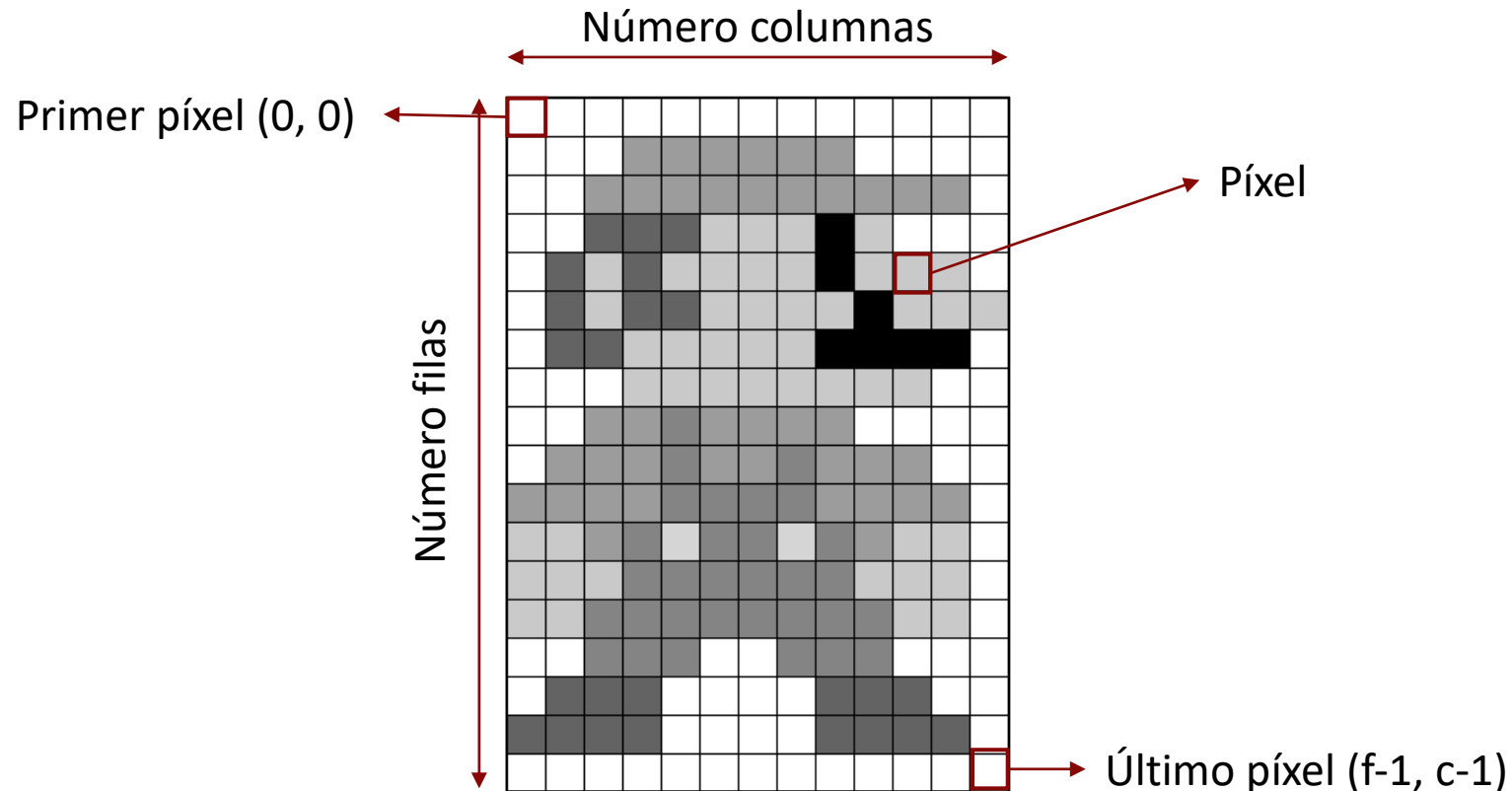


# Tema 1. Procesamiento de imagen

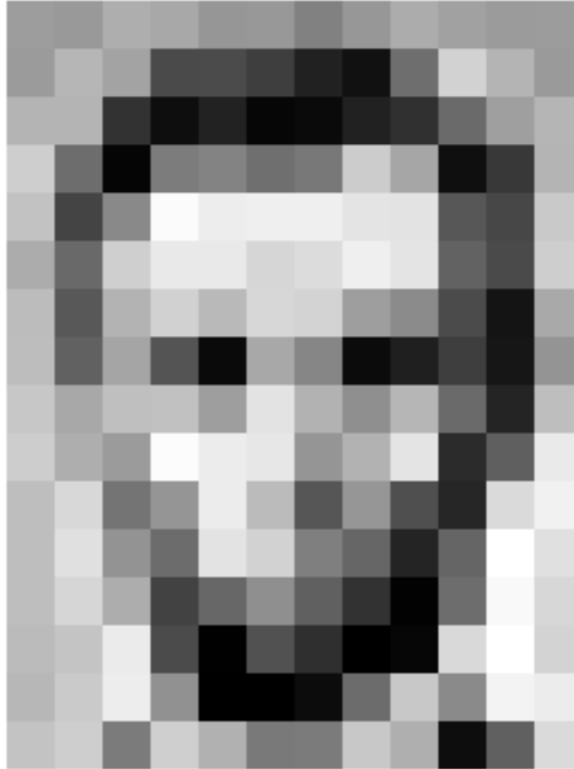
Ana Jiménez Pastor  
anjipas@gmail.com

# Imagen digital

Una **imagen** es una matriz bidimensional de tamaño  $(f, c)$  donde  $f$  es el número de filas y  $c$  el número de columnas. Una imagen **digital** está formada por un número finito de **píxeles**. Cada píxel  $(i)$  se encuentra en una **coordenada** dada  $(f_i, c_i)$  y tiene una determinada **intensidad**. Los píxeles toman **valores de intensidad** entre 0 (negro) y 255 (blanco) (enteros sin signo de 8 bits).



# Imagen digital



Lo que nosotros vemos

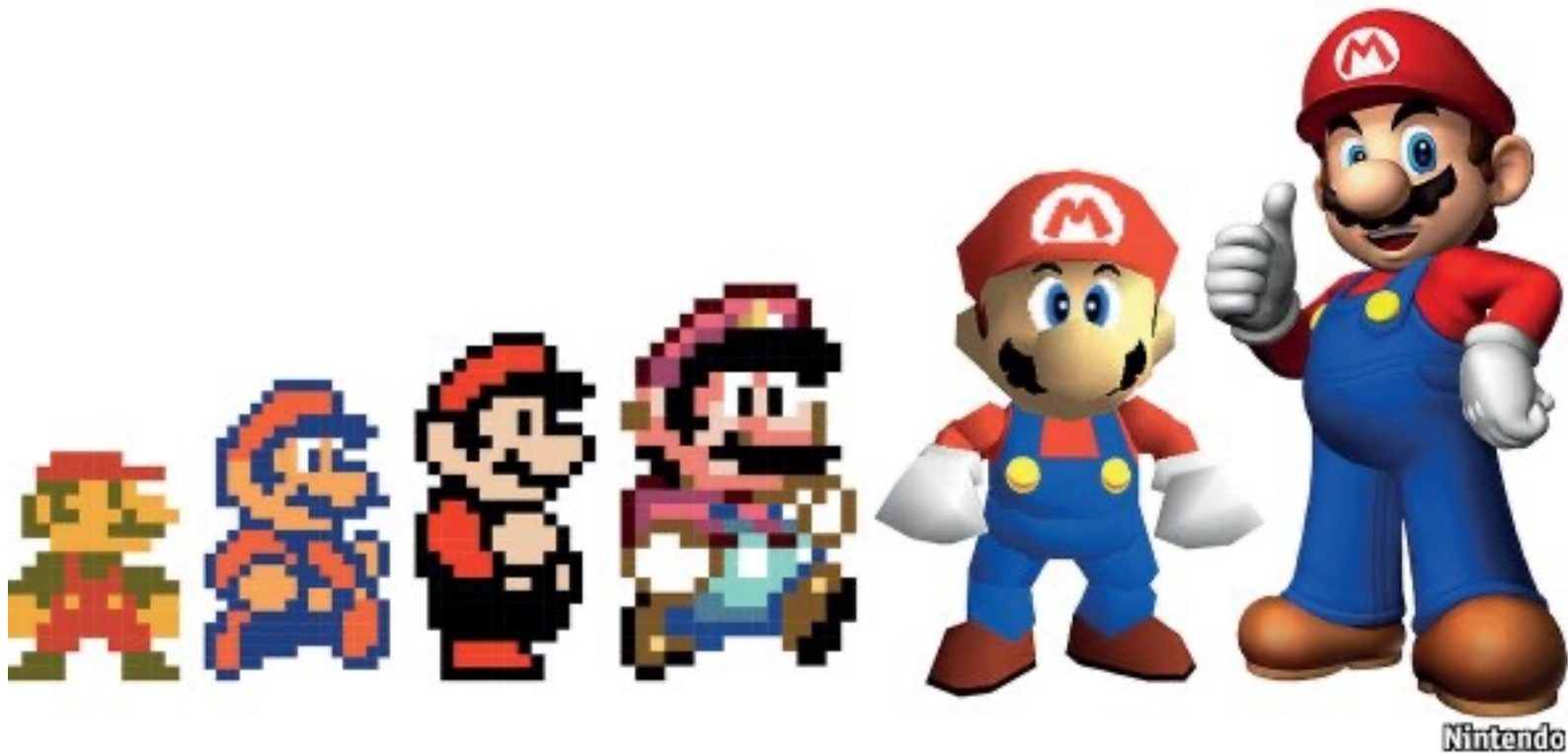
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	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

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	106	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

Lo que el ordenador ve

# Imagen digital

La digitalización implica que una imagen digital realmente es una **aproximación** de una escena real. La similitud entre esta aproximación y la realidad va a depender de la **resolución** de la imagen.



# Imagen digital

La imagen digital puede presentar diferentes **codificaciones**. Así, una imagen realmente se representa como una matriz con tres componentes (f, c, ch) donde ch hace referencia al **número de canales** que tiene la imagen y que va a diferenciar las diferentes formas de codificar una imagen:

- Imagen en escala de grises → Formada por un único canal → Tamaño: (f, c, 1)
- Imagen en color → Formada por varios canales en función del tipo de codificación. Las más conocidas son:
  - RGB (*Red, Green, Blue* – Rojo, Azul, Verde): Formada por 3 canales → Tamaño (f, c, 3) → Más empleada
  - HSV o HSB (*Hue, Saturation, Value/Brightness* – Matiz, Saturación, Valor/Brillo): Formada por 3 canales → Tamaño (f, c, 3)
  - CMYK (*Cyan, Magenta, Yellow, Key* – Cian, Magenta, Amarillo, Negro): Formada por 4 canales → Tamaño (f, c, 4)



Escala de grises



R



G

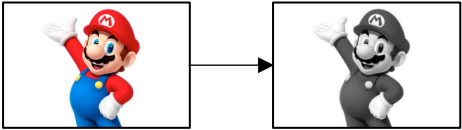
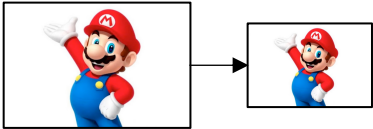
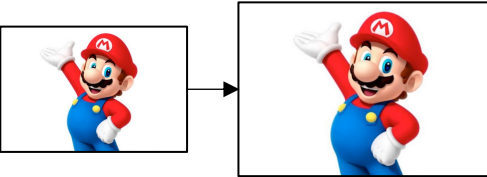
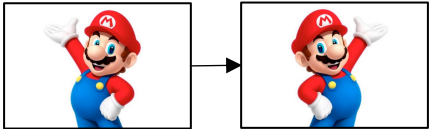
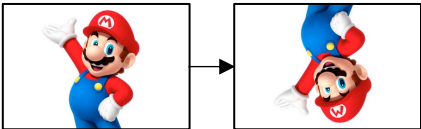






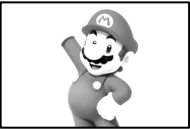
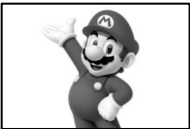
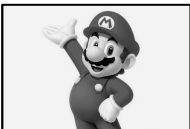


B

RGB

# Procesamiento de imagen

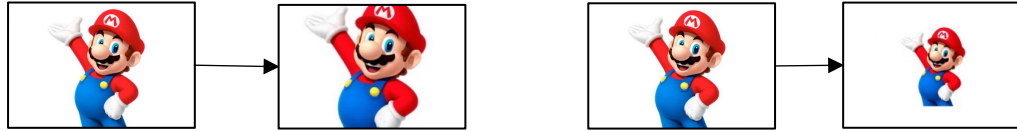
El procesamiento de imagen agrupa todas las técnicas que aplican **transformaciones** a una imagen de entrada para generar una **nueva imagen** de salida. Algunas de estas técnicas nos van a ser de gran utilidad a la hora de preparar las imágenes para entrenar un modelo de IA.

- Cambio de codificación 
- Redimensionado:  
- Giros:  
- Rotaciones 
- Filtrado  
     
     
Gaussiano      De media      De nitidez

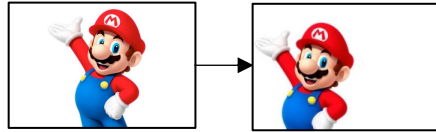
# Procesamiento de imagen

El procesamiento de imagen agrupa todas las técnicas que aplican **transformaciones** a una imagen de entrada para generar una **nueva imagen** de salida. Algunas de estas técnicas nos van a ser de gran utilidad a la hora de preparar las imágenes para entrenar un modelo de IA.

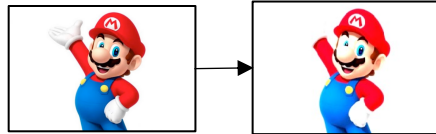
- Zoom



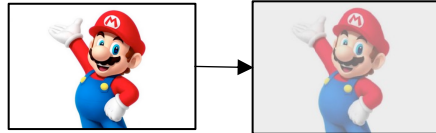
- Traslaciones:



- Cambio brillo:



- Cambio contraste:



# Procesamiento de imagen

Antes de empezar a ver código, si no lo tenéis ya, vamos a instalar OpenCV en el entorno que estéis trabajando.

Para ello, en terminal, vamos a ejecutar el comando:

- Si tenéis un entorno de Anaconda:  
**conda install -c conda-forge opencv**
- Si tenéis un entorno de pip:  
**pip install opencv-python**
- Si tenéis un entorno con poetry:  
**poetry add opencv-python**



# Procesamiento de imagen

## Codificación

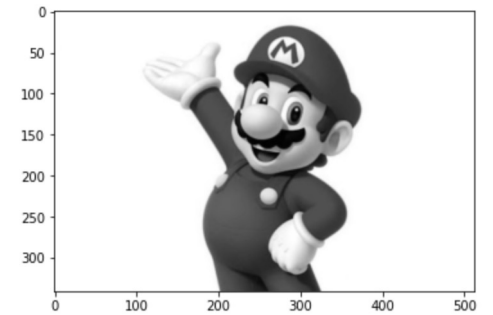
Es posible cambiar la codificación de las imágenes: RGB-Escala de grises, RGB-HSV, Escala de grises-RGB...

```
# Convertimos la imagen a escala de grises
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# Mostramos tamaño de la nueva imagen
print('Tamaño de imagen escala de grises: ', img_gray.shape)

# Convertimos imagen a codificación HSV
img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

# Mostramos tamaño
print('Tamaño de imagen HSV:', img_hsv.shape)
```



Tamaño de imagen escala de grises: (341, 512)

Tamaño de imagen HSV: (341, 512, 3)

Las imágenes en escala de grises solo cuentan con 1 canal, a diferencia del resto de codificaciones.

# Procesamiento de imagen

## Redimensionado

Permite cambiar el tamaño de la imagen.

```
img_resize = cv2.resize(img, (new_cols, new_rows))
```

Donde **img** es la matriz correspondiente a la imagen y **new\_cols**, **new\_rows** es el número de columnas, filas que debe tener la nueva imagen.

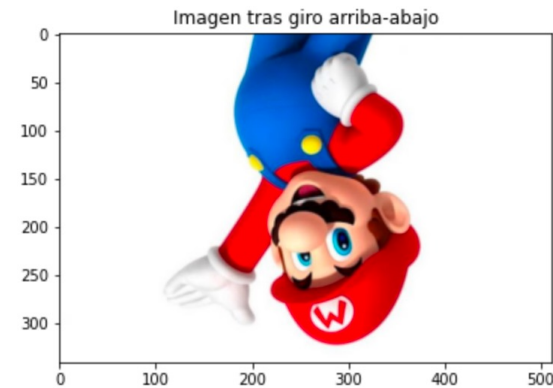
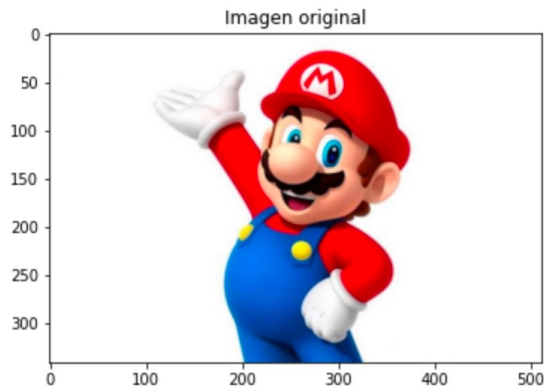
Cuando cambiamos el tamaño de la imagen a un tamaño superior, es recomendable especificar el tipo de transformación a hacer. Al redimensionar una imagen los más comunes serán **interpolación bilineal** (por defecto en OpenCV) o **bicúbica** para preservar la continuidad de los objetos. En algunas aplicaciones (e.g., segmentación) emplearemos el método del **vecino más cercano**.

```
img_resize = cv2.resize(img, (new_cols, new_rows), interpolation=cv2.INTER_CUBIC)
```

# Procesamiento de imagen

## Giro izquierda-derecha y arriba-abajo

```
# Haciendo uso de la librería numpy  
  
img_lr = np.fliplr(img) # Giro izquierda - derecha  
img_ud = np.flipud(img) # Giro arriba - abajo
```



# Procesamiento de imagen

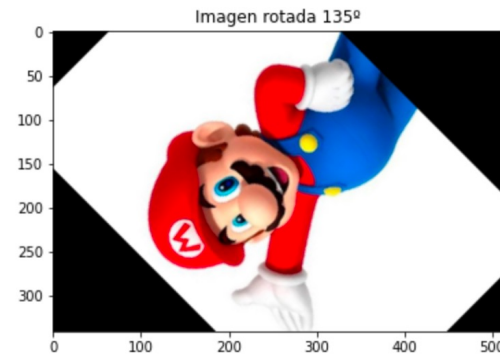
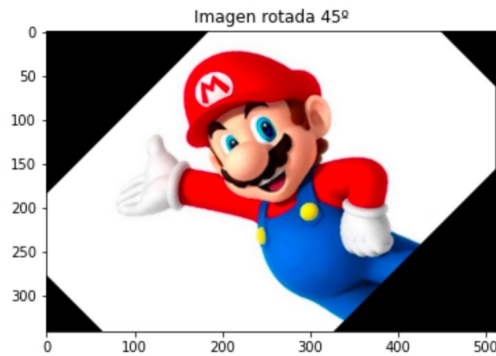
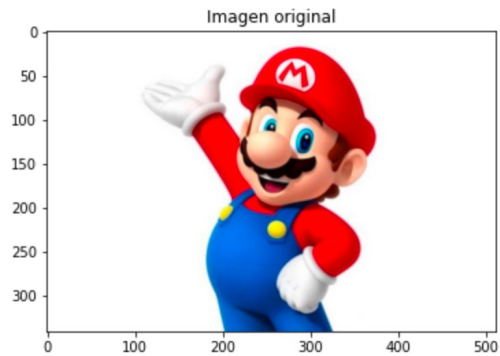
## Rotaciones

```
def rotate_image(image, angle, fill_color=(0,0,0)):  
  
    # Extraemos la altura y anchura de la imagen  
    height, width = image.shape[:2]  
  
    # Calculamos la matriz de rotación  
    # Empleamos el centro de la imagen como punto sobre el que realizar la rotación  
    center = (width // 2, height // 2)  
    rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)  
  
    # Rotamos la imagen  
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height), borderValue=fill_color)  
  
    return rotated_image
```

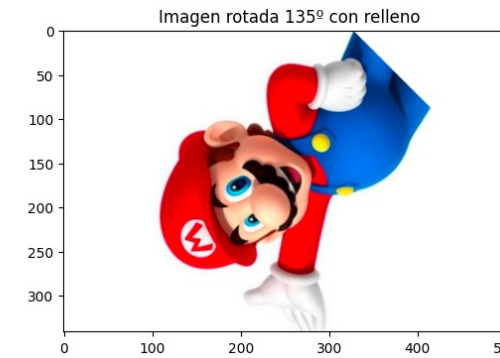
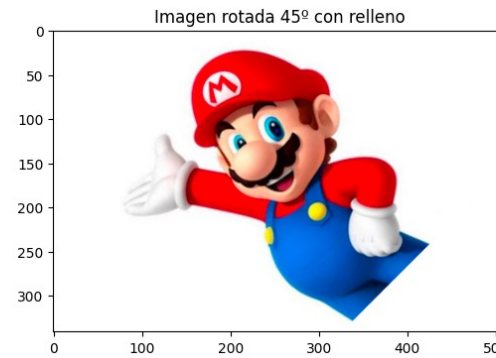
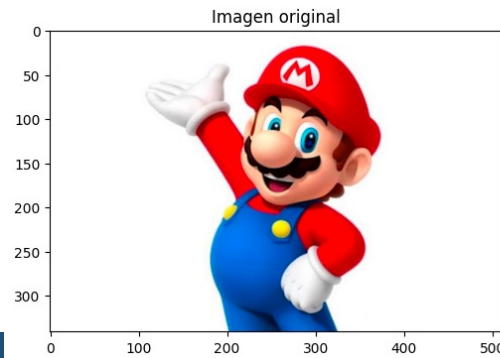
# Procesamiento de imagen

## Rotaciones

```
# Ejemplo: Rotación 45° y 135°  
img_rot_45 = rotate_image(img, 45)  
img_rot_135 = rotate_image(img, 135)
```



```
img_rot_45 = rotate_image(img, 45, fill_color=(255, 255, 255))  
img_rot_135 = rotate_image(img, 135, fill_color=(255, 255, 255))
```



# Procesamiento de imagen

## Zoom hacia dentro (positivo)

```
# Hacemos la imagen más grande para simular ese zoom y recortamos los bordes para tener una imagen del tamaño original
def zoom_in(image, factor):

    # Calculamos el nuevo tamaño de la imagen
    new_width = int(img.shape[1] * 2)
    new_height = int(img.shape[0] * 2)

    # Redimensionamos la imagen al tamaño nuevo
    zoomed_in_image = cv2.resize(img, (new_width, new_height),
                                  interpolation=cv2.INTER_CUBIC)

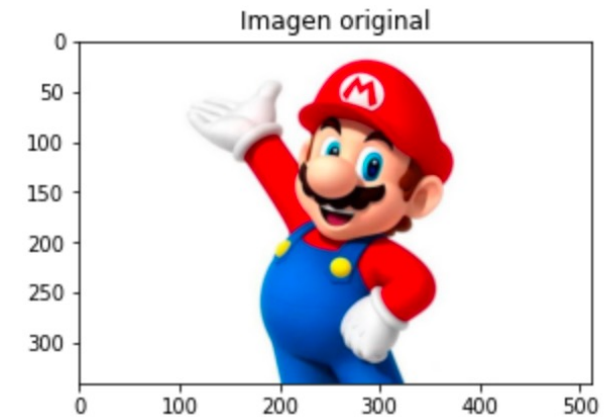
    # Hacemos el zoom con respecto al centro de la imagen
    center_rows = int(zoomed_in_image.shape[0] / 2)
    center_cols = int(zoomed_in_image.shape[1] / 2)

    # Recortamos la imagen al tamaño original alrededor del centro
    start_rows = center_rows - int(img.shape[0] / 2)
    end_rows = start_rows + img.shape[0]

    start_cols = center_cols - int(img.shape[1] / 2)
    end_cols = start_cols + img.shape[1]

    zoomed_in_image = zoomed_in_image[start_rows:end_rows, start_cols:end_cols]

    return zoomed_in_image
```



# Procesamiento de imagen

## Zoom hacia afuera (negativo)

```
def zoom_out(image, factor, fill_color=(0,0,0)):
```

```
    # Para hacer zoom hacia fuera lo que realmente estamos haciendo es añadir filas y columnas a los  
    # lados de la imagen para hacer la imagen más pequeña en relación
```

```
    rows_to_add = int(image.shape[0] * factor)
```

```
    cols_to_add = int(image.shape[1] * factor)
```

```
    # De todas las filas y columnas a añadir tenemos que distribuirlas entre el principio y final de la  
    # imagen para dejar la imagen centrada
```

```
    rows_to_add_ini = int(rows_to_add/2)
```

```
    rows_to_add_end = rows_to_add - rows_to_add_ini
```

```
    cols_to_add_ini = int(cols_to_add/2)
```

```
    cols_to_add_end = cols_to_add - cols_to_add_ini
```

```
    # Añadimos filas y columnas en el borde de la imagen (padding).
```

```
    # Primero definimos una matriz de ceros (vacía) que tenga el tamaño de la imagen original con el  
    # padding correspondiente
```

```
    img_zoom_negative = np.zeros(shape=(image.shape[0] + rows_to_add, image.shape[1] + cols_to_add,  
3)).astype('uint8')
```

```
    # A continuación a todos los píxeles le vamos a dar el valor del fondo de la imagen original.
```

```
    img_zoom_negative[:] = fill_color
```

```
    # Añadimos la imagen original en el centro de la imagen
```

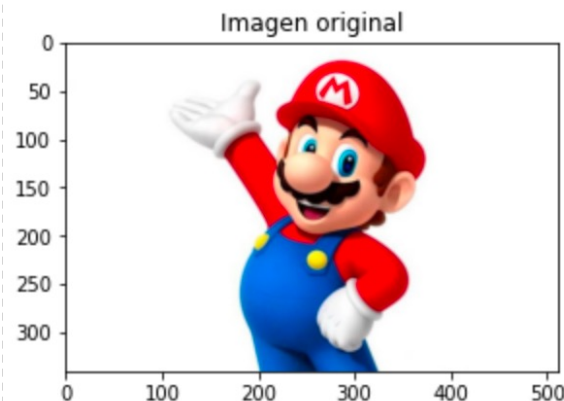
```
    img_zoom_negative[rows_to_add_ini:img_zoom_negative.shape[0] - rows_to_add_end,
```

```
    cols_to_add_ini:img_zoom_negative.shape[1] - cols_to_add_end, :] = image
```

```
    # Redimensionamos al tamaño original de la imagen
```

```
    img_zoom_negative = cv2.resize(img_zoom_negative, (image.shape[1], image.shape[0]),  
interpolation=cv2.INTER_CUBIC)
```

```
    return img_zoom_negative
```



# Procesamiento de imagen

## Zoom

```
# Finalmente vamos a crear una función de zoom
# En función del valor del factor aplicará zoom hacia dentro o hacia fuera

def zoom(image, factor, fill_color=(0,0,0)):
    if factor == 1:
        return image
    elif factor > 1:
        print('Zoom in')
        return zoom_in(image, factor)
    else:
        print('Zoom out')
        return zoom_out(image, factor, fill_color)
```

```
# Zoom hacia fuera
img_zoom_in = zoom(img, 2)
# Zoom hacia dentro
img_zoom_out = zoom(img, 0.5, fill_color=(255,255, 255))
```





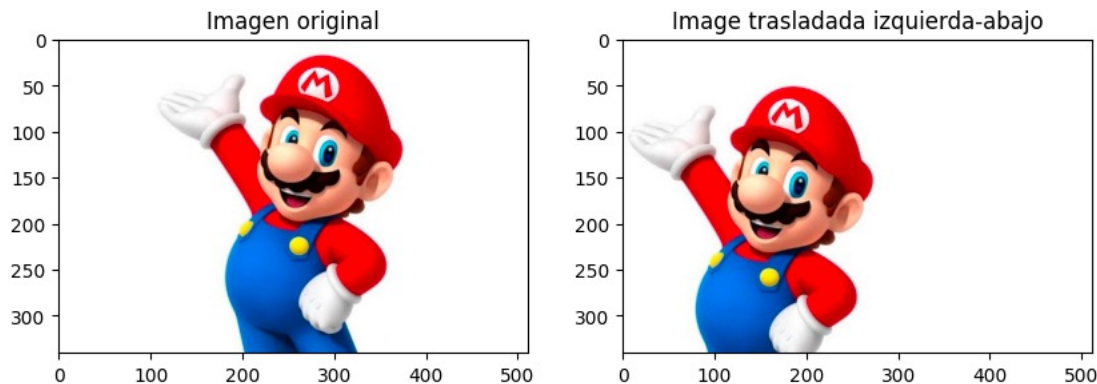
# Procesamiento de imagen

## Traslaciones

```
def traslation(image, rows_factor, cols_factor, fill_color=(0,0,0)):  
  
    # Calculamos el número de filas y columnas que se va a desplazar la imagen  
    rows_to_move = int(image.shape[0] * rows_factor)  
    cols_to_move = int(image.shape[1] * cols_factor)  
    # Calculamos matriz de traslación  
    T = np.float32([[1, 0, -cols_to_move], [0, 1, rows_to_move]])  
    img_traslated = cv2.warpAffine(img, T, (img.shape[1], img.shape[0]), borderValue=fill_color)  
  
    return img_traslated
```

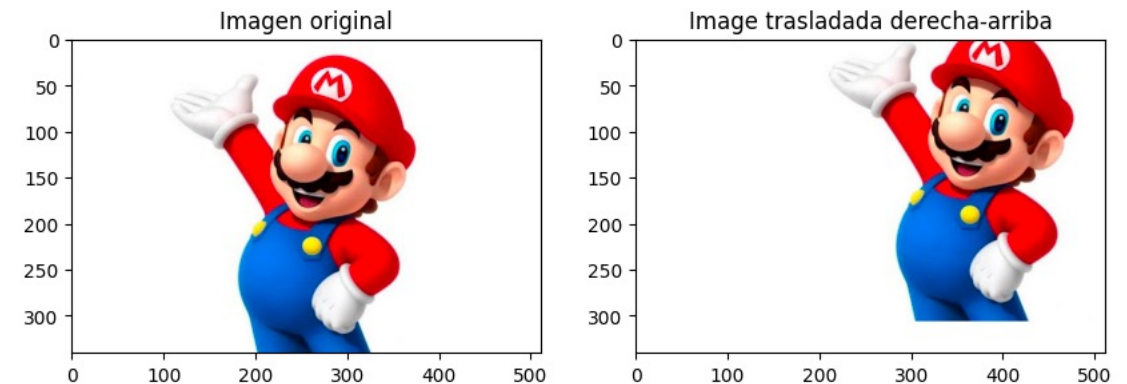
### Traslación izquierda - abajo

```
img_traslated = traslation(img, 0.1, 0.2, (255,255,255))
```



### Traslación derecha - arriba

```
img_traslated = traslation(img, -0.1, -0.2, (255,255,255))
```



# Procesamiento de imagen

## Traslaciones

Hay ocasiones en las que el relleno constante no es la mejor opción. Para esto existen otras opciones como el relleno en espejo o el uso del mismo valor de la fila y columna vecina.

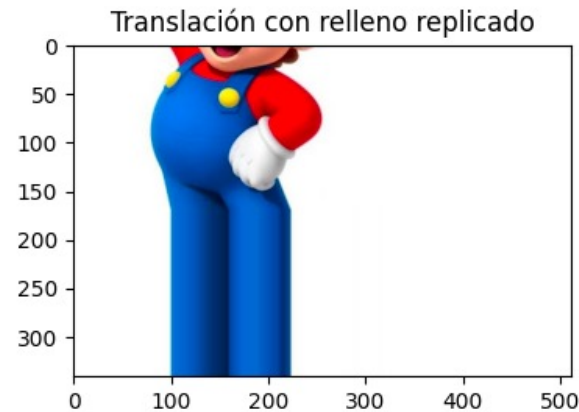
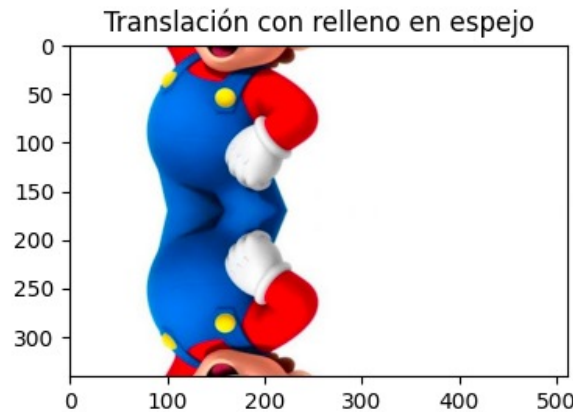
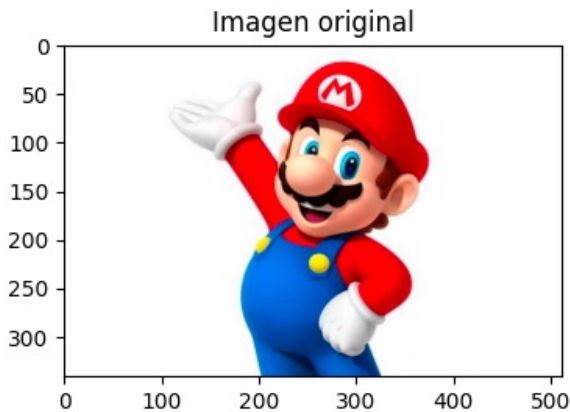
```
def traslation_border(image, rows_factor, cols_factor, border_mode=cv2.BORDER_REFLECT):  
    # Calculamos el número de filas y columnas que se va a desplazar la imagen  
    rows_to_move = int(image.shape[0] * rows_factor)  
    cols_to_move = int(image.shape[1] * cols_factor)  
    # Calculamos matriz de traslación  
    T = np.float32([[1, 0, -cols_to_move], [0, 1, rows_to_move]])  
    img_traslated = cv2.warpAffine(img, T, (img.shape[1], img.shape[0]), borderMode=border_mode)  
    return img_traslated
```

# Procesamiento de imagen

## Traslaciones

Hay ocasiones en las que el relleno constante no es la mejor opción. Para esto existen otras opciones como el rellenado en espejo o el uso del mismo valor de la fila y columna vecina.

```
# Relleno en espejo  
img_traslated_reflect = traslation_border(img, -0.5, 0.2, border_mode=cv2.BORDER_REFLECT)  
# Relleno en replicada  
img_traslated_replicate = traslation_border(img, -0.5, 0.2, border_mode=cv2.BORDER_REPLICATE)
```



# Procesamiento de imagen

## Cambio de brillo y contraste

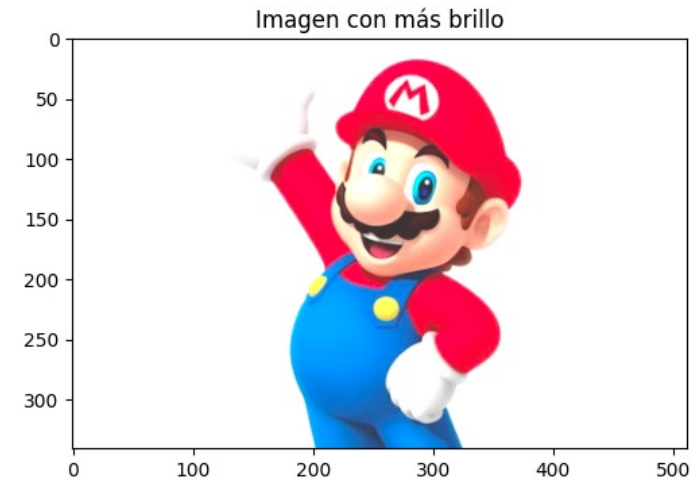
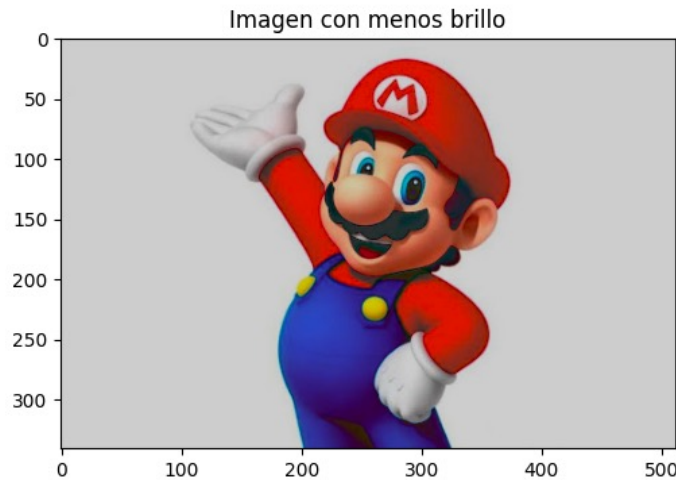
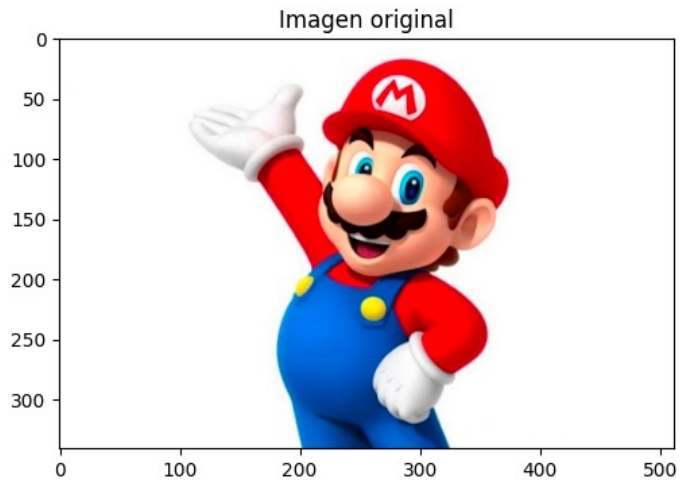
```
def brightness_contrast(image, brightness_control=0, contrast_control=1):  
    new_image = cv2.convertScaleAbs(image, alpha=contrast_control, beta=brightness_control)  
    return new_image
```

# Procesamiento de imagen

## Cambio de brillo y contraste

### Cambio de brillo

```
# Imagen con menos brillo (valor negativo)  
image_less_brightness = brightness_contrast(img, brightness_control=-50)  
# Imagen con más brillo (valor positivo)  
image_more_brightness = brightness_contrast(img, brightness_control=50)
```

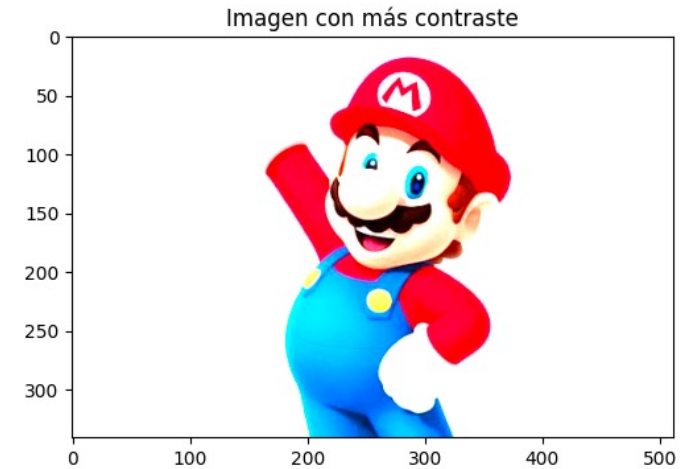
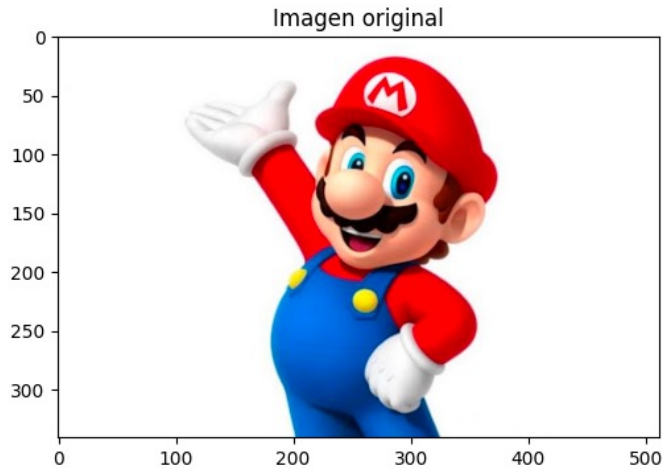


# Procesamiento de imagen

## Cambio de brillo y contraste

### Cambio de contraste

```
# Imagen con menos contraste (valor <1)  
image_less_contrast = brightness_contrast(img, contrast_control=0.5)  
# Imagen con más contraste (valor >1)  
image_more_contrast = brightness_contrast(img, contrast_control=2)
```

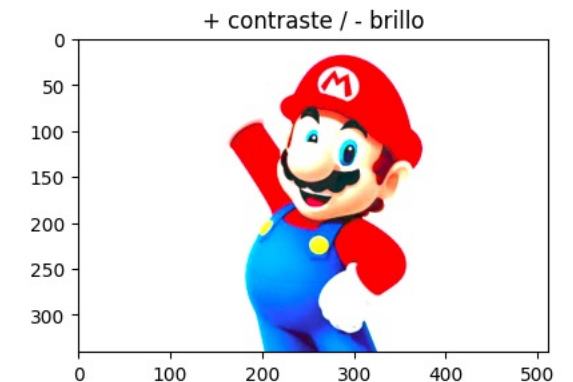
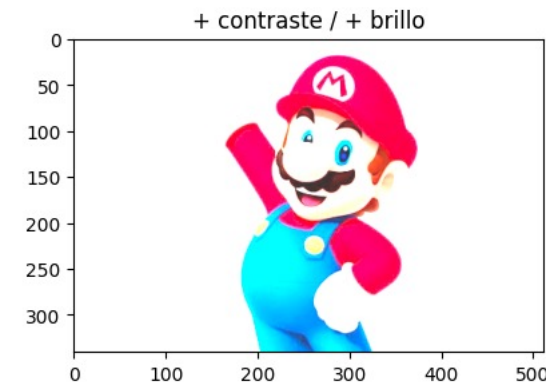
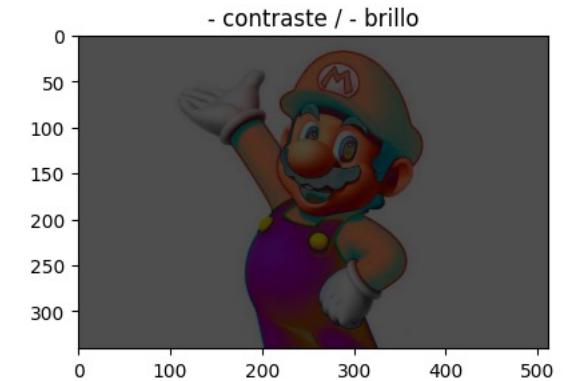
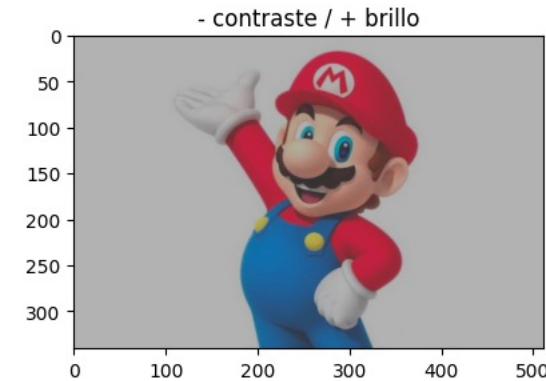


# Procesamiento de imagen

## Cambio de brillo y contraste

### Cambio de brillo y contraste simultáneo

```
# Imagen con menos contraste y más brillo
image_less_contrast_more_bright = brightness_contrast(img,
brightness_control=50, contrast_control=0.5)
# Imagen con menos contraste y menos brillo
image_less_contrast_less_bright = brightness_contrast(img,
brightness_control=-50, contrast_control=0.5)
# Imagen con más contraste y más brillo
image_more_contrast_more_bright = brightness_contrast(img,
brightness_control=50, contrast_control=2)
# Imagen con más contraste y menos brillo
image_more_contrast_less_bright = brightness_contrast(img,
brightness_control=-50, contrast_control=2)
```



# Preguntas

Q1. ¿Para qué creéis que puede ser de interés redimensionar la imagen en un modelo de redes neuronales convolucionales?

Q2. ¿Para qué creéis que se pueden emplear transformaciones como rotaciones, traslaciones, cambio de contraste, etc.?



# Procesamiento de imagen

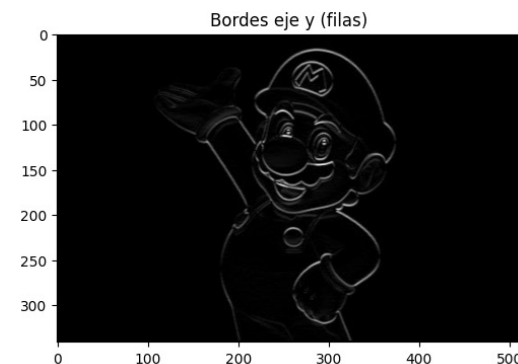
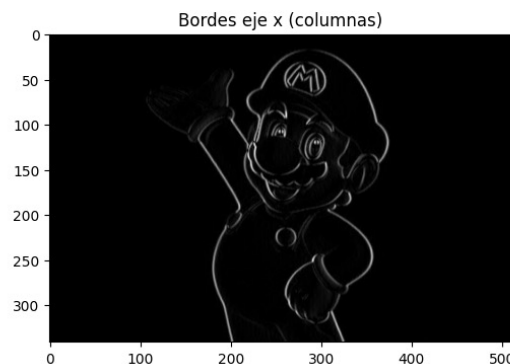
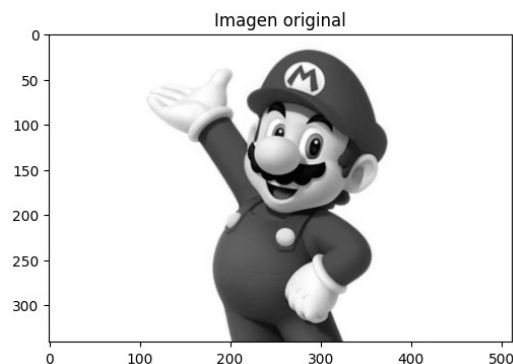
## Filtrado de imagen

En el campo de la visión por computador y el procesamiento de imagen es común emplear filtros como el filtro Sobel para la detección de bordes:

```
# Cargamos la imagen en escala de grises
image = cv2.imread('mariobros.jpeg', cv2.IMREAD_GRAYSCALE) # Convert to grayscale for edge detection

# Aplicamos el filtro Sobel en la dirección x y en la dirección y
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3) # Filtro Sobel en la dirección x (columnas)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3) # Filtro Sobel en la dirección y (filas)

# Nos quedamos con el valor absoluto para convertirlo en una matriz entre 0 y 1
sobel_x = np.abs(sobel_x)
sobel_y = np.abs(sobel_y)
```

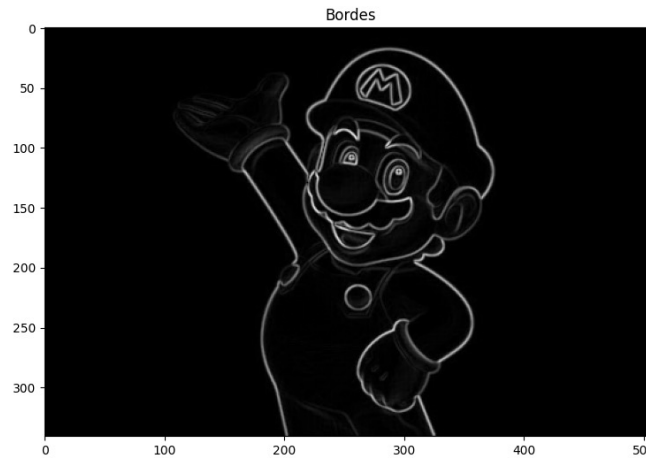
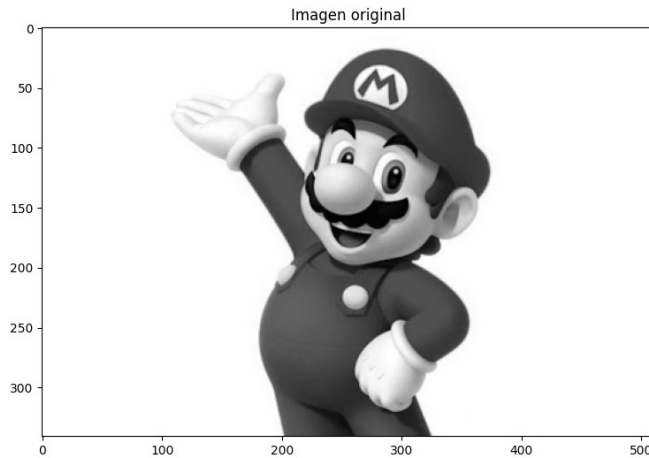


# Procesamiento de imagen

## Filtrado de imagen

También podemos generar una imagen con la combinación de ambas direcciones:

```
edge_image = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)
```



**Esto veremos que es de gran relevancia cuando trabajemos con redes neuronales convolucionales.**

# Ejercicio 1

Crea una función que reciba como entrada entrada la ruta a una imagen almacenada en el sistema de archivos. Esta función deberá:

1. Abrir la imagen y convertirla a un array de numpy.
  2. Redimensionarla a un tamaño de 1000x1500.
  3. A la imagen redimensionada aplicarle las siguientes transformaciones:
    - Giro izquierda-derecha
    - Rotación de 45º rellenando con valores (0, 0, 0).
    - Zoom hacia dentro con un factor de 1,4.
    - Desplazamiento de la imagen hacia la esquina inferior derecha, con un factor de 0,3 en ambas direcciones. Para rellenar filas y columnas generadas emplea la imagen espejo.
- Estas transformaciones se deberán aplicar de manera sucesiva, es decir, el resultado de la primera transformación será la entrada de la segunda transformación y así sucesivamente.

# Ejercicio 2

En la carpeta “imágenes” encontrarás un conjunto de 15 imágenes en formato JPG. El objetivo de este ejercicio es cargar las 15 imágenes y aplicarles la función generada en el Ejercicio 1 para aplicarles las transformaciones correspondientes.

Para ello:

1. Lista los archivos contenidos en el directorio “imágenes” de manera que esté contenidos en un vector.
2. Recorre uno a uno los archivos en un bucle y genera la ruta absoluta a cada archivo.
3. Para cada archivo llama a la función generada en el Ejercicio 1.

## NOTAS:

- La librería “os” facilita las acciones relacionadas con el trabajo con el sistema de archivos.
- El comando “os.listdir(<directorio>)” listará los archivos dentro de un directorio.
- El comando “os.path.join(<directorio1>, <directorio2>, ...)” permite generar los directorio sin tener que preocuparnos por la barra de separación (/ , \, \\) de cada sistema operativo.

# Ejercicio 3

Ahora vamos a crear una matriz donde vamos a concatenar el resultado de la transformación de cada una de las imágenes. Para ello, antes de empezar el bucle genera una matriz vacía de tamaño [n\_imágenes, n\_filas, n\_columnas, n\_canales], esto es, [15,1000,1500,3] (**PISTA: La función zeros de numpy te ayudara**). En cada iteración del bucle (i), almacena la imagen resultante en la posición correspondiente de la matriz que hemos creado [i, :, :, :].

Para comprobar que se está almacenando la imagen correctamente, en cada iteración, muestra cada imagen resultante por pantalla.

# **Tema 1. Procesamiento de imagen**

Ana Jiménez Pastor  
anjipas@gmail.com