso del método **detectMultiScale(imagen,param1,param2)** que realizara la búsqueda o detección de rostros sobre la imagen que se le pasa como argumento, ajustando los parámetros param1 y param2 se puede calibrar la precisión .

El método **detectMultiScale(imagen,param1,param2)** retorna un arreglo de tipo numpy.ndarray y almacena las coordenadas x,y que representa a la región en forma rectangular donde se detecta un rostro o mas de uno. Si en caso no encuentra ningún rostro devuelve un None.

### Ejemplo :

Creado el objeto anteriormente , se procede a usar el método **detectMultiScale()**

**Suponer que imag representa a una imagen digital en escala de grises y que existen 2 rostros.**

```
faces=caras. detectMultiScale(imag,param1,param2)
```

**faces = [ [100,200,50,30], [ 300,250,50,50] ]** de tipo numpy.ndarray

**para el primero rostros [100,200,50,30] ,xo=100,yo=200 , w=50 , h=30**

## DETECCION DE ROSTROS SOBRE UNA IMAGEN

```
import numpy as np
import cv2
#crear un objeto para la detección de rostros frontales

caras=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#leer una imagen

img=cv2.imread('planck_eistein.jpg')

#convertir a escala de grises

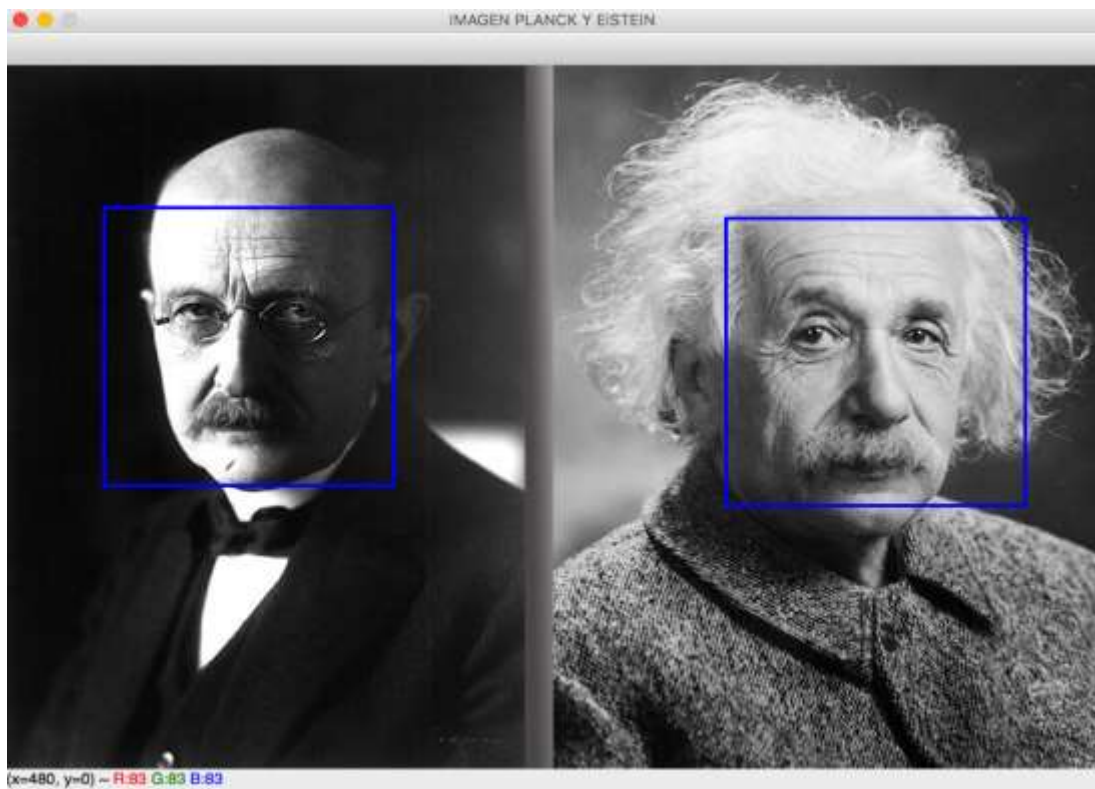
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#detectar rostros en el objeto caras

faces =caras.detectMultiScale(gray,1.2,3)

#si en caso hay caras
if len(faces) !=0:
    print("se detecto rostros en la imagen")
    #dibujar un rectangulo con las coordenadas de la región de las caras
    for (x, y, w, h) in faces:
        #dibujar el rectángulo para cada rostro detectado
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
    #mostrar la imagen
    cv2.imshow("IMAGEN",img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
#si en caso no hay caras
else:
    print("NO SE DETECTO")
```

RESULTADO:

Se logro detectar con éxito los rostros.



## DETECCION DE OJOS CON OPENCV

Una vez que se ha detectado la región rectangular donde se encuentra el rostro, a continuación, se procederá a buscar y encontrar los ojos de la persona.

```
# importar opencv
```

```
import cv2
```

```
# crear un objeto para la detección de ojos
```

```
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

```
#suponer que se tiene la región rectangular que contiene a  
un rostro
```

```
roi_gray = gray[y:y+h, x:x+w]
```

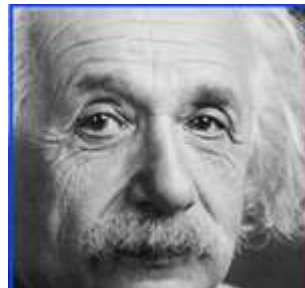
```
# buscar ojos
```

```
eyes = eye_cascade.detectMultiScale(roi_gray)
```

REGION DE CARA 1



REGION DE CARA 2



## MQTT

Es un protocolo machine to machine (M2M ) diseñado con el fin de transmitir y recibir datos poco ancho de banda que normalmente se obtiene a partir de sensores y un dispositivo electrónico para la adquisición de esos datos.

**MQTT(Message Queuing Telemetry Transport).**

**Terminos básicos : router, wifi , ip ,mascara**

### Router

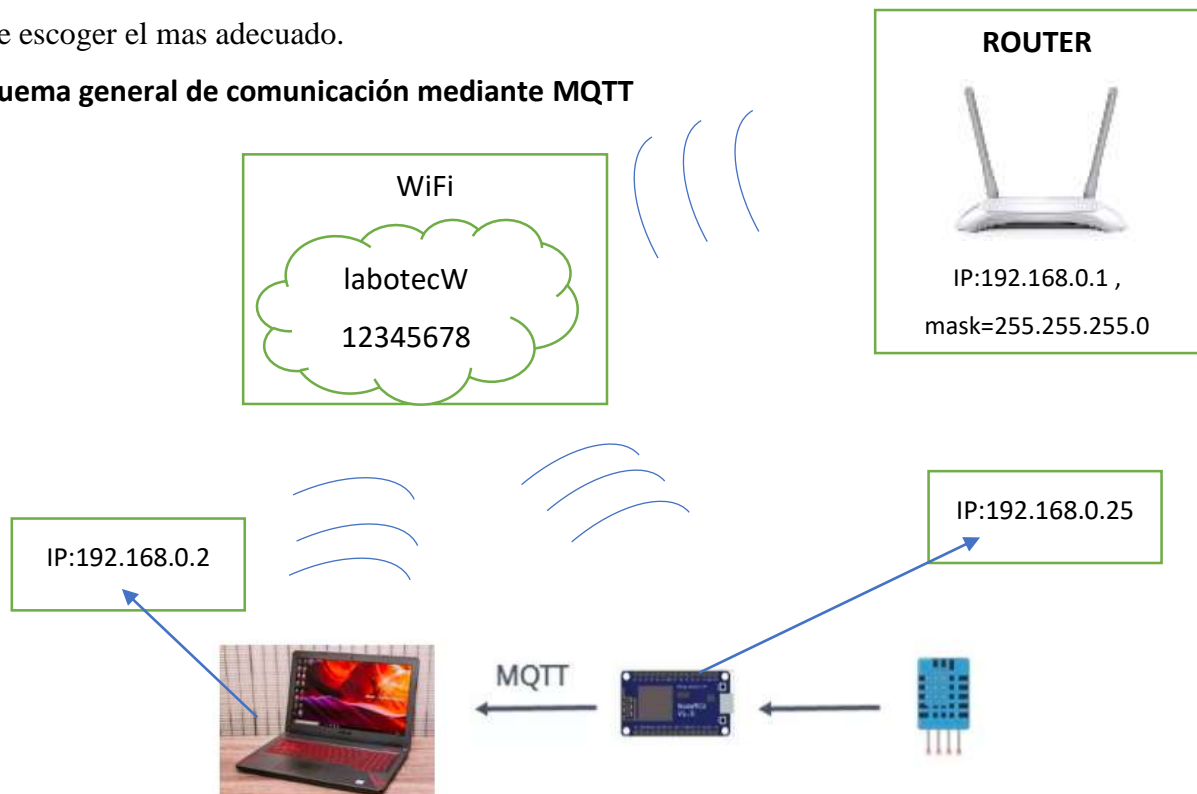
(enrutador) es el encargado de conectar redes además este genera una red wifi a la cual nos conectaremos mediante el uso del SSID y el PASSWORD.

La dirección IP de router es llamado gateway o puerta de enlace y junto a su mascara de red permite asignar direcciones IP a los dispositivos conectados a la red que este genera y que debe ser diferente , el encargado de hacer eso se llamado el servidor **DHCP**(esto es configurable entrando a la configuración del router).

### WiFi

es una tecnología de comunicación que opera en frecuencia 2.4 GHz y 5 GHz , algunos routers pueden operar en ambas frecuencias , dependiendo de modelo y del uso que se le quiera dar se debe escoger el mas adecuado.

**Esquema general de comunicación mediante MQTT**



## **IP**

La dirección IP representa identificación de un dispositivo conectado a una red en la cual si existen mas de un dispositivo conectado a una misma red , la dirección IP no se puede repetir en ningún ordenador u otro dispositivo con conexión a una red.

Existen 2 tipos de IP según su dinámica

### **IP estática**

### **IP dinámica**

## **Mascara**

La mascara de red juega un papel importante en redes y conectividad debido a que su valor junto con el IP permiten saber cuántos dispositivos como máximo se pueden tener en una misma red.