

▼ Procesamiento de imágenes digitales

David Eduardo Gallardo Fernández 0931556

En la primer parte del trabajo se utilizó el mismo código sólo se cambiaron las imágenes y algunos parámetros.

Por el tamaño del notebook github no me permitió subirlo así que lo estoy compartiendo desde drive con esta liga puede verlo:

<https://colab.research.google.com/drive/17XANK2qe7nkVshuVPJnqXOnSqPUeaMaE?usp=sharing>

Pillow vs OpenCV

Por lo que pude ver tanto Pillow como OpenCV son bastante sencillas de utilizar, con ambas se pueden realizar las tareas que queramos hacer con las imágenes pero sentí que OpenCV era un poco más rápida, al menos en Colab, habría que probar en otros ambientes para ver si se obtienen resultados similares o diferentes.

▼ Operaciones basicas con Pillow

+ Code

+ Text

```
1  from PIL import Image
2  import matplotlib as plt
3  # filename = "/content/drive/MyDrive/Datasets/buildings.jpg"
4  filename = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/mandalas-1084082.jpg" # Aquí solame
5  with Image.open(filename) as img:
6      img.load()
7
8  type(img)
9
10 isinstance(img, Image.Image)

True
```

```
1  #img.show() # Esta línea de código con me funcionada así que la comenté.
2  img. #Desplegué la pantalla solamente llamando la variable img pero siento que se tarda.
```



```
1 print(img.format)
2 print(img.size)
3 print(img.mode)
```

```
JPEG
(3000, 2000)
RGB
```

```
1 cropped_img = img.crop((300, 150, 700, 1000))  
2 cropped_img.size  
3 #cropped_img.show()  
4 cropped_img
```



Una vez que tenemos la imagen recortada, podemos cambiar la resolución de la imagen con el comando de `resize()`.

```
1 low_res_img = cropped_img.resize((cropped_img.width // 4, cropped_img.height // 4))
2 #low_res_img.show()
3 low_res_img
```



La diferencia que vemos en esta imagen para el uso de las tuplas en los parametros, solo estamos definiendo la nueva altura y el nuevo ancho. En el caso del código queremos generar un ancho y alto proporcional, por lo que usamos el divisor piso `//`.

```
1 low_res_img = cropped_img.reduce(4)
2 low_res_img
```



El argumento determina el factor por el cual reduce la escala de la imagen. Si prefiere establecer un tamaño máximo en lugar de un factor de escala, puede usar `thumbnail()`. El tamaño de la miniatura será menor o igual al tamaño que establezca.

Un factor importante del valor a `thumbnail` es que este método cambia la imagen original, mientras que `crop`, `resize` y `reduce` regresan un objeto de imagen nuevo, que se guarda en la variable que

elegiste.

Una vez que estes satisfecho con los cambios realizados a la imagen, tienes que guardarlos con la funcion `save()`.

```
1 cropped_img.save('/content/drive/MyDrive/MCD/4_PCD/Tarea05/crooped_img.jpg')
2 low_res_img.save('/content/drive/MyDrive/MCD/4_PCD/Tarea05/low_res_crop.jpg')
```

▼ Manipulacion basica de imagenes.

```
1 converted_img = img.transpose(Image.FLIP_TOP_BOTTOM)
2 #converted_img.show()
3 converted_img
```



Existen siete diferentes opciones que podemos utilizar para pasar el argumento de `transpose()`

1. **Image.FLIP_LEFT_RIGHT:** Voltea la imagen de izquierda a derecha, resultando en una imagen como reflejo de espejo.
2. **Image.FLIP_TOP_BOTTOM:** Voltea la imagen de arriba a abajo.
3. **Image.ROTATE_90:** Rota la imagen por 90 grados contra reloj.
4. **Image.ROTATE_180:** Rota la imagen por 180 grados.
5. **Image.ROTATE_270:** Rota la imagen por 270 grados contra reloj, lo que seria lo mismo que 90 grados en sentido al reloj.
6. **Image.TRANSPOSE:** Transpone las filas y las columnas usando el pixel de arriba a la izquierda como el origen, siendo el píxel superior izquierdo el mismo en la imagen transpuesta que en la imagen original
7. **Image.TRANSVERSE:** Transpone las filas y columnas utilizando el píxel inferior izquierdo como origen, siendo el píxel inferior izquierdo el que permanece fijo entre la versión original y la modificada

En caso de que necesites una rotacion en un angulo mas especifico, se usa la funcion de `rotate()`.

```
1 rotated_img = img.rotate(45)
2 rotated_img
```



Como podemos ver, para completar la rotacion y dejar la imagen con la misma resolucion de tamaño, se cortan los bordes de la imagen. Si necesitamos conservarlos, tenemos que agregar el parametro `expand` al código anterior.

```
1 rotated_img = img.rotate(45, expand=True)
2 rotated_img
```

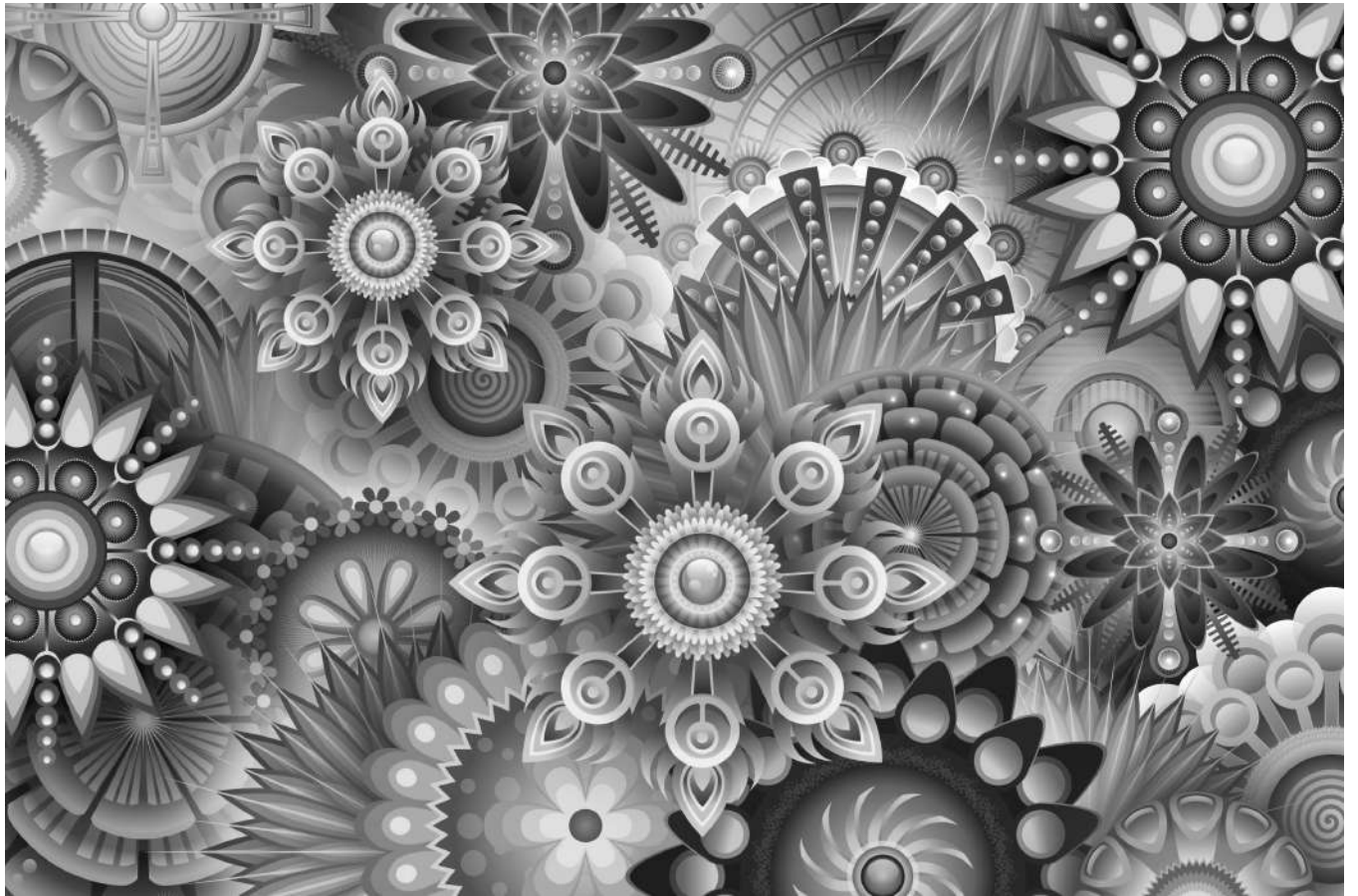



▼ Diferentes modos de imagen

Como mencionamos anteriormente, una imagen es un vector de pixeles donde cada pixel corresponde a un color. Cada pixel puede ser representado por uno o mas valores. En el caso de las imagenes RGB por ejemplo, si tenemos una del tamaño 100x100, los pixeles se representan como 100x100x3.

El modo de una imagen describe con qué tipo de imagen está trabajando. Pillow admite la mayoría de los modos estándar, incluidos blanco y negro (binario), escala de grises, RGB, RGBA y CMYK.

```
1 filename = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/mandalas-1084082.jpg"
2 with Image.open(filename) as img:
3     img.load()
4
5
6 cmyk_img = img.convert("CMYK")
7 gray_img = img.convert("L") # Grayscale
8
9 #cmyk_img
10 gray_img
11
```



```
1 print(img.getbands())
2 print(cmyk_img.getbands())
3 print(gray_img.getbands())

('R', 'G', 'B')
('C', 'M', 'Y', 'K')
('L',)
```

```
1 red, green, blue = img.split()
2 red
3
```




```
1 red.mode
```

```
    'L'
```



```
1 zeroed_band = red.point(lambda _: 0)
```

```
2
```

```
3 red_merge = Image.merge("RGB", (red, zeroed_band, zeroed_band))
```

```
4 green_merge = Image.merge("RGB", (zeroed_band, green, zeroed_band))
```

```
5 blue_merge = Image.merge("RGB", (zeroed_band, zeroed_band, blue))
```

```
6
```

```
7 red_merge
```



1 green_merge



1 blue_merge



▼ Procesamiento de imágenes usando Pillow

La manipulación más sencilla de las imágenes como rotar, recortar y modificar su tamaño son acciones que no dañan o modifican el contenido de la imagen.



▼ Blurring, sharpening, smoothing.

```
1 from PIL import Image, ImageFilter
2 filename = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/mandalas-1084082.jpg"
3 with Image.open(filename) as img:
4     img.load()
```

```
1 blur_img = img.filter(ImageFilter.BLUR)
2 blur_img
```



```
1 img.crop((300, 300, 500, 500))
```



```
1 blur_img.crop((300, 300, 500, 500))
```



```
1 img.filter(ImageFilter.BoxBlur(5))
```




```
1 img.filter(ImageFilter.BoxBlur(20))
```



```
1 img.filter(ImageFilter.GaussianBlur(20))
```



```
1 sharp_img = img.filter(ImageFilter.SHARPEN)
2 img.crop((300, 300, 500, 500))
3
```



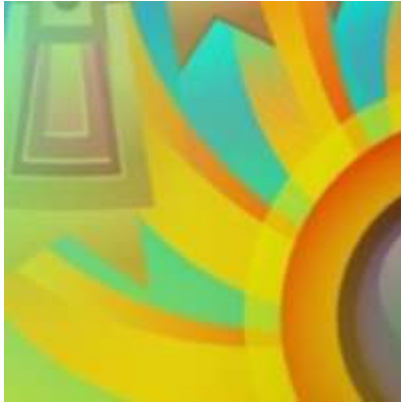
```
1 sharp_img.crop((300, 300, 500, 500))
```



```
1 smooth_img = img.filter(ImageFilter.SMOOTH)
2 img.crop((300, 300, 500, 500))
```



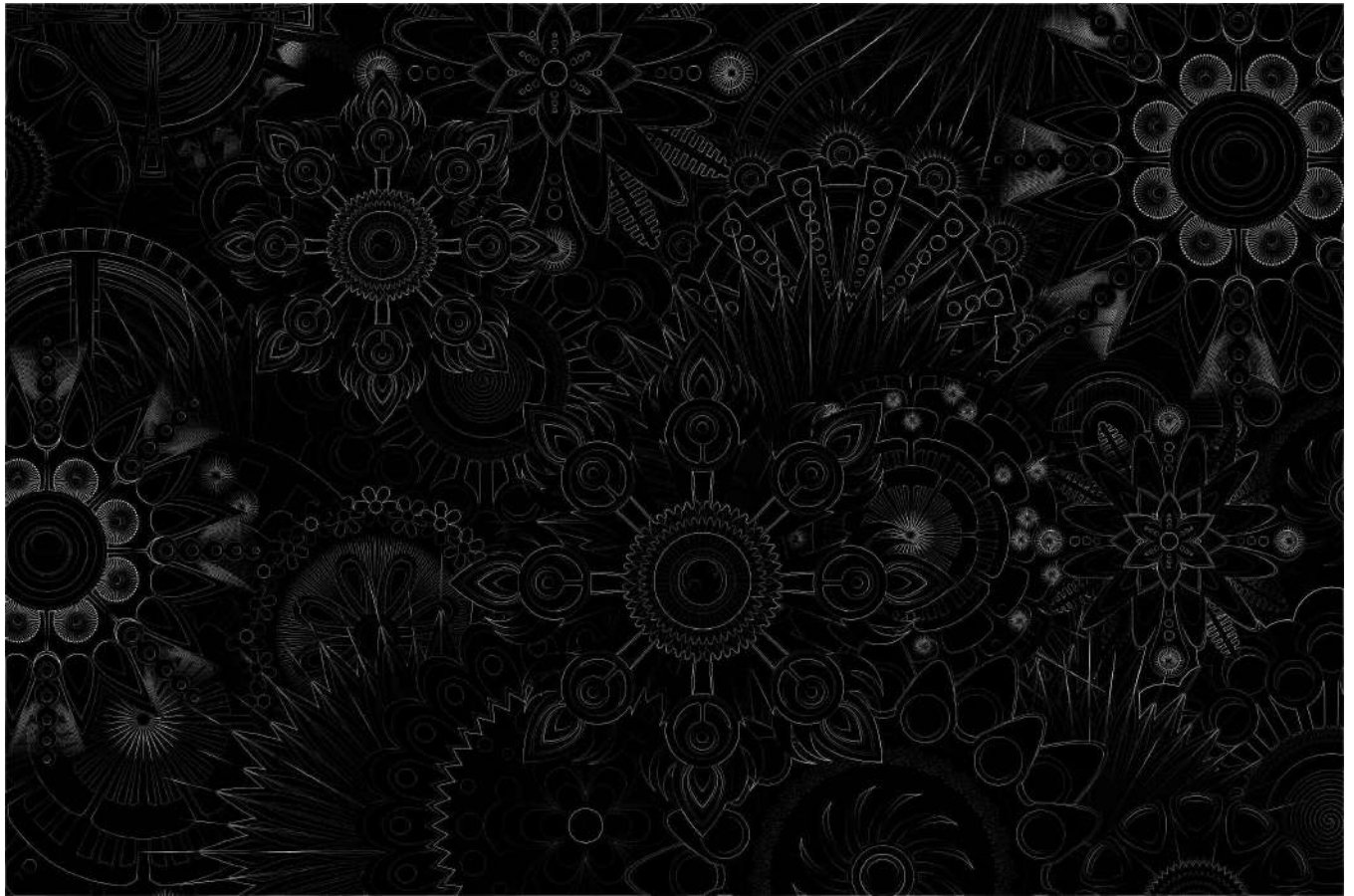
```
1  
2 smooth_img.crop((300, 300, 500, 500))
```



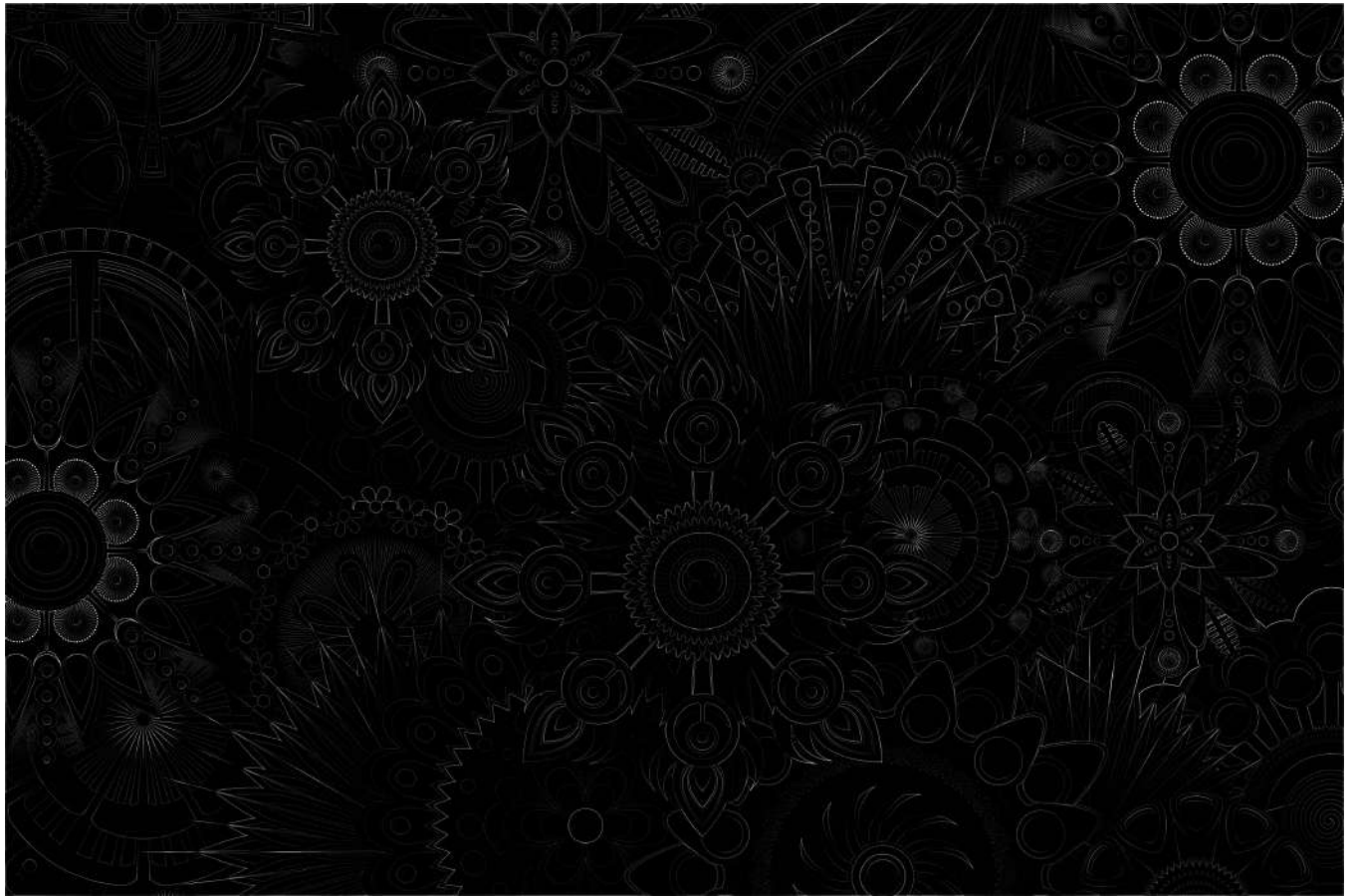
▼ Deteccion de bordes, mejora de bordes y embossing.

Cuando miramos una imagen para nosotros es bastante sencillo determinar los bordes de los objetos. Para un algoritmo se necesita un kernel especial.

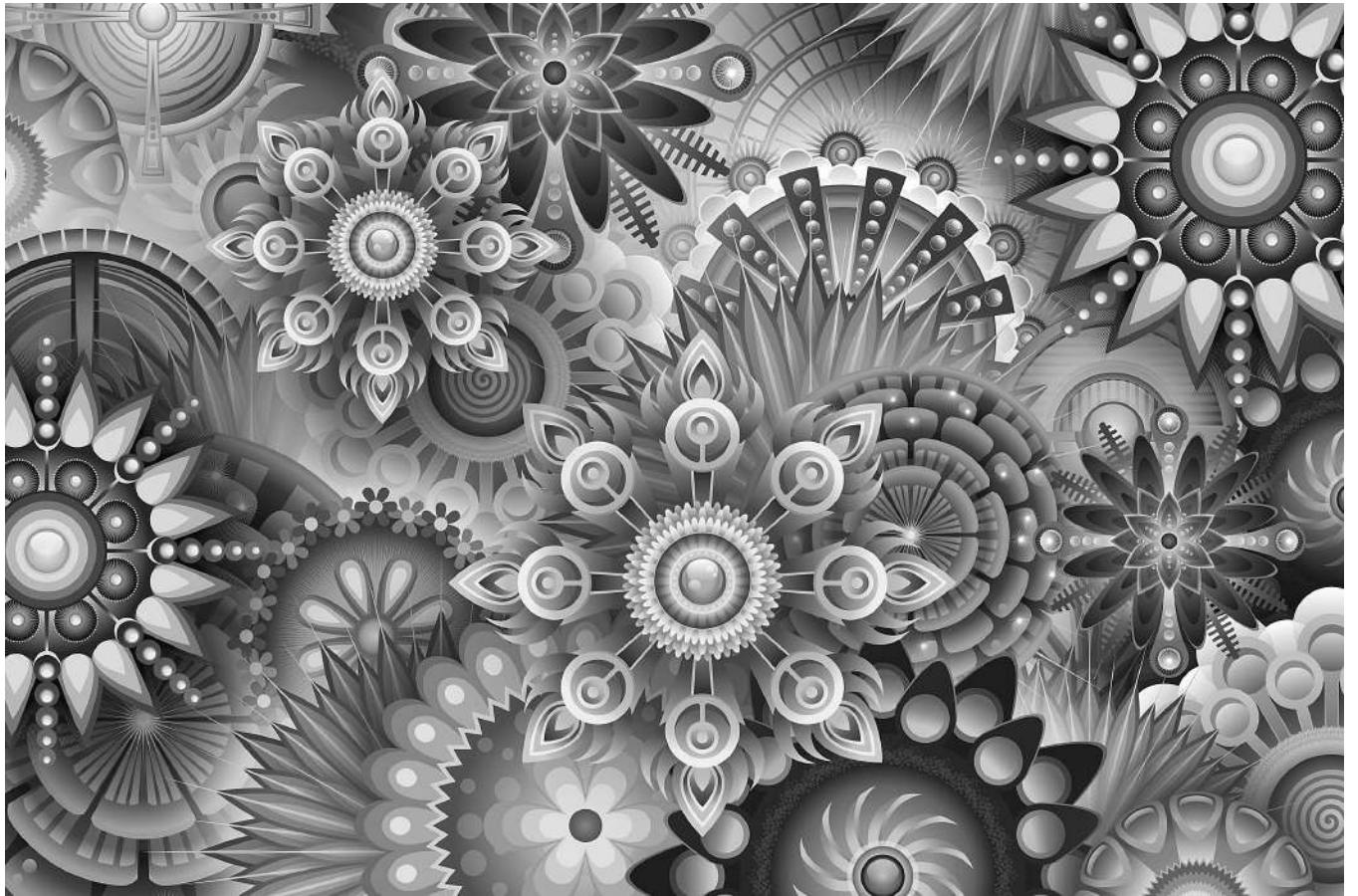
```
1 img_gray = img.convert("L")  
2 edges = img_gray.filter(ImageFilter.FIND_EDGES)  
3 edges
```

```
1  img_gray_smooth = img_gray.filter(ImageFilter.SMOOTH)
2  edges_smooth = img_gray_smooth.filter(ImageFilter.FIND_EDGES)
3  edges_smooth
```



```
1 edge_enhance = img_gray_smooth.filter(ImageFilter.EDGE_ENHANCE)
2 edge_enhance
```



```
1 emboss = img_gray_smooth.filter(ImageFilter.EMBOSS)
2 emboss
```



▼ Segmentacion de imagenes y superposicion.

Pillow tiene una libreria para usar este tipo de procesamiento.

Nota: La imagen del auto que seleccione parece que no funcionó muy bien.

```
1 from PIL import Image
2 filename_cat = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/car.jpg"
3
4 with Image.open(filename_cat) as img_cat:
5     img_cat.load()
6 # img_cat = img_cat.crop((800, 0, 1650, 1281))
7 img_cat
```




```
1 img_cat_gray = img_cat.convert("L")  
2 img_cat_gray  
3
```



```
1 threshold = 100
2 img_cat_threshold = img_cat_gray.point(
3     lambda x: 255 if x > threshold else 0
4 )
5 img_cat_threshold
```



```
1 red, green, blue = img_cat.split()
2 red
3
```



1 green

2



1 blue



El canal azul tiene un mayor contraste entre los píxeles que representan al gato y los que representan el fondo. Puede usar la imagen del canal azul para establecer el umbral:

```
1 threshold = 57
2 img_cat_threshold = blue.point(lambda x: 255 if x > threshold else 0)
3 img_cat_threshold = img_cat_threshold.convert("1")
4 img_cat_threshold
```




Utiliza un valor de umbral de 57 en este ejemplo. También convierte la imagen en un modo binario usando "1" como argumento para `.convert()`. Los píxeles de una imagen binaria solo pueden tener los valores 0 o 1.

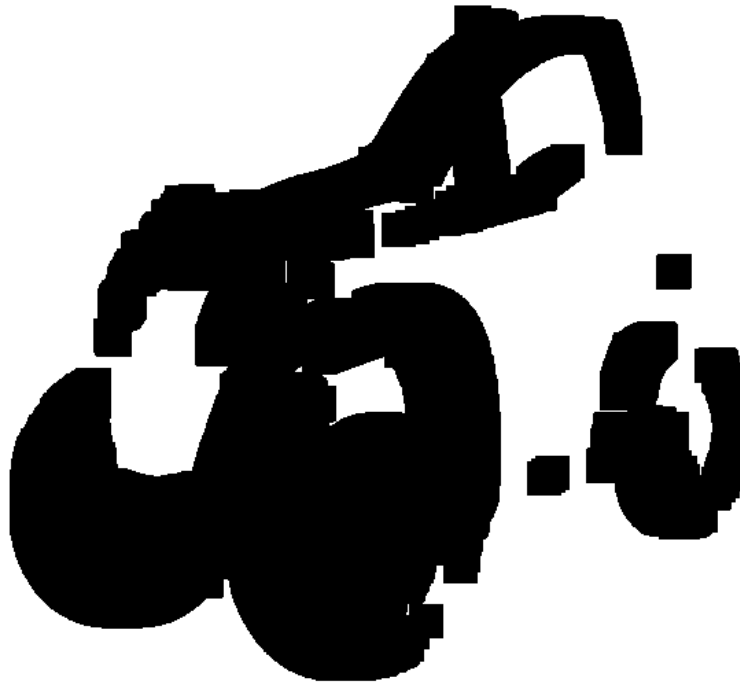
Nota: cuando se trata de ciertos formatos de imagen, como JPEG, que dependen de la compresión con pérdida, las imágenes pueden variar ligeramente según los decodificadores JPEG que esté utilizando. Por lo tanto, es posible que tengas que ajustar ligeramente el valor del umbral si los resultados no coinciden con los que se muestran en este ejemplo.

Puedes identificar al gato en esta imagen en blanco y negro. Sin embargo, quieren tener una imagen en la que todos los píxeles que corresponden al gato sean blancos y todos los demás píxeles sean negros. En esta imagen, todavía tiene regiones negras en el área que corresponde al gato, como donde están los ojos, la nariz y la boca, y también tiene píxeles blancos en otras partes de la imagen.

Dentro de pillow están las técnicas de procesamiento de imágenes llamadas erosión y dilatación para crear una mejor máscara que represente al gato.

▼ Segmentacion de imagen con Tresholding.


```
1 def erode(cycles, image):  
2     for _ in range(cycles):  
3         image = image.filter(ImageFilter.MinFilter(3))  
4     return image  
5  
6  
7 def dilate(cycles, image):  
8     for _ in range(cycles):  
9         image = image.filter(ImageFilter.MaxFilter(3))  
10    return image  
11  
12 step_1 = erode(12, img_cat_threshold)  
13 step_1  
14
```



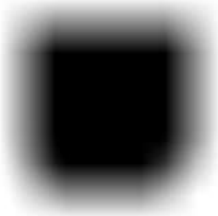
```
1 step_2 = dilate(58, step_1)
2 step_2
```



```
1 cat_mask = erode(45, step_2)
2 cat_mask
```



```
1 cat_mask = cat_mask.convert("L")
2 cat_mask = cat_mask.filter(ImageFilter.BoxBlur(20))
3 cat_mask
```



```
1 blank = img_cat.point(lambda _: 0)
2 cat_segmented = Image.composite(img_cat, blank, cat_mask)
```


3 cat segmented



▼ Superposicion de imagenes

```
1 filename_monastery = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/mandalas-1084082.jpg"
2 with Image.open(filename_monastery) as img_monastery:
3     img_monastery.load()
4
5 img_monastery.paste(
6     img_cat.resize((img_cat.width // 5, img_cat.height // 5)),
7     (1300, 750),
8     cat_mask.resize((cat_mask.width // 5, cat_mask.height // 5)),
9 )
10
11 img_monastery
```



Para esta seccion utilizamos el formato `paste()` en donde agregamos una imagen a otra. Este metodo puede cambiar en tres parametros:

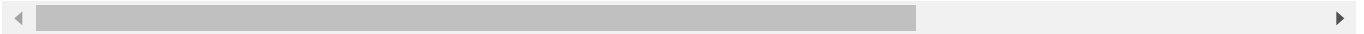
1. El primer argumento es la imagen que tu quieres agregar. En este caso estamos modificando el tamaño de la imagen a un quinto del tamaño de la imagen de fondo con el operador //.
2. El segundo argumento es la localización (location) de la imagen principal y la imagen secundaria. Esta tupla incluye las coordenadas de la imagen principal y en este caso mencionamos que queremos que se ponga la imagen secundaria.

6 # El tercer argumento es la imagen que se va a agregar a la imagen principal

▼ Crear una marca de agua

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

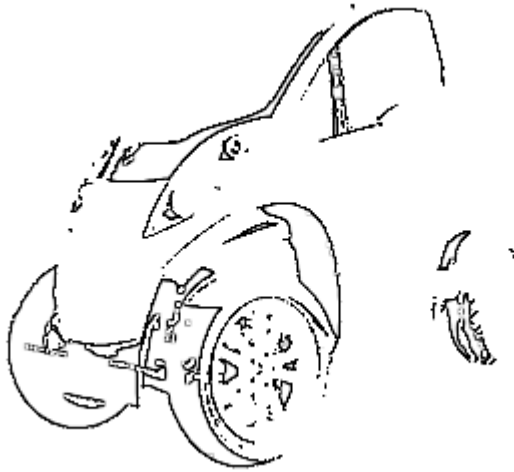
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun



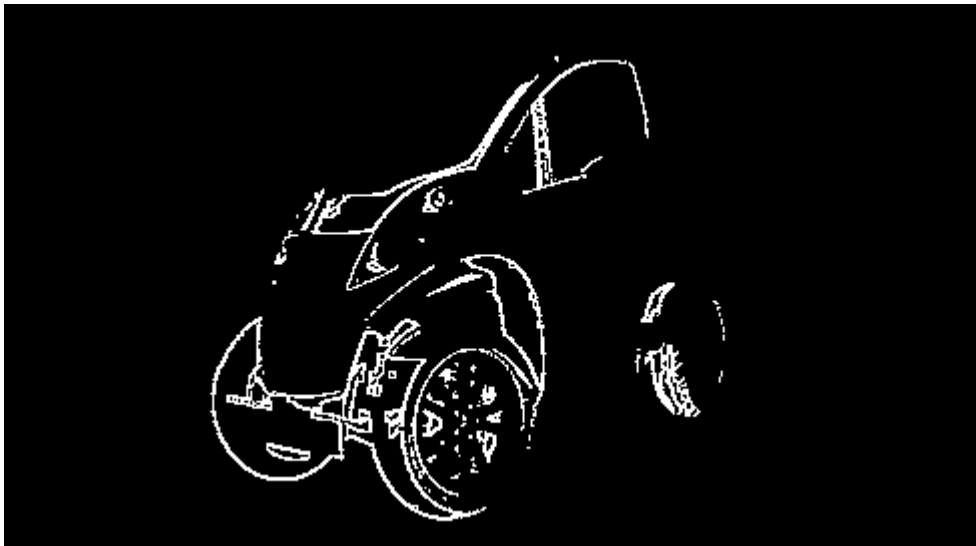
```
1 logo = "/content/drive/MyDrive/MCD/4_PCD/Tarea05/car.png"
2 with Image.open(logo) as img_logo:
3     img_logo.load()
4
5
6 img_logo = Image.open(logo)
7 img_logo
```



```
1 img_logo = img_logo.convert("L")
2 threshold = 50
3 img_logo = img_logo.point(lambda x: 255 if x > threshold else 0)
4 img_logo = img_logo.resize(
5     (img_logo.width // 2, img_logo.height // 2)
6 )
7 img_logo = img_logo.filter(ImageFilter.CONTOUR)
8 img_logo
```



```
1 img_logo = img_logo.point(lambda x: 0 if x == 255 else 255)
2 img_logo
```



```
1 img_monastery.paste(img_logo, (480, 400), img_logo)
```




```

1 import numpy as np
2 import imageio
3 import matplotlib.pyplot as plt
4 from scipy.signal import convolve2d
5
6 def Convolution(image, kernel):
7     conv_bucket= []
8     for d in range(image.ndim):
9         conv_channel= convolve2d(image[:, :, d], kernel,
10                                 mode="same", boundary="symm")
11     conv_bucket.append(conv_channel)
12     return np.stack(conv_bucket, axis=2).astype("uint8")
13
14
15 kernel_sizes= [9,15,30,60]
16 fig, axs=plt.subplots(nrows=1, ncols=len(kernel_sizes), figsize=(15,15));
17
18 pic =imageio.imread('/content/drive/MyDrive/MCD/4_PCD/Tarea05/car.jpg')
19
20 for k, ax in zip(kernel_sizes, axs):
21     kernel =np.ones((k,k))
22     kernel /=np.sum(kernel)
23     ax.imshow(Convolution(pic, kernel));
24     ax.set_title("Convolved By Kernel: {}".format(k));
25     ax.set_axis_off();

```



▼ OpenCV

En esta segunda parte se probará la librería OpenCV para realizar algunas operaciones similares a las realizadas con Pillow.

```

1 import cv2
2 from google.colab.patches import cv2_imshow

1 img = cv2.imread("/content/drive/MyDrive/MCD/4_PCD/Tarea05/car wallpaper.jpg")

1 cv2_imshow(img)

```




▼ Girar la imagen 90 grados.

En esta parte hago un giro de 90 grados.

```
1 height, width = img.shape[0:2]
```

```
2  rotationMatrix = cv2.getRotationMatrix2D((width/2, height/2), 90, .5)
3  rotatedImage = cv2.warpAffine(img, rotationMatrix, (width, height))
4  cv2_imshow(rotatedImage)
```



▼ Cortar la imagen

```
1 startRow = int(height*.15)
2 startCol = int(width*.15)
3 endRow = int(height*.85)
4 endCol = int(width*.85)
5 croppedImage = img[startRow:endRow, startCol:endCol]
6 cv2_imshow(croppedImage)
```



▼ Redimencionar

```
1 newImg = cv2.resize(img, (0,0), fx=0.75, fy=0.75)
2 cv2_imshow(newImg)
```



▼ Contrast

```
1 import numpy as np

1 contrast_img = cv2.addWeighted(img, 2.5, np.zeros(img.shape, img.dtype), 0, 0)
2
3 cv2.imshow('contrast_img')
```



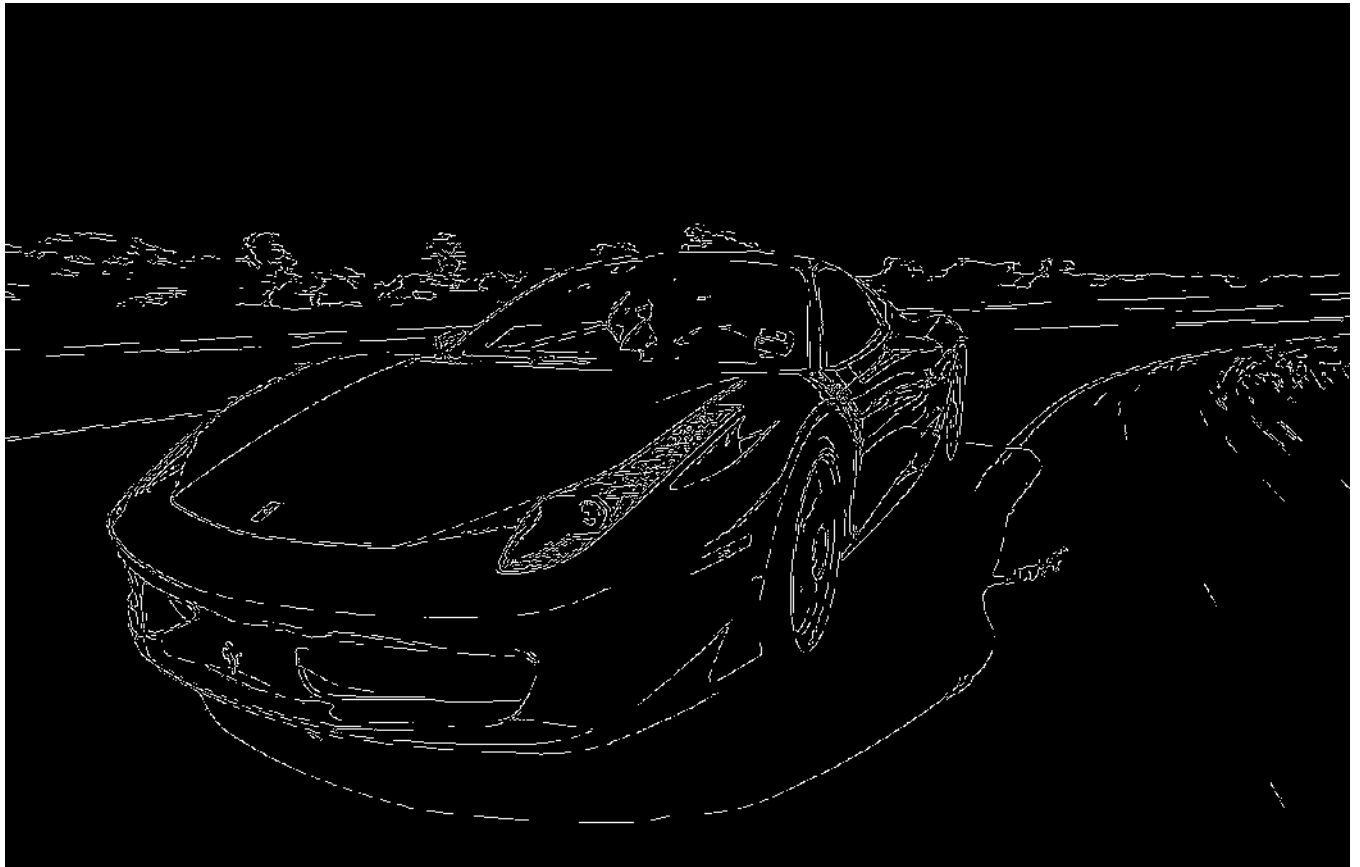

▼ Borroso

```
1 blur_image = cv2.GaussianBlur(img, (7,7), 0)
2
3 cv2.imshow(blur_image)
```



▼ Detectar orillas

```
1 edge_img = cv2.Canny(img,100,200)
2
3 cv2.imshow(edge_img)
```

▼ Escala de grises

```
1 gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
2
3 cv2.imshow(gray_img)
```



▼ Fusión de imágenes

Se combinan dos imágenes para generar una nueva imagen.

```
1 img1 = cv2.imread('/content/drive/MyDrive/MCD/4_PCD/Tarea05/car.jpg')
2 img2 = cv2.imread('/content/drive/MyDrive/MCD/4_PCD/Tarea05/car wallpaper.jpg')
3
4 imgadd = cv2.add(cv2.resize(img1,(500,300)),cv2.resize(img2,(500,300)))
5 cv2_imshow(imgadd)
```



```

1 combine = cv2.addWeighted(cv2.resize(img1,(500,300)),0.5,cv2.resize(img2,(500,300)),0.5,0)
2
3 cv2_imshow(combine)
4

```



```

1
2 img2 = cv2.resize(img2,(200,100))
3
4 # I want to put logo on top-left corner, So I create a ROI
5 # Primero obtenga el roi de la imagen original
6 rows,cols,channels = img2.shape
7 roi = img1[0:rows, 0:cols ]
8
9 # Imagen original convertida a valor gris
10 # Now create a mask of logo and create its inverse mask also
11 img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
12
13 cv2_imshow(img2gray)
14

```

```

15 '''
16     Convierte una imagen gris en blanco o negro. (Mayor que el valor de umbral especificado
17 '''
18     # Binarizar el valor gris para obtener la máscara del área de ROI
19     ret, mask = cv2.threshold(img2gray, 200, 255, cv2.THRESH_BINARY)
20
21     cv2.imshow(mask)
22
23
24     # Máscara invertida del área de la máscara
25     mask_inv = cv2.bitwise_not(mask)
26
27     cv2.imshow(mask_inv)
28
29
30     # Fondo de pantalla de máscara
31     # Now black-out the area of logo in ROI
32     img1_bg = cv2.bitwise_and(roi,roi,mask = mask)
33
34     cv2.imshow(img1_bg)
35
36     #Máscara en primer plano
37     # Take only region of logo from logo image.
38     img2_fg = cv2.bitwise_and(img2,img2,mask = mask_inv)
39     cv2.imshow(img2_fg)
40
41
42     # Superposición de imagen de fondo frontal
43     # Put logo in ROI and modify the main image
44     dst = cv2.add(img1_bg,img2_fg)
45     img1[0:rows, 0:cols ] = dst
46
47     cv2.imshow(img1)
48
49
50     img1[0:rows, 0:cols ] = dst
51
52     cv2.imshow(img1)
53

```



