

Tarea 5 - Preprocesado de Imágenes

Alumno: Irving Daniel Estrada López

Matrícula: 1739907

Introducción

En este notebook se llevarán a cabo técnicas de preprocesado de imágenes. El objetivo de esta práctica es familiarizarnos con técnicas las cuales nos ayuden a adecuar nuestro conjunto de datos de imágenes, para que en un futuro se lleve a cabo de manera efectiva un análisis de imágenes, ya sea su clasificación, detección de objetos, entre otros análisis. Las distintas técnicas de preprocesado se llevaron a cabo utilizando las siguientes tres librerías

1. Scikit-image
2. OpenCV
3. Pillow

En cada una de ellas se realizaron distintas operaciones básicas de manipulación de imágenes, mostrando su dicha transformación al final de cada operación utilizando matplotlib. Al final de este notebook se compararán dichas librerías, se hablará acerca de la facilidad de utilizar cada una de ellas, la utilidad de su documentación, entre otras cosas.

Carga de librerías

```
In [1]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

```
In [2]: filename = '/Users/irvingestrada/Documents/panda.jpeg'
```

scikit-image

Es una librería de procesamiento de imágenes que implementa algoritmos para su uso en aplicaciones de investigación, educación e industria. Se publica bajo la licencia liberal de código abierto Modified BSD, proporciona una API bien documentada en el lenguaje de programación Python y está desarrollado por un activo equipo internacional de colaboradores. Es importante destacar que, scikit-image es un paquete de procesamiento de imágenes que funciona con los arreglos de NumPy.

Carga de librerías de scikit-image

In [3]:

```
from skimage import io
import matplotlib.pyplot as plt
from skimage import data
from skimage import color
from skimage import img_as_float
from skimage import segmentation
from skimage.future import graph
from skimage import transform
from skimage import img_as_float
from skimage import exposure
from functools import partial
```

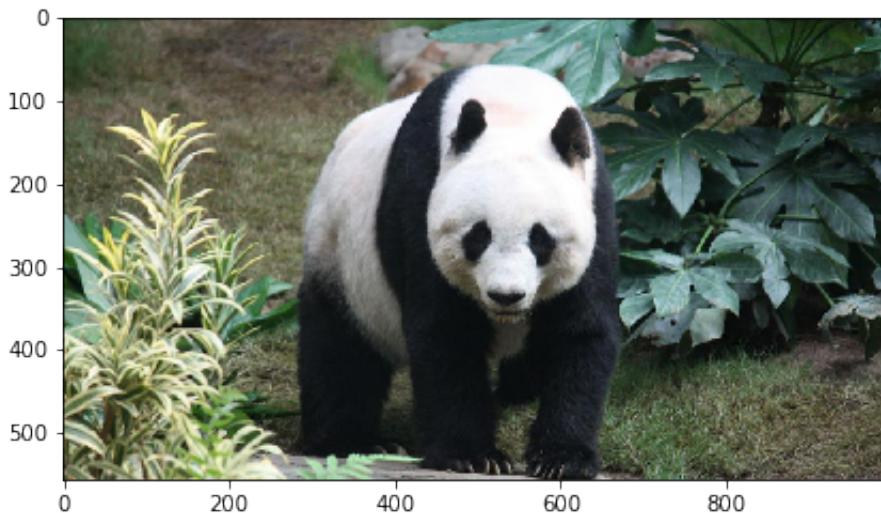
Carga de imagen con scikit-image

Para la carga de imágenes utilizaremos el paquete `io`.

io.imread: Carga una imagen de un archivo

In [4]:

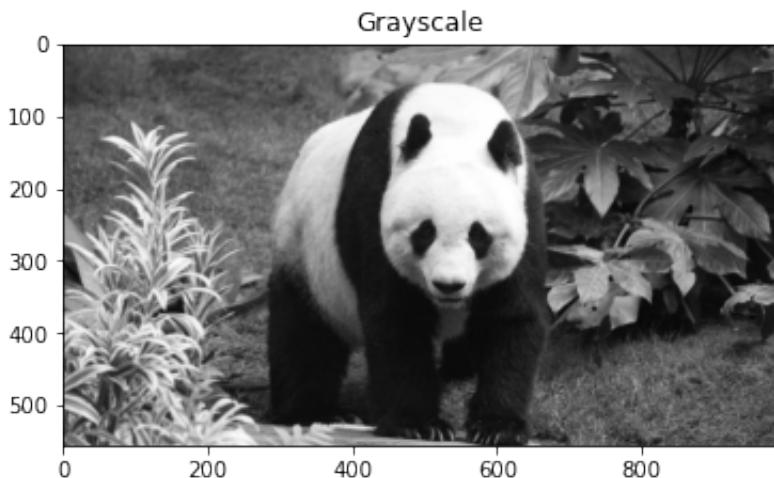
```
img = io.imread(filename)
io.imshow(img)
io.show()
```



color.rgb2gray: obtiene la iluminación de una imagen RGB. Trasformandola en escala de grises.

```
In [5]: img_gray = color.rgb2gray(img)
plt.imshow(img_gray, cmap='gray')
plt.title('Grayscale')
```

```
Out[5]: Text(0.5, 1.0, 'Grayscale')
```



color.rgb2hsv: Hace una conversión de RGB a HSV (Hue, Saturation, Value).

Por lo general, los objetos en las imágenes tienen colores distintos (hues) y luminosidad distinta, estas características se pueden usar para separar diferentes áreas de la imagen. En la representación RGB, el tono y la luminosidad se expresan como una combinación lineal de los canales R,G,B, mientras que corresponden a canales individuales de la imagen HSV.

```
In [6]: def colorize(image, hue, saturation=1):
    hsv = color.rgb2hsv(image)
    hsv[:, :, 1] = saturation
    hsv[:, :, 0] = hue
    return color.hsv2rgb(hsv)
```

```
In [7]: hue_rotations = np.linspace(0, 1, 6)
fig, axes = plt.subplots(nrows=2, ncols=3, sharex=True, sharey=True)
for ax, hue in zip(axes.flat, hue_rotations):
    # Turn down the saturation to give it that vintage look.
    tinted_image = colorize(img, hue, saturation=0.3)
    ax.imshow(tinted_image, vmin=0, vmax=1)
    ax.set_axis_off()
fig.tight_layout()
```



transform.EuclideanTransform: Transformación euclidiana, también conocida como transformación rígida.

La transformación euclidiana es una transformación rígida con parámetros de rotación y traslación. La transformación de similitud extiende la transformación euclidiana con un solo factor de escala.

```
In [8]: matrix = np.array([[np.cos(np.pi/12), -np.sin(np.pi/12), 100],
                      [np.sin(np.pi/12), np.cos(np.pi/12), -20],
                      [0, 0, 1]])
tform = transform.EuclideanTransform(matrix)
```

transform.warp: Deforma una imagen de acuerdo con una transformación de coordenadas dada.

```
In [9]: tf_img = transform.warp(img, tform.inverse)
```

```
In [10]: plt.imshow(tf_img)
```

Out[10]: <matplotlib.image.AxesImage at 0x14f2c2430>



scikit-image brinda fácil acceso a una poderosa variedad de funciones de procesamiento de imágenes. A continuación se enlistan sus paquetes

- **color:** conversión de espacio de color.
- **data:** imágenes de prueba y ejemplos de datos.
- **draw:** dibujos primitivos como líneas, textos, entre otros. Estos operan con arreglos de NumPy.
- **exposure:** ajustes de la intensidad de la imagen, histogramas de ecualización, entre otras cosas.
- **feature:** detección y extracción de características como análisis de texturas, esquinas, entre otras.
- **filter:** sharpening, encontrar bordes, rank filters, thresholding, entre otros.
- **graph:** operaciones con grafos como encontrar la ruta más corta.
- **io:** es un conjunto de distintas librerías para leer, guardar y desplegar imágenes y videos, como Pillow y Freimage.
- **measure:** medición de las propiedades de una imagen.
- **morphology:** operaciones morfológicas como opening o skeletonization.
- **novice:** simplifica la interfaz para propósitos educativos.
- **restoration:** Algoritmos de restauración como el algoritmo de deconvolution y denoising.
- **segmentation:** Partición de imágenes en múltiples regiones.
- **transform:** transformaciones geométricas y otras como rotación o la transformación de Radon.
- **viewer:** Una simple interfaz gráfica para visualizar .

CV2

OpenCV-Python es una librería de enlaces de Python diseñada para resolver problemas de visión computacional. OpenCV-Python hace uso de Numpy, que es una librería altamente optimizada para operaciones numéricas con una sintaxis de estilo MATLAB. Todas las estructuras de matriz OpenCV se convierten en Numpy arrays. Esto también facilita la integración con otras bibliotecas que usan Numpy, como SciPy y Matplotlib.

```
In [11]: import cv2
```

cv2.imread: Este método carga una imagen de un archivo en específico. En caso de que la imagen no se pueda leer retorna una matriz vacía.

```
In [12]: img = cv2.imread(filename)
```

img.shape: Retorna una tupla con el número de filas, columnas y canales.

Si una imagen está en escala de grises, la tupla devuelta contiene solo el número de filas y columnas, por lo que es un buen método para verificar si la imagen cargada está en escala de grises o en color.

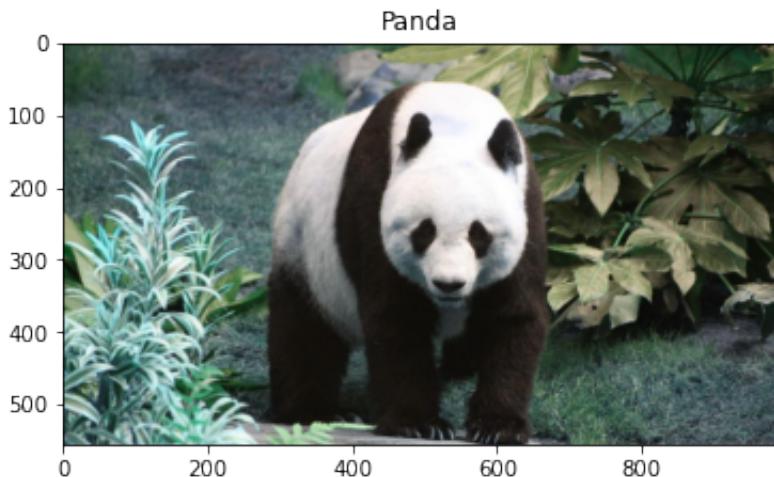
En la siguiente celda tenemos las dimensiones de la imagen, tenemos el ancho, el largo y la cantidad de canales que contiene la imagen. Por ultimo tenemos el tamaño de la imagen el cual consiste en la multiplicación del alto, el largo y sus canales

$$556 * 990 * 3 = 1651320$$

```
In [13]: rows,cols,chn = img.shape
print('Height: {}'.format(rows))
print('Width: {}'.format(cols))
print("Image Shape: {}".format(img.shape))
print("Image Size: {}".format(img.size))
```

```
Height: 556
Width: 990
Image Shape: (556, 990, 3)
Image Size: 1651320
```

```
In [14]: plt.imshow(img)
plt.title('Panda')
plt.show()
```



Filtros

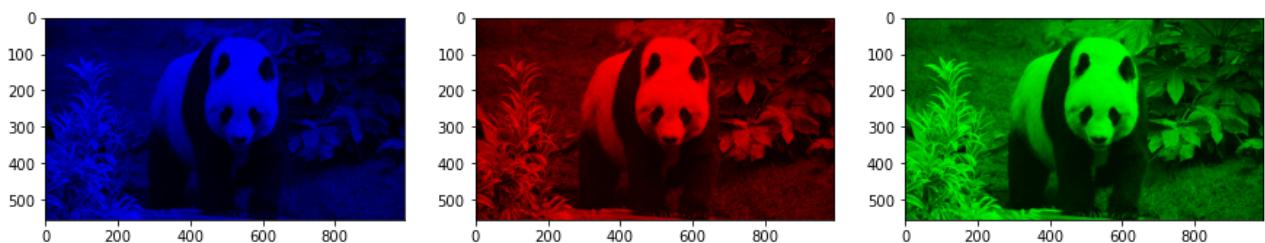
A continuación se lleva a cabo la obtención de cada uno de los canales RGB.

```
In [15]: image_copy = img.copy()  
image_copy2 = img.copy()  
image_copy3 = img.copy()
```

```
In [16]: image_copy[:, :, 0] = 0  
image_copy[:, :, 1] = 0  
  
image_copy2[:, :, 1] = 0  
image_copy2[:, :, 2] = 0  
  
image_copy3[:, :, 0] = 0  
image_copy3[:, :, 2] = 0
```

En cada uno de estos canales se destacan distintos aspectos de la imagen los cuales podemos utilizar a nuestro beneficio para obtener información valiosa.

```
In [17]: f, axes = plt.subplots(1, 3, figsize = (15, 15))  
list = [image_copy, image_copy2, image_copy3]  
i = 0  
for ax in axes:  
    ax.imshow(list[i])  
    i+=1
```

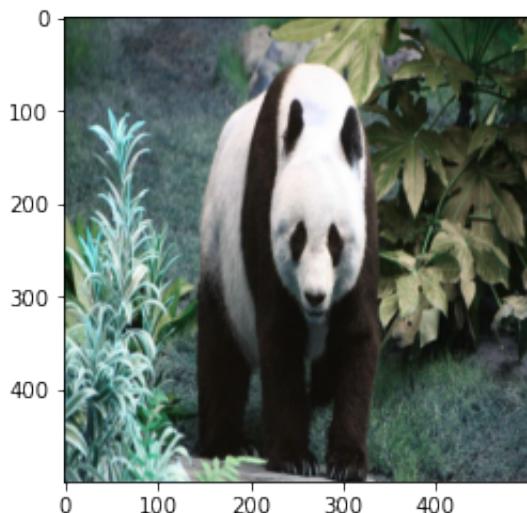


cv2.resize: Escala la imagen

Como podemos darnos cuenta en la siguiente celda, escalamos la imagen con las medidas de (500,500) no se pierde ninguna región de la imagen, sin embargo, su tamaño en píxeles se reduce.

```
In [18]: resized_pic = cv2.resize(img, (500,500))
plt.imshow(resized_pic)
```

```
Out[18]: <matplotlib.image.AxesImage at 0x299102310>
```

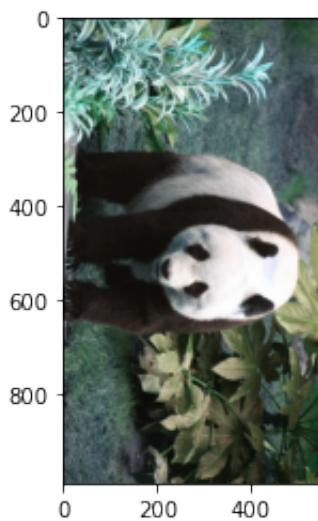


cv2.rotate: Rota la imagen, según la instrucción dada. Algunas de las instrucciones que podemos darle a rotate, son las siguientes:

- ROTATE_90_CLOCKWISE: Giro de 90 grados en el sentido de las agujas del reloj.
- ROTATE_180: Giro de 180 grados.
- ROTATE_90_COUNTERCLOCKWISE: Giro de 270 grados en el sentido de las agujas del reloj.

```
In [19]: image_90 = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
plt.imshow(image_90)
```

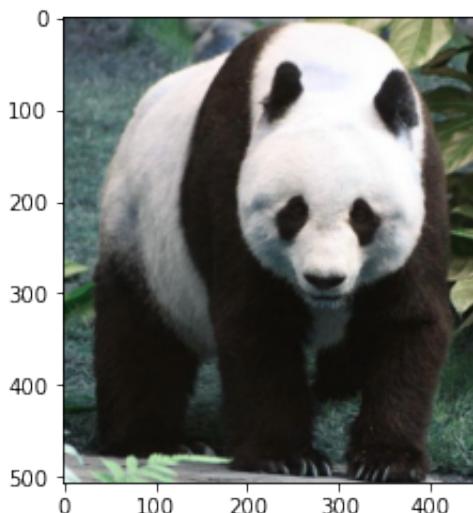
```
Out[19]: <matplotlib.image.AxesImage at 0x2991627c0>
```



Es posible trabajar la imagen como una matriz, permitiéndonos utilizar indexing y slicing como se utiliza con los tipos de datos list. De esta forma podemos obtener regiones las cuales nos interesan de la imagen, como podemos ver en la siguiente celda.

```
In [20]: crop = img[50:556, 250:700]  
plt.imshow(crop)
```

```
Out[20]: <matplotlib.image.AxesImage at 0x2991c5640>
```



threshold

- El método de segmentación más simple.
- Ejemplo de aplicación: Separar regiones de una imagen correspondientes a los objetos que queremos analizar. Esta separación se basa en la variación de intensidad entre los píxeles del objeto y los píxeles del fondo.
- Para diferenciar los píxeles que nos interesan del resto, realizamos una comparación del valor de intensidad de cada píxel con respecto a un umbral este depende del problema a resolver.
- Una vez que hayamos separado correctamente los píxeles importantes, podemos configurarlos con un valor determinado para identificarlos es decir, podemos asignarles un valor de 0 (negro), 255 (blanco) o cualquier valor que se adapte a sus necesidades.

A continuación tenemos los **cv2.threshold** con sus respectivos intervalos, los cuales cambian nuestra imagen dejandonos destacar puntos importantes, como la luz, los bordes, entre otras cosas.

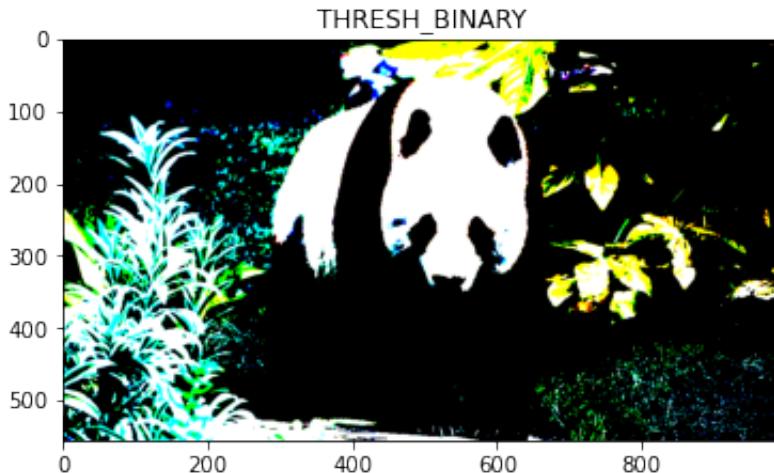
```
In [21]: ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
```

THRESH_BINARY

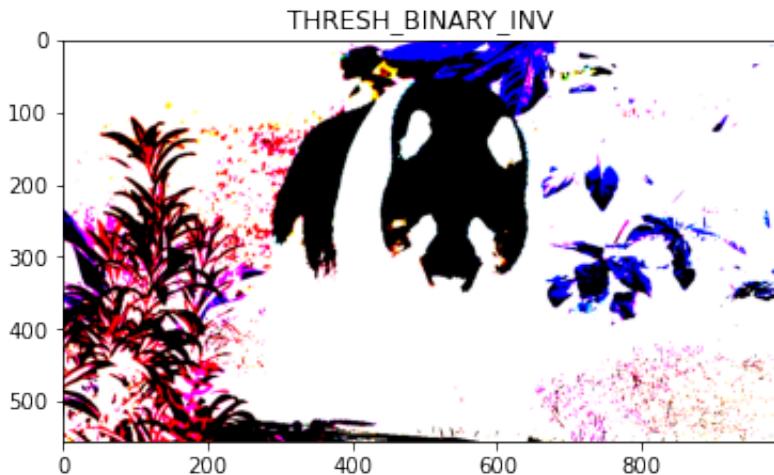
$$dst(x,y) = \begin{cases} maxval & src(x,y) > thresh \\ 0 & otherwise \end{cases}$$

```
In [22]: plt.imshow(thresh1)
plt.title('THRESH_BINARY')
plt.show()
```

**THRESH_BINARY_INV**

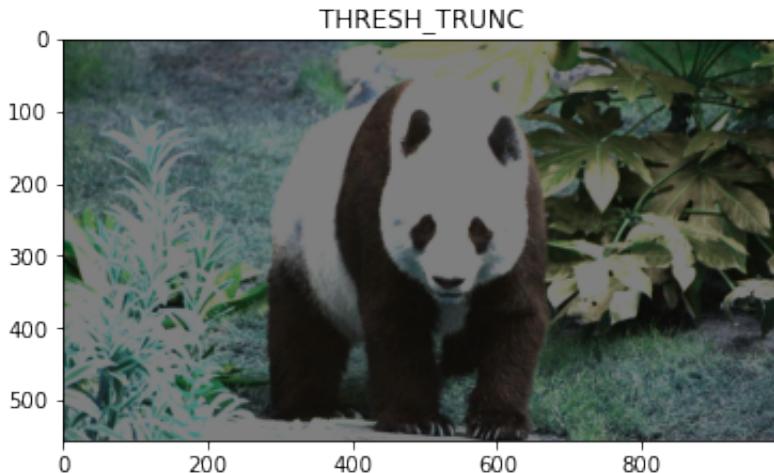
$$dst(x, y) = \begin{cases} 0 & src(x, y) > thresh \\ maxval & otherwise \end{cases}$$

```
In [23]: plt.imshow(thresh2)
plt.title('THRESH_BINARY_INV')
plt.show()
```

**THRESH_TRUNC**

$$dst(x, y) = \begin{cases} threshold & src(x, y) > thresh \\ src(x, y) & otherwise \end{cases}$$

```
In [24]: plt.imshow(thresh3)
plt.title('THRESH_TRUNC')
plt.show()
```



THRESH_TOZERO

$$dst(x, y) = \begin{cases} src(x, y) & src(x, y) > thresh \\ 0 & otherwise \end{cases}$$

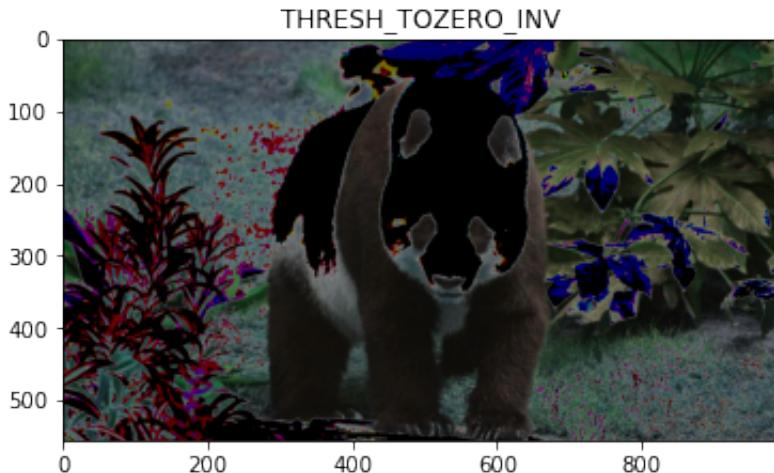
```
In [25]: plt.imshow(thresh4)
plt.title('THRESH_TOZERO')
plt.show()
```



THRESH_TOZERO_INV

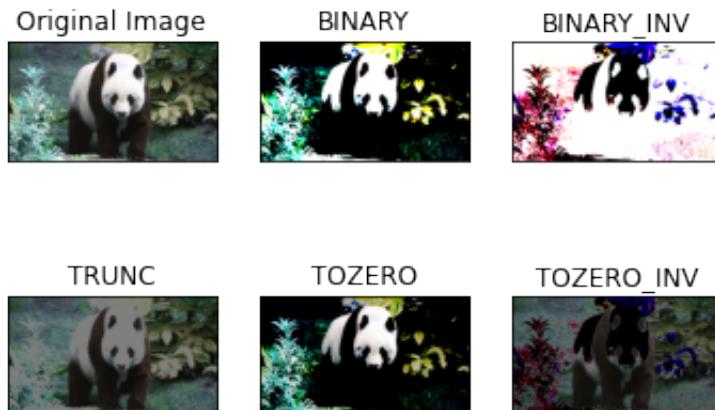
$$dst(x, y) = \begin{cases} 0 & src(x, y) > thresh \\ src(x, y) & otherwise \end{cases}$$

```
In [26]: plt.imshow(thresh5)
plt.title('THRESH_TOZERO_INV')
plt.show()
```



Reiterando lo anteriormente mencionado y teniendo todas las imágenes en una sola gráfica en la siguiente celda, podemos darnos cuenta que cada uno de los umbrales nos permiten identificar características de la imagen. Por ejemplo, el umbral BINARY nos permite contemplar mejor la iluminación, mandando a negro los colores que son oscuros. Cada uno de estos umbrales nos brindan información valiosa dependiendo de la situación

```
In [27]: for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray', vmin=0, vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



```
In [28]: img2 = cv2.imread(filename, 0)
```

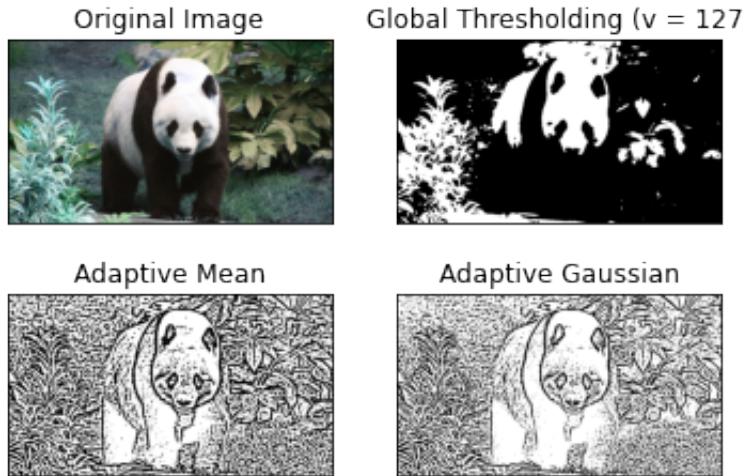
Dentro de esta sección de Threshold OpenCV nos brinda una función de un threshold adaptativo las cuales son:

- ADAPTIVE_THRESH_MEAN_C
- ADAPTIVE_THRESH_GAUSSIAN_C

Estos son útiles cuando nuestras imágenes tienen diferentes condiciones como su iluminación, entre otras cosas. El algoritmo determina el umbral para un píxel en función de una pequeña región a su alrededor. Entonces obtenemos diferentes umbrales para diferentes regiones de la misma imagen, lo que da mejores resultados para imágenes con iluminación variable.

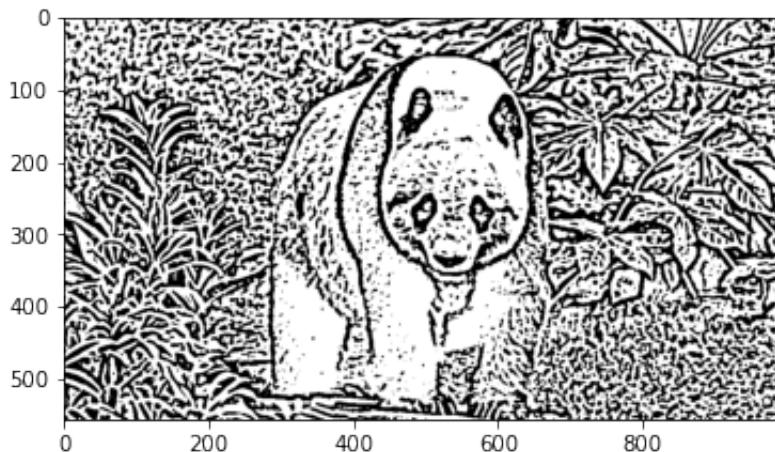
```
In [29]: img2 = cv2.medianBlur(img2,5)
ret,th1 = cv2.threshold(img2,127,255, cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img2,255, cv2.ADAPTIVE_THRESH_MEAN_C,\n
                           cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img2,255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\n
                           cv2.THRESH_BINARY,11,2)
titles = ['Original Image', 'Global Thresholding (v = 127)',\n
          'Adaptive Mean', 'Adaptive Gaussian']
images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
```



```
In [30]: plt.imshow(th2,'gray')
```

```
Out[30]: <matplotlib.image.AxesImage at 0x299682460>
```



```
In [31]: plt.imshow(th3, 'gray')
```

```
Out[31]: <matplotlib.image.AxesImage at 0x2996e7400>
```

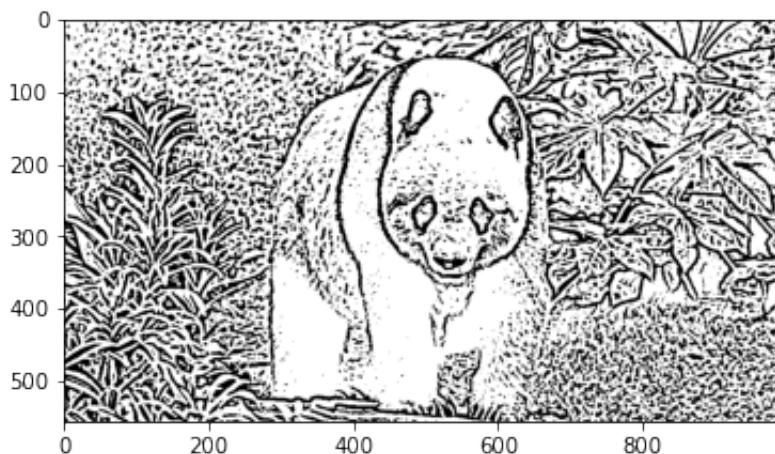


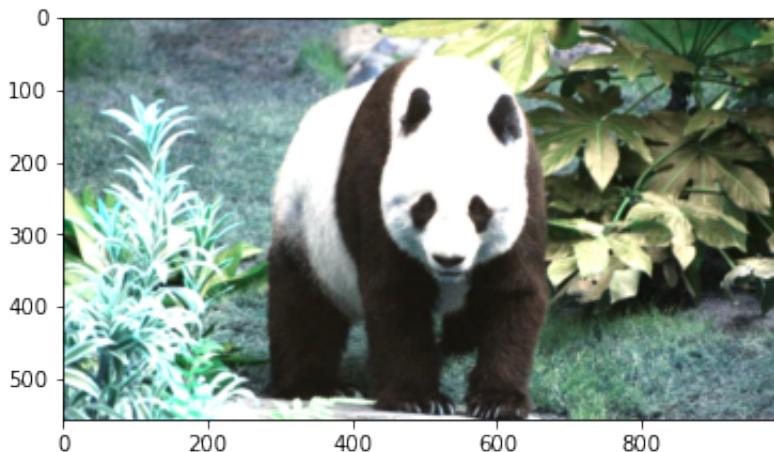
Image Blurring (Suavización de imagen)

cv2.filter2D: Convoluciona una imagen con el kernel.

```
In [32]: kernel = np.ones(shape=(3,3), dtype = np.float32)/6.07
```

```
In [33]: dst = cv2.filter2D(img,-1,np.array(kernel))
plt.imshow(dst)
```

```
Out[33]: <matplotlib.image.AxesImage at 0x29974a280>
```



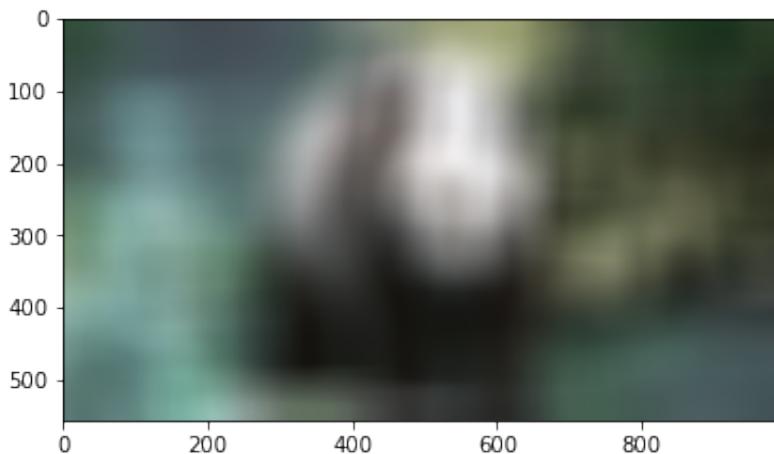
cv2.blur: Desenfoca una imagen utilizando el filtro de caja normalizado.

Esto se hace convolucionando una imagen con un filtro de caja normalizado.

Simplemente toma el promedio de todos los píxeles debajo del área del kernel y reemplaza el elemento central.

```
In [34]: blur = cv2.blur(img, ksize = (100,100))
plt.imshow(blur)
```

```
Out[34]: <matplotlib.image.AxesImage at 0x2997a4c70>
```



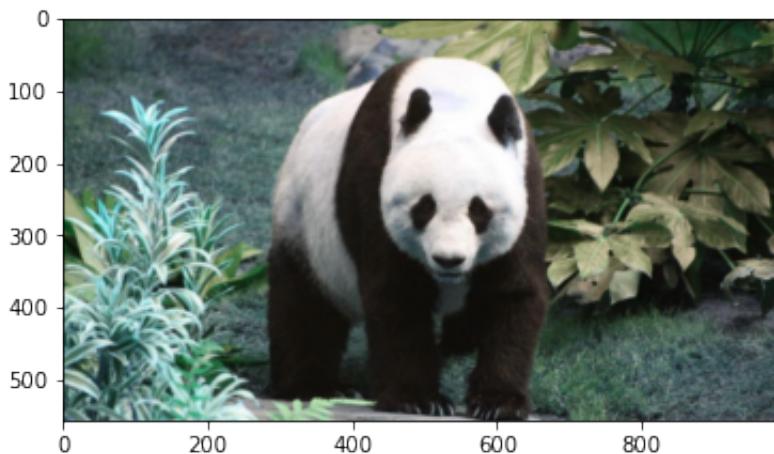
cv2.GaussianBlur: Desenfoca una imagen utilizando un filtro gaussiano.

En este método, en lugar de un filtro de caja, se utiliza un kernel gaussiano. El desenfoque gaussiano es muy eficaz para eliminar el ruido gaussiano de una imagen.

NOTA: El ruido Gaussiano se encuentra asociado con la radiación electromagnética.

```
In [35]: gaussian_blur = cv2.GaussianBlur(img, (3,3), 0)
plt.imshow(gaussian_blur)
```

Out [35]: <matplotlib.image.AxesImage at 0x29a0189a0>



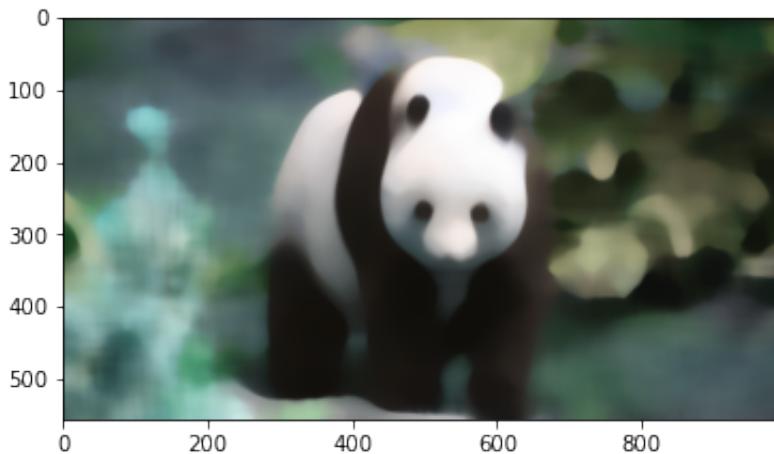
cv2.medianBlur: Desenfoca una imagen usando el median filter.

Toma la mediana de todos los píxeles debajo del área del kernel y el elemento central se reemplaza con este valor de la mediana. Esto es muy efectivo contra el ruido de salt-and-pepper en una imagen.

Curiosamente, en los filtros anteriores, el elemento central es un valor recién calculado que puede ser un valor de píxel en la imagen o un valor nuevo. Pero en el median blur, el elemento central siempre se reemplaza por algún valor de píxel en la imagen. Reduce el ruido de manera efectiva. Su tamaño de núcleo debe ser un número entero impar positivo.

In [36]: `median.blur = cv2.medianBlur(img, 41)
plt.imshow(median.blur)`

Out [36]: <matplotlib.image.AxesImage at 0x29a081430>



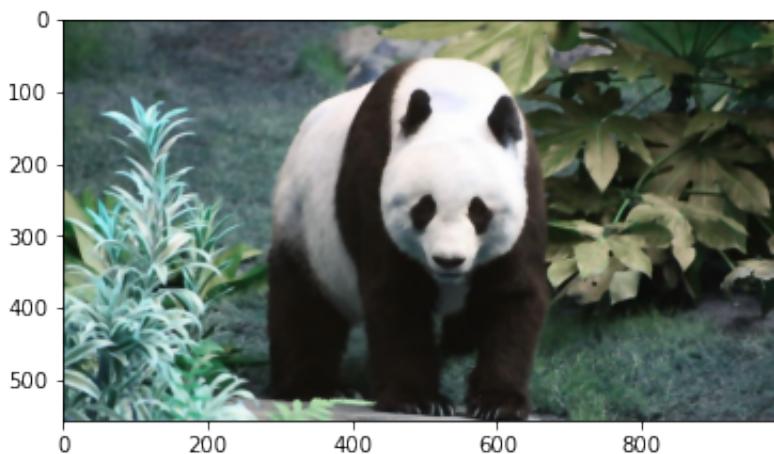
cv2.bilateralFilter: Aplica el filtro bilateral a una imagen.

Es altamente efectivo en la eliminación de ruido mientras mantiene los bordes. Pero el funcionamiento es más lento en comparación con otros filtros.

El filtrado bilateral también toma un filtro gaussiano en el espacio, pero usa un filtro gaussiano más, el cual es una función de la diferencia de píxeles. La función gaussiana del espacio asegura que solo los píxeles cercanos se consideren para el desenfoque, mientras que la función gaussiana de diferencia de intensidad asegura que solo aquellos píxeles con intensidades similares al píxel central se consideren para el desenfoque. Por lo tanto, conserva los bordes, ya que los píxeles en los bordes tendrán una gran variación de intensidad.

```
In [37]: bilateral_blur = cv2.bilateralFilter(img,9,75,75)
plt.imshow(bilateral_blur)
```

```
Out[37]: <matplotlib.image.AxesImage at 0x29a0e6130>
```



```
In [ ]:
```

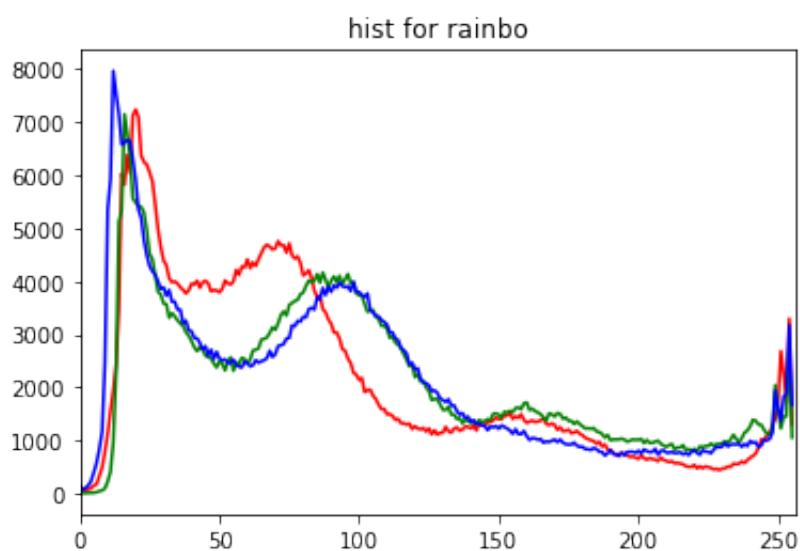
A continuación tenemos la gráfica de cada uno de los canales de nuestra imagen RGB, en la cual podemos ilustrar el comportamiento de los valores de cada uno de los canales.

```
In [38]: color = ('r','g','b')

for i,col in enumerate(color):
    hist = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(hist,color = col)
    plt.xlim([0,256])

#plt.ylim()
plt.title('hist for rainbo')
```

```
Out[38]: Text(0.5, 1.0, 'hist for rainbo')
```



```
In [ ]:
```

Python Image Library (Pillow/PIL)

Python Imaging Library agrega capacidades de procesamiento de imágenes a su intérprete de Python. Esta librería proporciona una amplia compatibilidad con formatos de archivo, una representación interna eficiente y capacidades de procesamiento de imágenes bastante potentes. La librería de imágenes principal está diseñada para un acceso rápido a los datos almacenados en unos pocos formatos de píxeles básicos. También debería proporcionar una base sólida para un procesamiento general de imágenes.

El módulo Image proporciona una clase con el mismo nombre que se utiliza para representar una imagen PIL. El módulo también proporciona una serie de funciones de fábrica, incluidas funciones para cargar imágenes desde archivos y para crear nuevas imágenes.

```
In [39]: from PIL import Image, ImageFilter
```

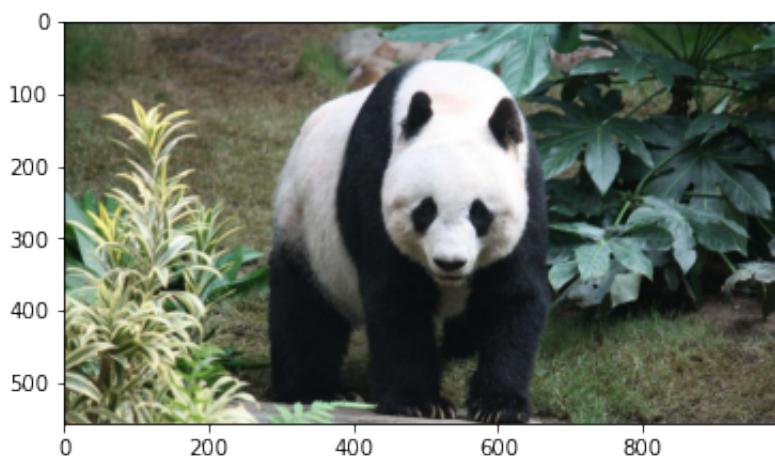
En la siguiente celda tenemos como se lee una imagen con PIL.Image, se trabaja como si fuera un archivo el cual cargamos.

```
In [40]: with Image.open(filename) as img:  
    img.load()
```

Continuaremos utilizando matplotlib para mostrar nuestras imágenes, para que de esta manera nos muestre las reglas en los márgenes de la imagen

```
In [41]: plt.imshow(img)
```

```
Out[41]: <matplotlib.image.AxesImage at 0x29a1c5730>
```



Al igual que en las demás librerías podemos obtener detalles de la imagen.

```
In [42]: print('Format: {}'.format(img.format))  
print('Shape: {}'.format(img.size))  
print('Mode: {}'.format(img.mode))
```

```
Format: JPEG  
Shape: (990, 556)  
Mode: RGB
```

crop: Obtiene una región de la imagen

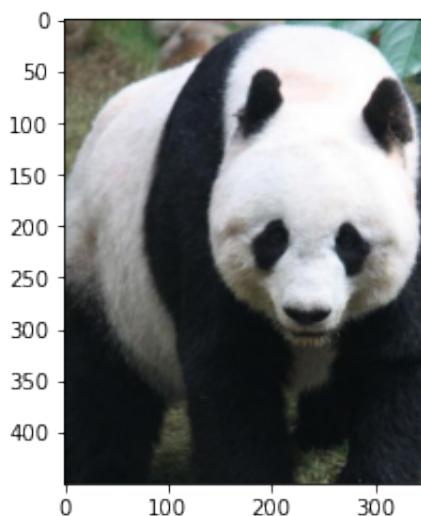
La clase contiene métodos que le permiten manipular regiones dentro de una imagen.
Para extraer un subrectángulo de una imagen.

```
In [43]: cropped_img = img.crop((300, 50, 650, 500))  
cropped_img.size
```

```
Out[43]: (350, 450)
```

```
In [44]: plt.imshow(cropped_img)
```

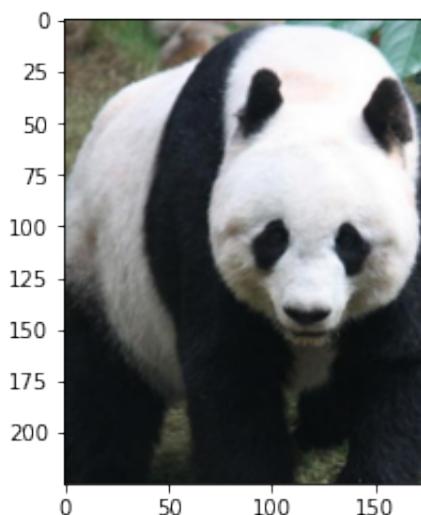
```
Out[44]: <matplotlib.image.AxesImage at 0x29a229730>
```



resize: Nos permite escalar la imagen según lo deseado. Toma una tupla dando el nuevo tamaño

```
In [45]: low_res_img = cropped_img.resize((cropped_img.width // 2, cropped_img.height // 2))  
plt.imshow(low_res_img)
```

```
Out[45]: <matplotlib.image.AxesImage at 0x29a294f40>
```

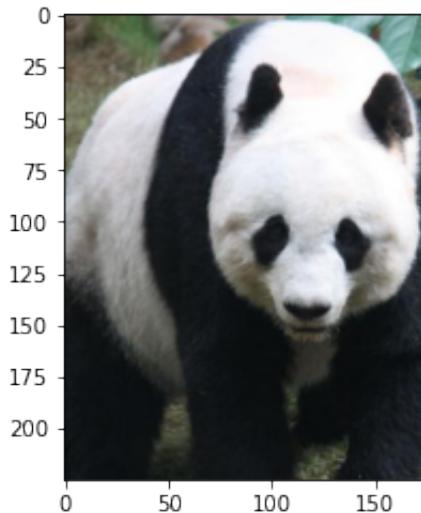


reduce: Reduce la imagen al parámetro dado.

En este caso el argumento que le damos es 2, reduciendo a la mitad la imagen. Es parecido a resize con la diferencia que con resize tenemos más control de los parámetros.

```
In [46]: low_res_img = cropped_img.reduce(2)  
plt.imshow(low_res_img)
```

```
Out[46]: <matplotlib.image.AxesImage at 0x29a312130>
```

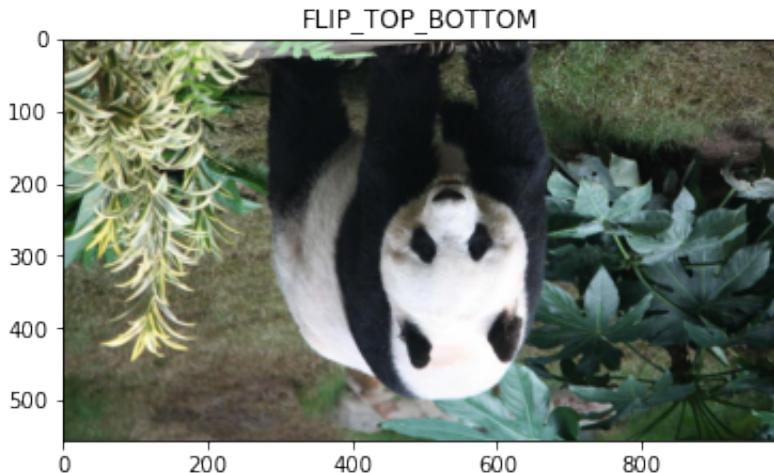


transpose: traspone la imagen según el argumento dado, los posibles parametros son los siguientes:

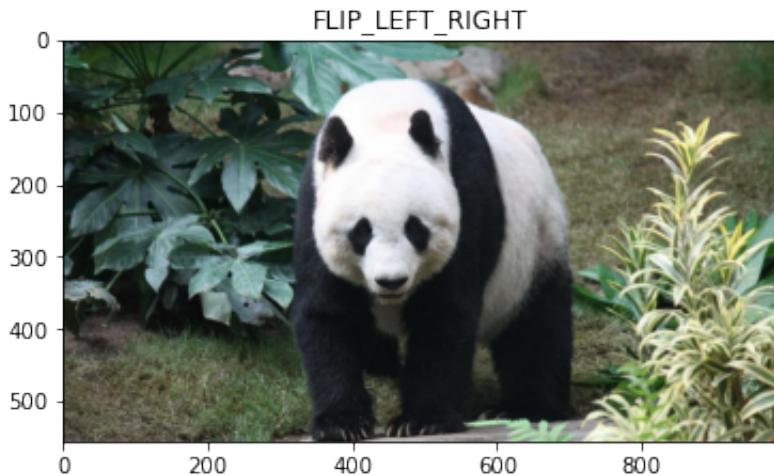
- FLIP_LEFT_RIGHT = 0
- FLIP_TOP_BOTTOM = 1
- ROTATE_180 = 3
- ROTATE_270 = 4
- ROTATE_90 = 2
- TRANSPOSE = 5
- TRANSVERSE = 6

Es posible escribir tanto el texto como el número.

```
In [47]: converted_img = img.transpose(Image.FLIP_TOP_BOTTOM)  
plt.imshow(converted_img)  
plt.title('FLIP_TOP_BOTTOM')  
plt.show(converted_img)
```



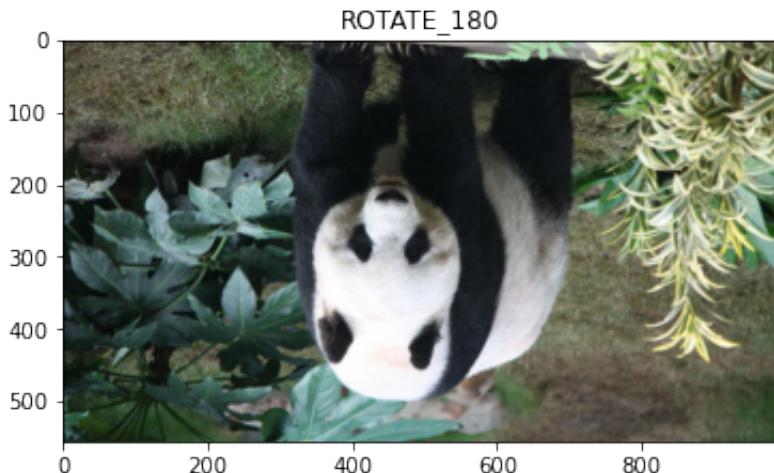
```
In [48]: converted_img = img.transpose(Image.FLIP_LEFT_RIGHT)
plt.imshow(converted_img)
plt.title('FLIP_LEFT_RIGHT')
plt.show(converted_img)
```



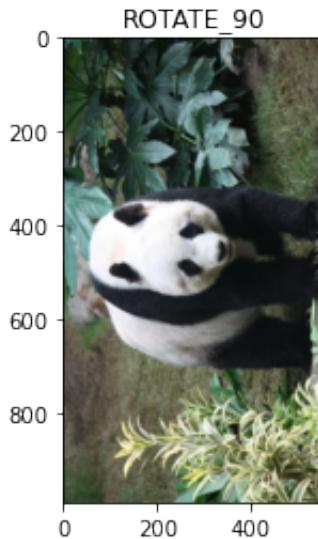
```
In [49]: converted_img = img.transpose(Image.ROTATE_90)
plt.imshow(converted_img)
plt.title('ROTATE_90')
plt.show(converted_img)
```



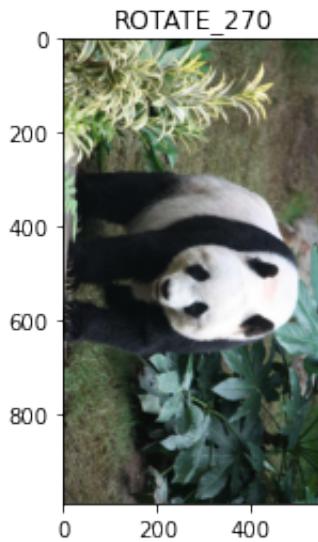
```
In [50]: converted_img = img.transpose(Image.ROTATE_180)
plt.imshow(converted_img)
plt.title('ROTATE_180')
plt.show(converted_img)
```



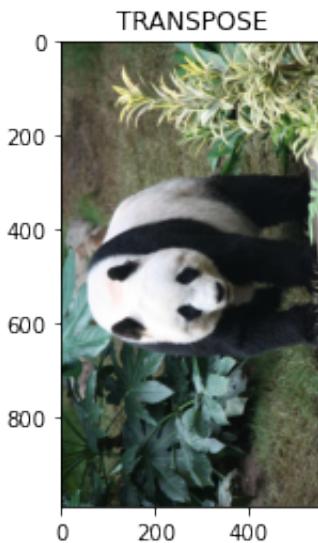
```
In [51]: converted_img = img.transpose(Image.ROTATE_90)
plt.imshow(converted_img)
plt.title('ROTATE_90')
plt.show(converted_img)
```



```
In [52]: converted_img = img.transpose(Image.ROTATE_270)
plt.imshow(converted_img)
plt.title('ROTATE_270')
plt.show(converted_img)
```



```
In [53]: converted_img = img.transpose(Image.TRANSPOSE)
plt.imshow(converted_img)
plt.title('TRANSPOSE')
plt.show(converted_img)
```



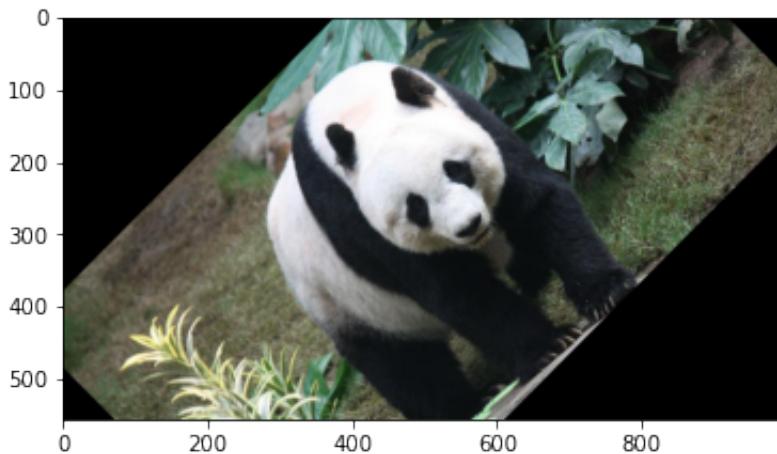
```
In [54]: converted_img = img.transpose(Image.TRANSVERSE)
plt.imshow(converted_img)
plt.title('TRANSVERSE')
plt.show(converted_img)
```



rotate: Rota la imagen segun los grados dados.

```
In [55]: rotated_img = img.rotate(45)
plt.imshow(rotated_img)
```

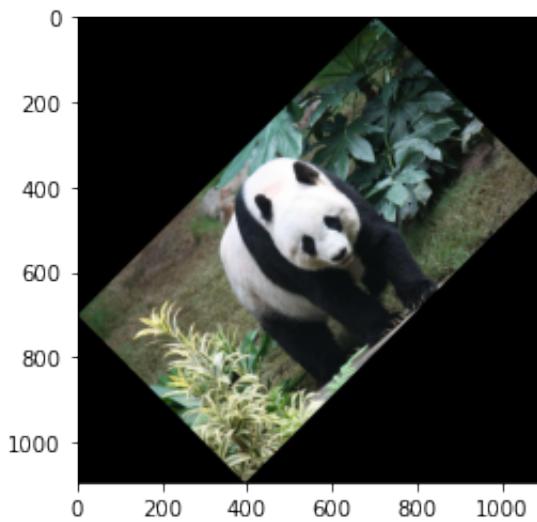
```
Out[55]: <matplotlib.image.AxesImage at 0x29a3f8b80>
```



El atributo `expand` ayuda a que la imagen original se centre en la transformación.

```
In [56]: rotated_img = img.rotate(45, expand=True)
plt.imshow(rotated_img)
```

```
Out[56]: <matplotlib.image.AxesImage at 0x29a45b850>
```



convert: Devuelve una copia convertida de esta imagen.

Para el modo "P", este método traduce píxeles a través de la paleta. Si se omite el modo, se elige un modo para que toda la información de la imagen y la paleta se puedan representar sin paleta.

```
In [57]: cmyk_img = img.convert("CMYK")
gray_img = img.convert("L") # Grayscale #cmyk_img
gray_img
```

Out [57]:



split: Divide esta imagen en bandas individuales. Este método devuelve una tupla de bandas de imágenes individuales de una imagen. Por ejemplo, dividir una imagen "RGB" crea tres nuevas imágenes, cada una de las cuales contiene una copia de una de las bandas originales (rojo, verde, azul).

In [58]:

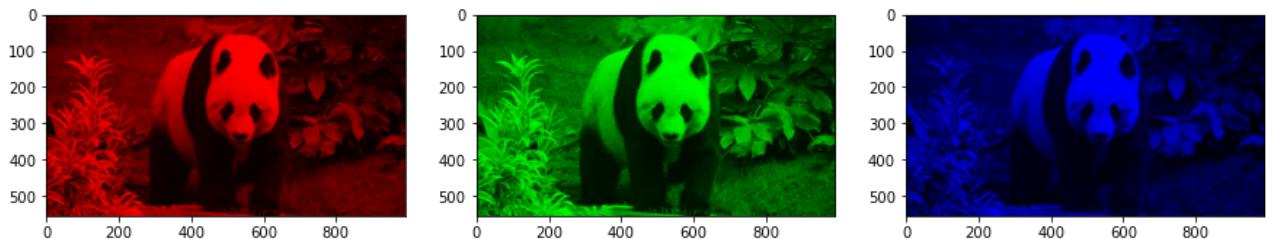
```
red, green, blue = img.split()
```

In [59]:

```
zeroed_band = red.point(lambda _: 0)
red_merge = Image.merge("RGB", (red, zeroed_band, zeroed_band))
green_merge = Image.merge("RGB", (zeroed_band, green, zeroed_band))
blue_merge = Image.merge("RGB", (zeroed_band, zeroed_band, blue))
```

In [60]:

```
f, axes = plt.subplots(1,3, figsize = (15,15))
list = [red_merge,green_merge,blue_merge]
i = 0
for ax in axes:
    ax.imshow(list[i])
    i+=1
```



filter: Filtra esta imagen usando el filtro dado.

Los posibles parametros de filter son:

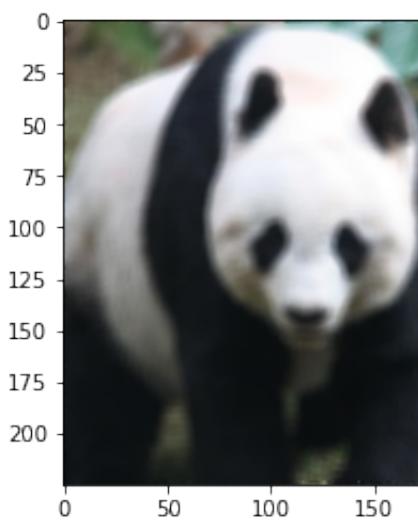
- BLUR
- CONTOUR
- DETAIL
- EDGE_ENHANCE
- EDGE_ENHANCE_MORE
- EMBOSSED
- FIND_EDGES
- SHARPEN
- SMOOTH
- SMOOTH_MORE

BoxBlur: Desenfoca la imagen configurando cada píxel al valor promedio de los píxeles en un cuadrado que extiende los píxeles del radio en cada dirección. Admite radio flotante de tamaño arbitrario. Utiliza una implementación optimizada que se ejecuta en tiempo lineal en relación con el tamaño de la imagen para cualquier valor de radio.

GaussianBlur: Desenfoca la imagen con una secuencia de filtros de cuadro extendidos, que se aproxima a un núcleo gaussiano.

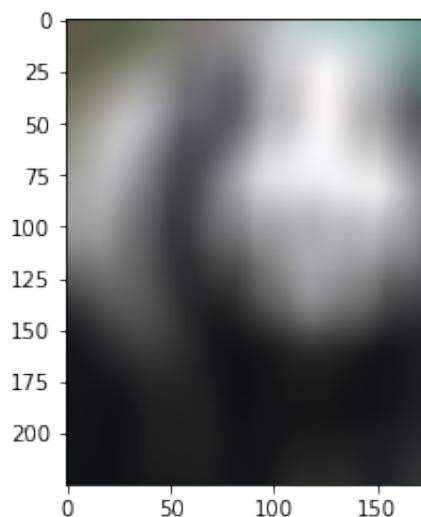
```
In [61]: blur_img = low_res_img.filter(ImageFilter.BLUR)
plt.imshow(blur_img)
```

```
Out[61]: <matplotlib.image.AxesImage at 0x29a5d0be0>
```



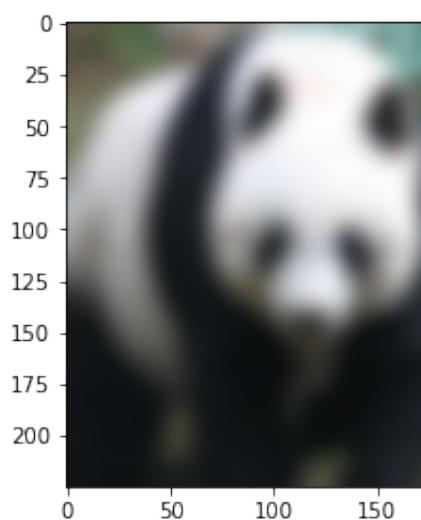
```
In [62]: plt.imshow(low_res_img.filter(ImageFilter.BoxBlur(20)))
```

Out[62]: <matplotlib.image.AxesImage at 0x29a6547f0>



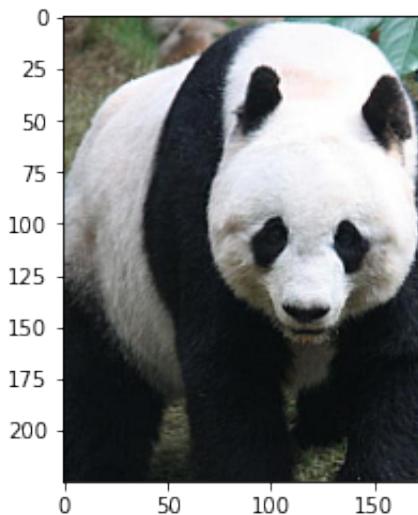
In [63]: plt.imshow(low_res_img.filter(ImageFilter.GaussianBlur(5)))

Out[63]: <matplotlib.image.AxesImage at 0x29a6c4700>



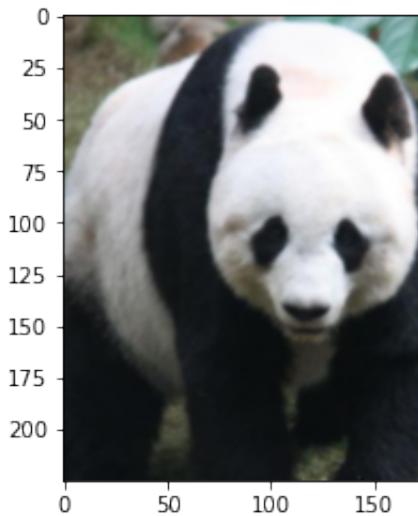
In [64]: plt.imshow(low_res_img.filter(ImageFilter.SHARPEN))

Out[64]: <matplotlib.image.AxesImage at 0x29a7339d0>



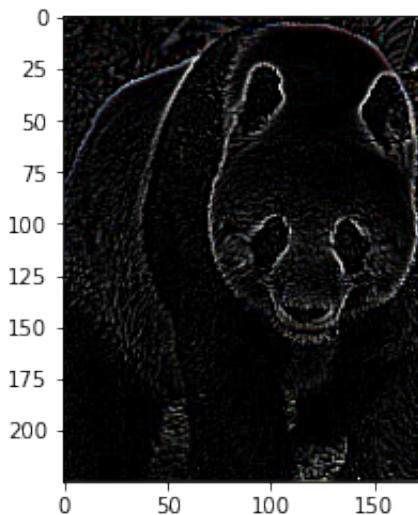
```
In [65]: plt.imshow(low_res_img.filter(ImageFilter.SMOOTH))
```

```
Out[65]: <matplotlib.image.AxesImage at 0x29a7a7550>
```



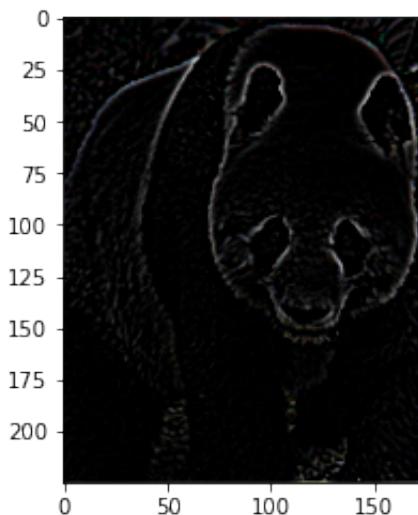
```
In [66]: plt.imshow(low_res_img.filter(ImageFilter.FIND_EDGES))
```

```
Out[66]: <matplotlib.image.AxesImage at 0x29b019400>
```



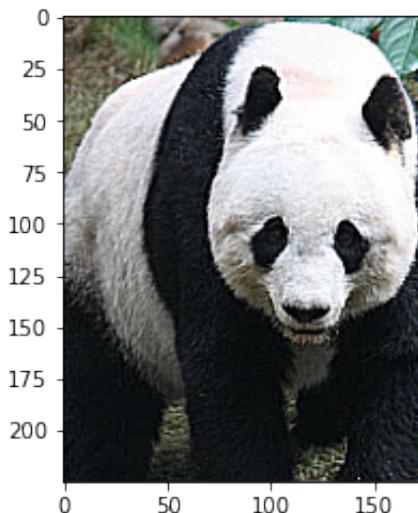
```
In [67]: #combinando filtros smooth, find_edges  
smooth_img = low_res_img.filter(ImageFilter.SMOOTH)  
plt.imshow(smooth_img.filter(ImageFilter.FIND_EDGES))
```

```
Out[67]: <matplotlib.image.AxesImage at 0x29b089670>
```



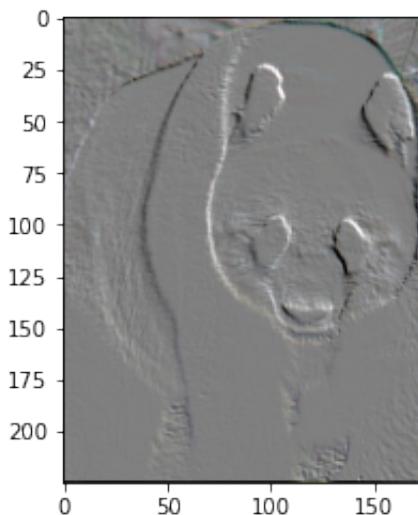
```
In [68]: plt.imshow(low_res_img.filter(ImageFilter.EDGE_ENHANCE))
```

```
Out[68]: <matplotlib.image.AxesImage at 0x29b0f9700>
```



```
In [69]: plt.imshow(low_res_img.filter(ImageFilter.EMBOSS))
```

```
Out[69]: <matplotlib.image.AxesImage at 0x29b16d520>
```



```
In [70]: filename2 = '/Users/irvingestrada/Documents/perro.jpg'
```

```
In [71]: with Image.open(filename2) as img2:  
    img2.load()
```

```
In [72]: plt.imshow(img2)
```

```
Out[72]: <matplotlib.image.AxesImage at 0x29a3788e0>
```



```
In [73]: img2_gray = img2.convert("L")
img2_gray
```

Out[73]:



```
In [74]: red, green, blue = img2.split()
threshold = 57
img2_threshold = blue.point(lambda x: 255 if x > threshold else 0)
img2_threshold = img2_threshold.convert("1")
img2_threshold
```

Out[74]:



ImageFilter.MinFilter: Crea un filtro mínimo. Selecciona el valor de píxel más bajo en una ventana con el tamaño dado.

ImageFilter.MaxFilter: Crea un filtro máximo. Selecciona el valor de píxel más grande en una ventana con el tamaño dado.

```
In [75]: def erode(cycles, image):
    for _ in range(cycles):
        image = image.filter(ImageFilter.MinFilter(3))
    return image

def dilate(cycles, image):
    for _ in range(cycles):
        image = image.filter(ImageFilter.MaxFilter(3))
    return image
```

```
In [76]: step_1 = erode(12, img2_threshold)
step_1
```

Out[76]:

In [77]:
step_2 = dilate(140, step_1)
step_2

Out[77]:

In [78]:
mask = erode(45, step_2)
mask

Out[78]:



convert: Devuelve una copia convertida de la imagen.

In [79]:

```
mask = mask.convert("L")
mask = mask.filter(ImageFilter.BoxBlur(25))
mask
```

Out[79]:



point: se puede utilizar para traducir los valores de píxeles de una imagen

composite: Crea una imagen compuesta combinando imágenes usando una máscara de transparencia.

```
In [80]: blank = img2.point(lambda _: 0)
segmented = Image.composite(img2, blank, mask)
segmented
```

Out[80]:



```
In [81]: img_copy_paste = img
```

```
In [82]: img.size
```

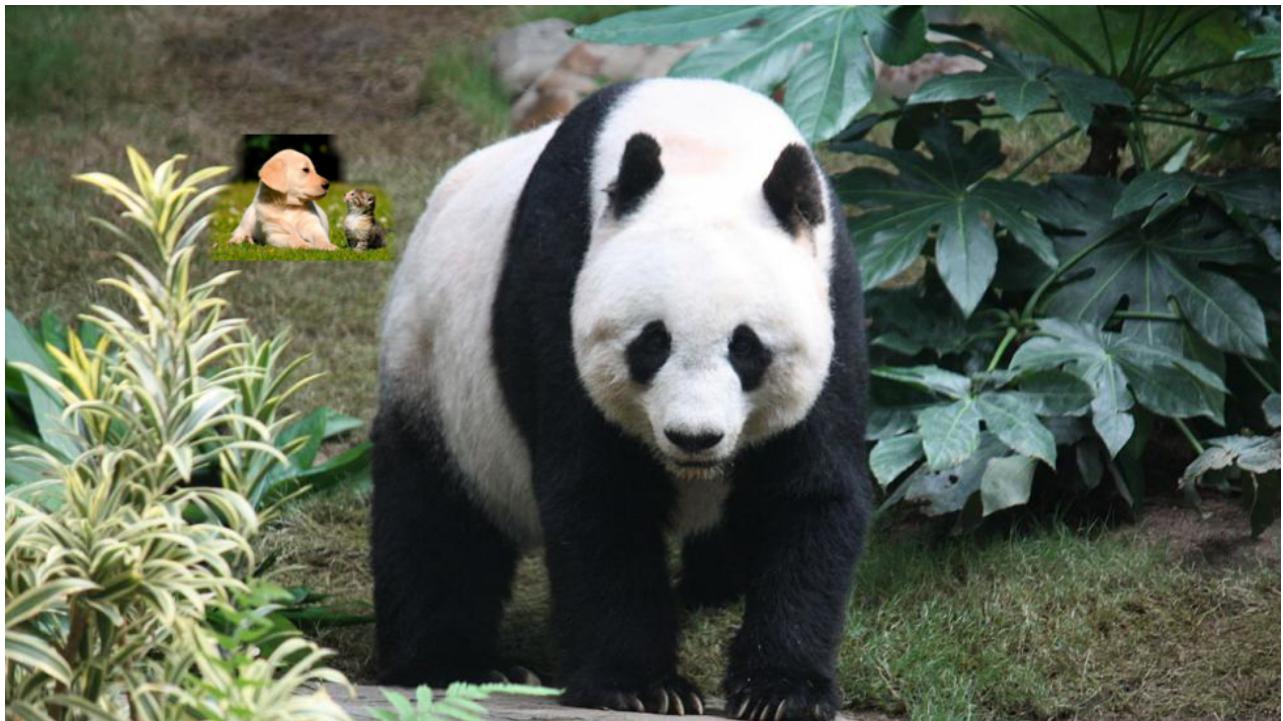
Out[82]: (990, 556)

```
In [83]: img_copy_paste.paste(
    img2.resize((img2.width // 5, img2.height // 5)),
    (150, 100),
    mask.resize((mask.width // 5, mask.height // 5)),)
```

En el proceso anterior obtenemos una máscara de la imagen para poder extraer a ambos animales, obteniendo la sección relevante de la imagen y pegándola en nuestra imagen original. Este ejercicio nos ayuda a aterrizar los procesos que trabajamos de forma individual, ya que unimos todos para hacer ésta edición.

```
In [84]: img_copy_paste
```

Out[84]:



Comparación entre librerías

A continuación se presenta la descripción más detallada de cada una de las librerías con la intención de posteriormente compararlas de una forma general.

Sickit-image: A diferencia de OpenCV, Sickit-image no tiene integrado Numpy tal cual hablando de operaciones detrás de las transformaciones, sin embargo, sus imágenes son representadas como Numpy arrays. Esta librería está basada en `scipy.ndimage` para proporcionar un conjunto versátil de rutinas de procesamiento de imágenes en Python. En su documentación nos instruyen que podemos apoyarnos en Numpy para realizar operaciones más complejas, pero se importa esta librería por separado, de igual forma se importa Matplotlib para poder visualizar las imágenes.

OpenCV: Es una librería con una amplia posibilidad de operaciones. Su documentación es basta, se muestra tanto el ejemplo de código, así como la profundidad matemática que tiene cada uno de los procesos. Esta utiliza Numpy para hacer sus operaciones matriciales. Todas las estructuras de matriz de OpenCV se convierten en matrices Numpy. Esto también facilita la integración con otras bibliotecas que usan Numpy, como SciPy y Matplotlib.

PIL: La documentación de PIL está más enfocada hacia el código, a diferencia de las dos anteriores librerías. Desglosa muy bien cada uno de los módulos los cuales podemos utilizar. Su objetivo menciona que nos brinda un acceso rápido a los datos almacenados en unos pocos formatos de píxeles básicos, también proporciona una base sólida para una herramienta general de procesamiento de imágenes.

Reiterando lo anteriormente mencionado, las tres librerías son bastas en operaciones básicas con imágenes, es importante destacar que al menos las dos primeras librerías mencionadas trabajan en conjunto con Numpy, en la documentación de PIL no se encontró información del funcionamiento de bajo nivel de ésta. Dos librerías que debemos de tener en cuenta al momento de hablar de imágenes es NumPy y SciPy, las cuales funcionan en un nivel más bajo y menos intuitivo que las que trabajamos en este notebook. Dichas librerías trabajan explícitamente con las matrices y sus transformaciones, incluso como hemos estado mencionando algunas de las librerías de alto nivel trabajan con éstas.

Conclusión

Las librerías utilizadas en este notebook son bastante parecidas al menos en sus operaciones básicas de preprocesado de imágenes, cada una de las librerías contienen operaciones más complejas de análisis de imágenes y video. Las librerías que utilizamos están compuestas de otras librerías para realizar sus operaciones y métodos numéricos como NumPy. Las imágenes al final son matrices con los valores de cada uno de pixeles. Las transformaciones que se pueden llevar a cabo no dejan de ser operaciones matriciales. En la mayor parte de este documento vimos los procesos y las transformaciones de las imágenes de manera individual, hay que tomar en cuenta que la combinación de distintos métodos nos pueden llevar a un mejor resultado en nuestro análisis de imágenes. Esta práctica nos abre la posibilidad de utilizar las técnicas que se adecuen a nuestras imágenes y beneficien su análisis. A diferencia de otros tipos de análisis, el análisis de imágenes tiene una gran cantidad de parámetros los cuales determinan la calidad del modelo. El tener una gran cantidad de parámetros nos aumenta las combinaciones posibles entre ellos y las pruebas posibles por hacer, por eso es importante conocer las técnicas de preprocesado para tener herramientas las cuales mejoren el resultado.

Referencias

- Irving Estrada. Github. 2022, Sitio web: <https://github.com/Irving-Estrada/Procesamiento>
- Alex Clark. (2010). Pillow Documentation. 28 De junio del 2022, de Pillow Sitio web: <https://pillow.readthedocs.io/en/stable/index.html>
- Gary Bradsky. (2005). OpenCV-Python Tutorials. 28 de Junio del 2022, de OpenCV Sitio web: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors.. (2022-02-17). scikit-image 0.19.2 docs. 28 de junio 2022, de scikit-image Sitio web: <https://scikit-image.org>
- Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) <https://doi.org/10.7717/peerj.453>

In []: