



**Tecnológico  
de Monterrey**

**TC4033**

## **Visión computacional para imágenes y video**

### **2.2 Google Colab - Simple ImgProcessing**

#### **Equipo 30:**

- Julio Baltazar Colín: A01794476
- Julio Osvaldo Hernández Bucio: A01794366
- Helmy Andrea Moreno Navarro: A01793918
- Maricel Parra Osorio - A01793932
- Yeison Fernando Villamil Franco: A01793803

---

**Los ejercicios que deben agregarse a este proyecto serán las siguientes:**

1. Las transformaciones pixel a pixel son sumamente utilizadas para aumentar la cantidad de imágenes para entrenar modelos de inteligencia artificial, sobre todo aquellas de tipo fotométrico. Investiga 3 tipos de transformaciones y aplicarlas en el proyecto de Google Collab sobre imágenes propias.

2. Investiga una aplicación donde obtener el negativo de imagen tenga un valor específico e integra el código en en una fila de google collab, justificar brevemente tu investigación y haciendo una demo sencilla.
3. Investiga una aplicación donde se puede aplicar la corrección de gamma en una imagen. Integra el código en en una fila de google collab, justifica brevemente tu investigación y haz una demo sencilla.
4. Investiga una aplicación donde se puede usar la sustracción de imágenes e integra el código en en una fila de google collab, justificar brevemente tu investigación, haciendo una demo sencilla.

# 1. Simple Image Operations

## Table of Contents

1. Libraries
2. Loading Images
3. Resizing Images
4. Negative Images
5. Logarithmic Transformation
6. Image Binarizer
7. Image Quantizer

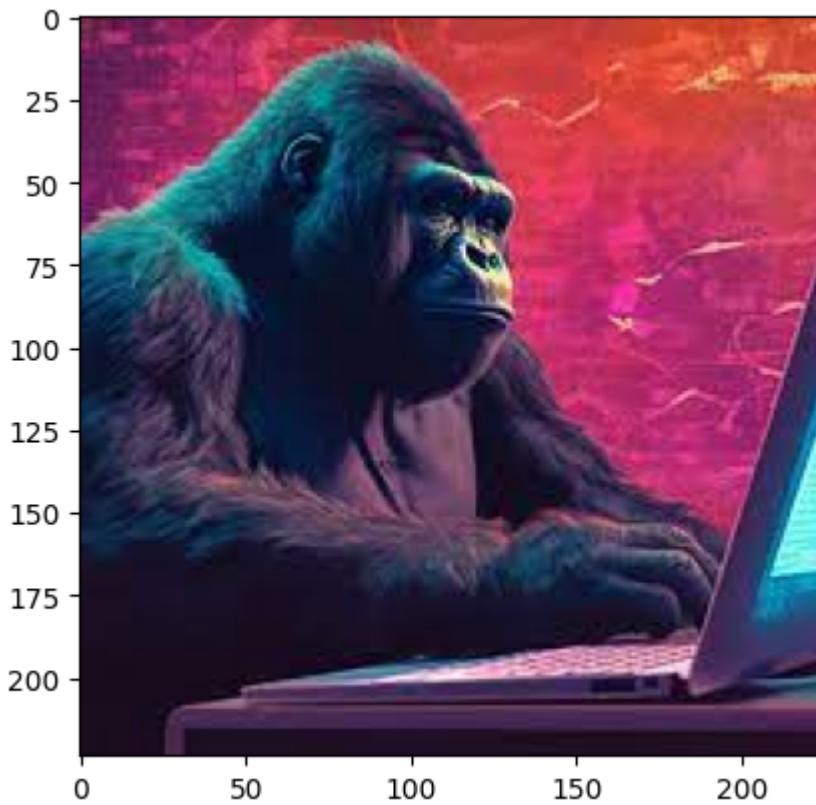
## Importing Libraries

```
In [1]: from matplotlib import image as mpimg  
import matplotlib.pyplot as plt  
import numpy as np  
import cv2
```

## Loading Images

```
In [4]: img1 = mpimg.imread('.../data/semana_2/Gorila.jpeg')  
plt.imshow(img1)  
print(type(img1))
```

<class 'numpy.ndarray'>



Look at the shape of this array:

```
In [6]: img1.shape
```

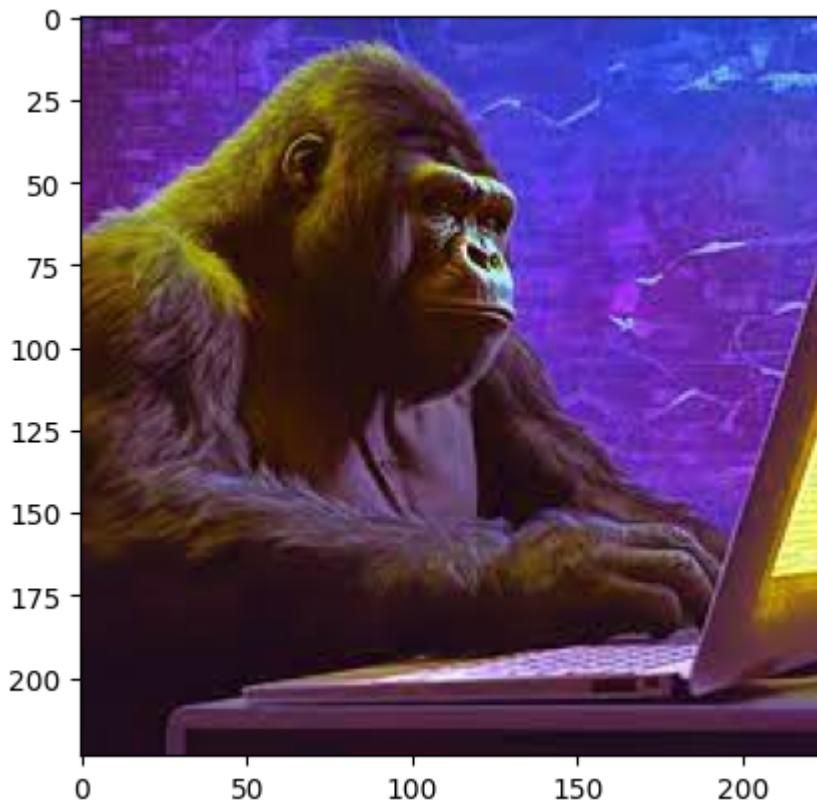
```
Out[6]: (224, 224, 3)
```

The image is actually composed of three "layers, or *channels*, for red, green, and blue (RGB) pixel intensities.

Display the same image but this time we'll use another popular Python library for working with images - **cv2**.

```
In [7]: img2 = cv2.imread('../data/semana_2/Gorila.jpeg')
plt.imshow(img2)
type(img2)
```

```
Out[7]: numpy.ndarray
```

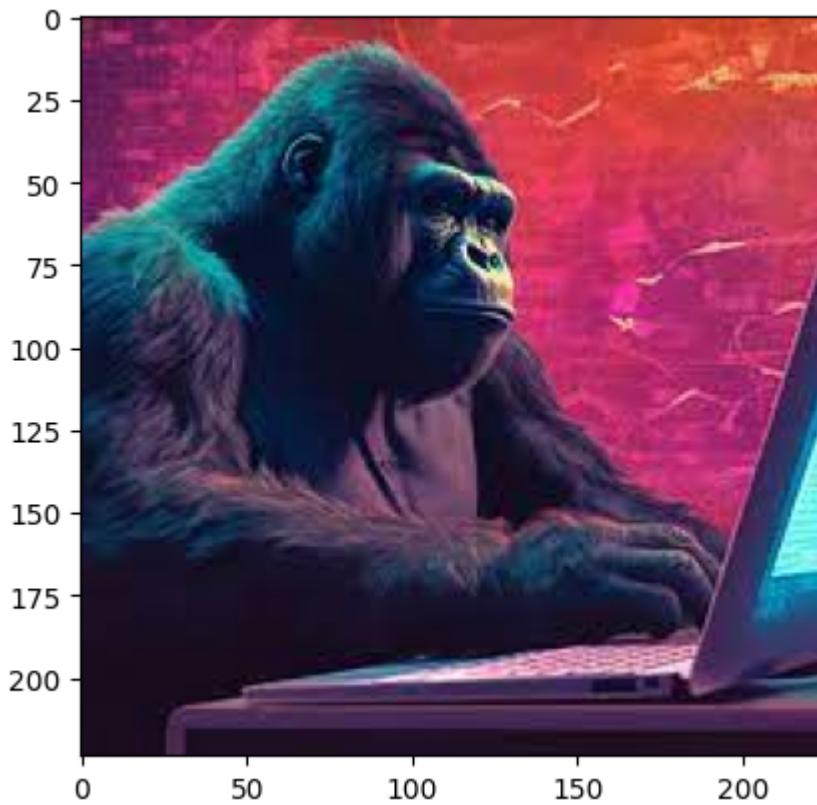


The trouble is that cv2 loads the array of image data with the channels ordered as blue, green, red (BGR) instead of red, green blue (RGB).

Let's fix that

```
In [8]: img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
print(plt.imshow(img2))

AxesImage(size=(224, 224))
```

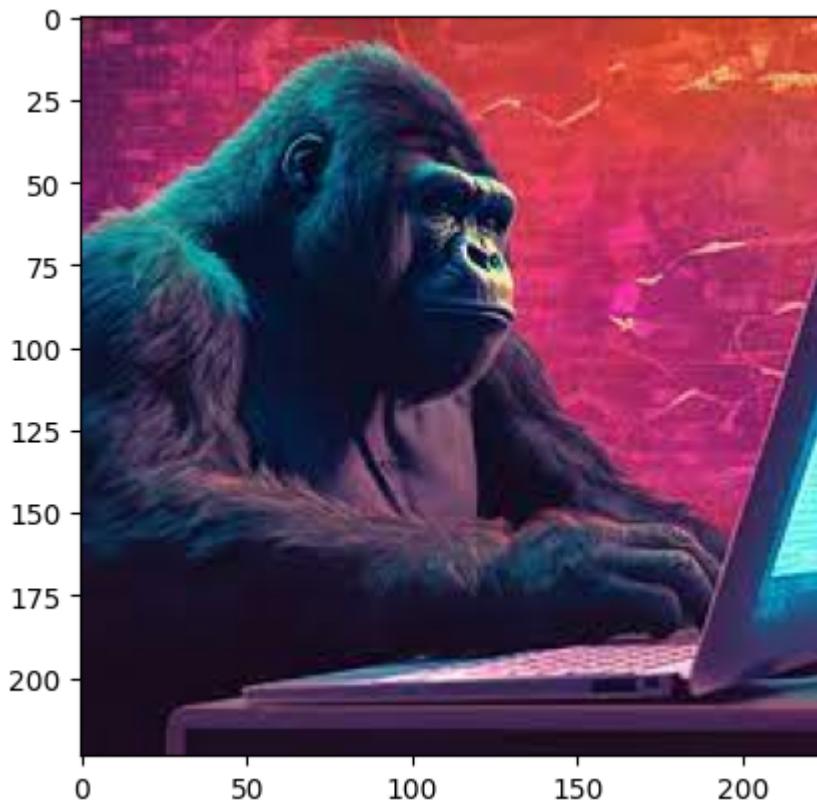


Lastly, one more commonly used library for image processing in Python we should consider - **PIL**:

```
In [9]: from PIL import Image
import matplotlib.pyplot as plt

img3 = Image.open('../data/semana_2/Gorila.jpeg')
plt.imshow(img3)
print(type(img3))
```

<class 'PIL.JpegImagePlugin.JpegImageFile'>



It's easy to convert a PIL JpegImageFile to a numpy array

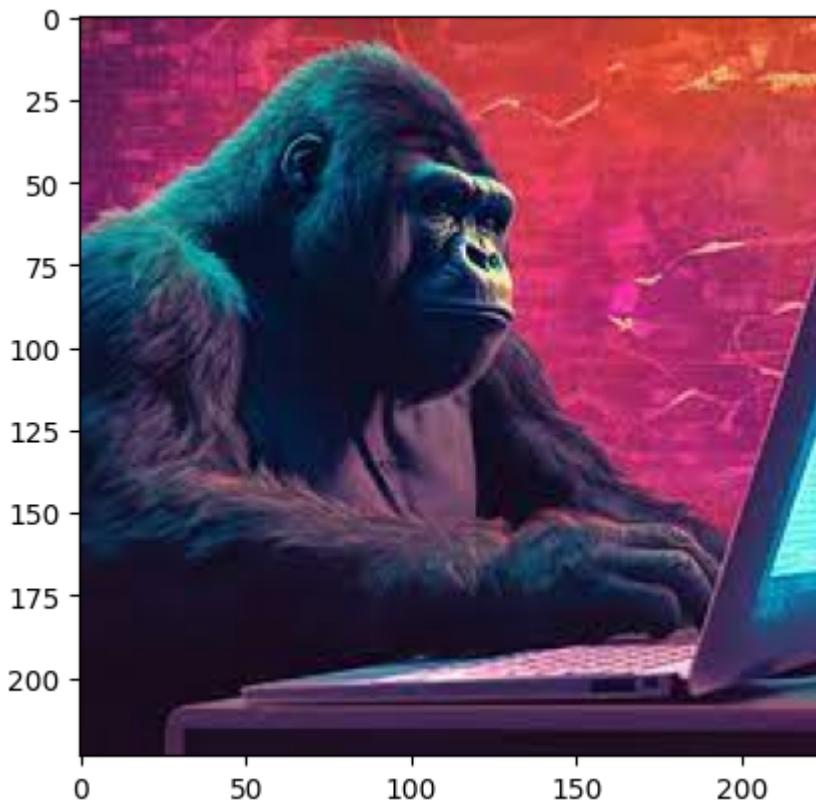
```
In [10]: img3 = np.array(img3)  
img3.shape
```

```
Out[10]: (224, 224, 3)
```

Saving a numpy array in an optimized format, should you need to persist images into storage

```
In [14]: # Save the image  
np.save('../data/semana_2/img.npy', img3)  
  
# Load the image  
img3 = np.load('../data/semana_2/img.npy')  
  
plt.imshow(img3)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x7f6c78db7690>
```



## Resizing an Image

One of the most common manipulations of an image is to resize it.

Generally, we want to ensure that all of your training images have consistent dimensions.

```
In [15]: from PIL import Image, ImageOps

# Load the image array into a PIL Image
orig_img = Image.fromarray(img3)

# Get the image size
o_h, o_w = orig_img.size
print('Original size:', o_h, 'x', o_w)

# We'll resize this so it's 200 x 200
target_size = (200,200)
new_img = orig_img.resize(target_size)
n_h, n_w = new_img.size
print('New size:', n_h, 'x', n_w)

# Show the original and resized images
# Create a figure
fig = plt.figure(figsize=(12, 12))

# Subplot for original image
a=fig.add_subplot(2,1,1)
```

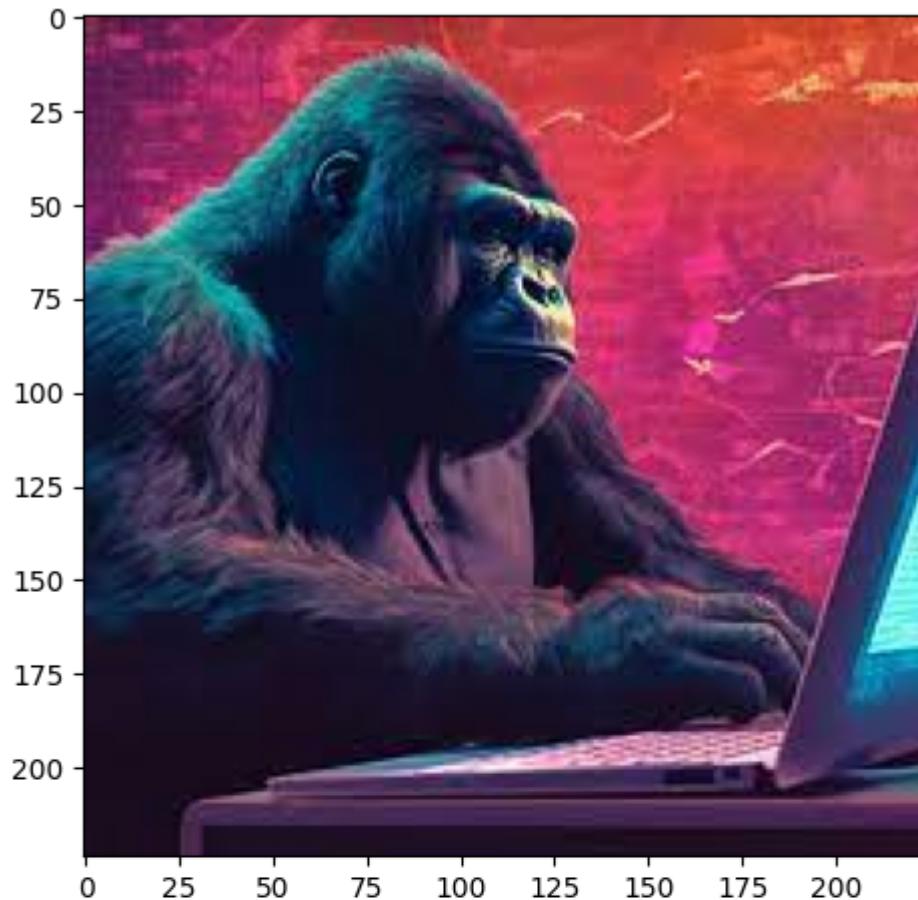
```
imgplot = plt.imshow(orig_img)
a.set_title('Before')

# Subplot for resized image
a=fig.add_subplot(2,1,2)
imgplot = plt.imshow(new_img)
a.set_title('After')

plt.show()
```

Original size: 224 x 224  
New size: 200 x 200

Before



After



0 25 50 75 100 125 150 175

If we want to resize the image and change its shape without distorting it, we'll need to *scale* the image so that its largest dimension fits our new desired size.

```
In [17]: # Get the image size
orig_height, orig_width = orig_img.size
print('Original size:', orig_height, 'x', orig_width)

# We'll resize this so it's 200 x 200
target_size = (150,150)

# Scale the image to the new size using the thumbnail method
scaled_img = orig_img
scaled_img.thumbnail(target_size, Image.ANTIALIAS)
scaled_height, scaled_width = scaled_img.size
print('Scaled size:', scaled_height, 'x', scaled_width)

# Create a new white image of the target size to be the background
new_img = Image.new("RGB", target_size, (255, 255, 255))

# paste the scaled image into the center of the white background image
new_img.paste(scaled_img, (int((target_size[0] - scaled_img.size[0]) / 2), i
new_height, new_width = new_img.size
print('New size:', new_height, 'x', new_width)

# Show the original and resized images
# Create a figure
fig = plt.figure(figsize=(12, 12))

# Subplot for original image
a=fig.add_subplot(3,1,1)
imgplot = plt.imshow(orig_img)
a.set_title('Original')

# Subplot for scaled image
a=fig.add_subplot(3,1,2)
imgplot = plt.imshow(scaled_img)
a.set_title('Scaled')

# Subplot for resized image
a=fig.add_subplot(3,1,3)
imgplot = plt.imshow(new_img)
a.set_title('Resized')

plt.show()
```

Original size: 200 x 200

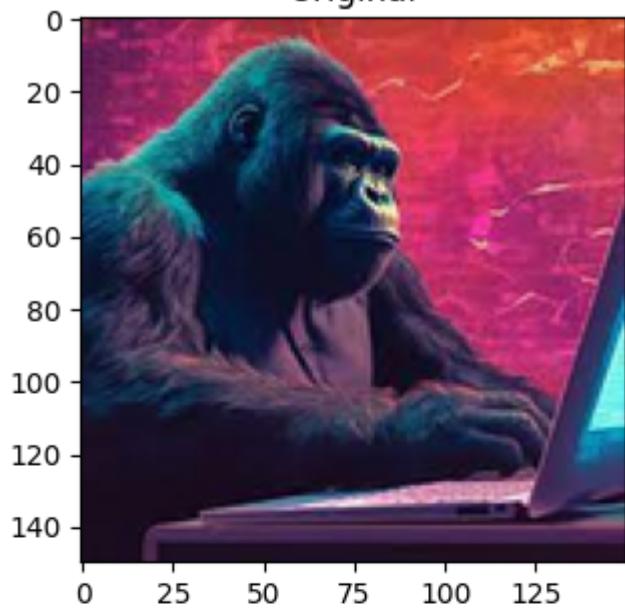
Scaled size: 150 x 150

New size: 150 x 150

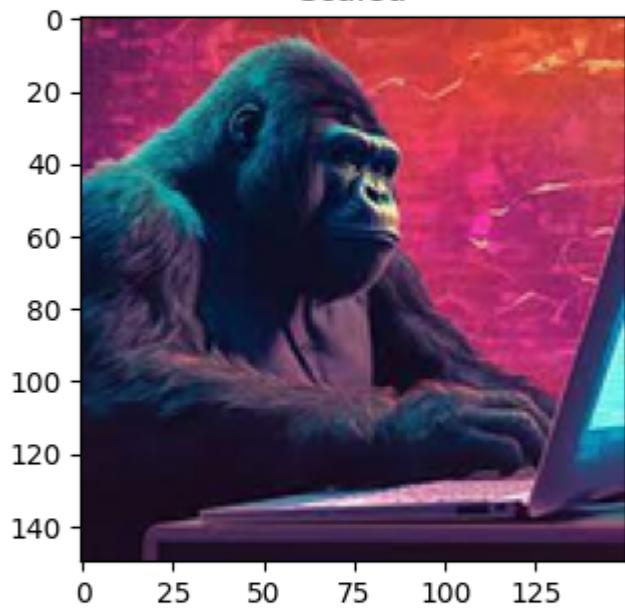
```
/tmp/ipykernel_153889/3445694785.py:10: DeprecationWarning: ANTIALIAS is dep
recated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS or Resamp
ling.LANCZOS instead.
```

```
    scaled_img.thumbnail(target_size, Image.ANTIALIAS)
```

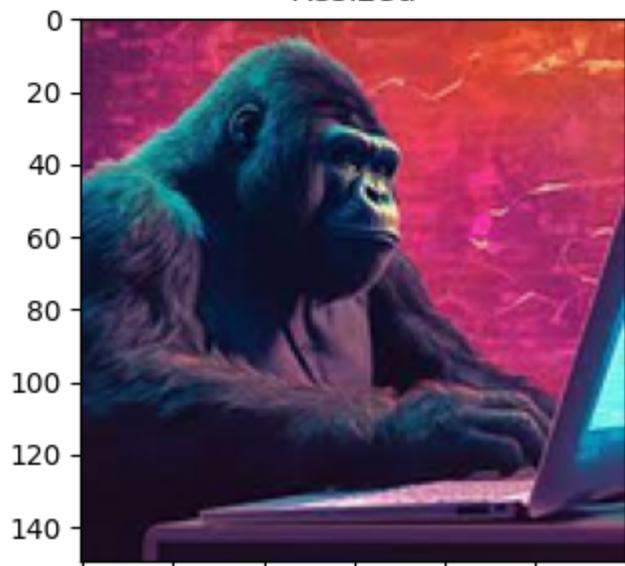
Original



Scaled



Resized



0 25 50 75 100 125

## Negative Images

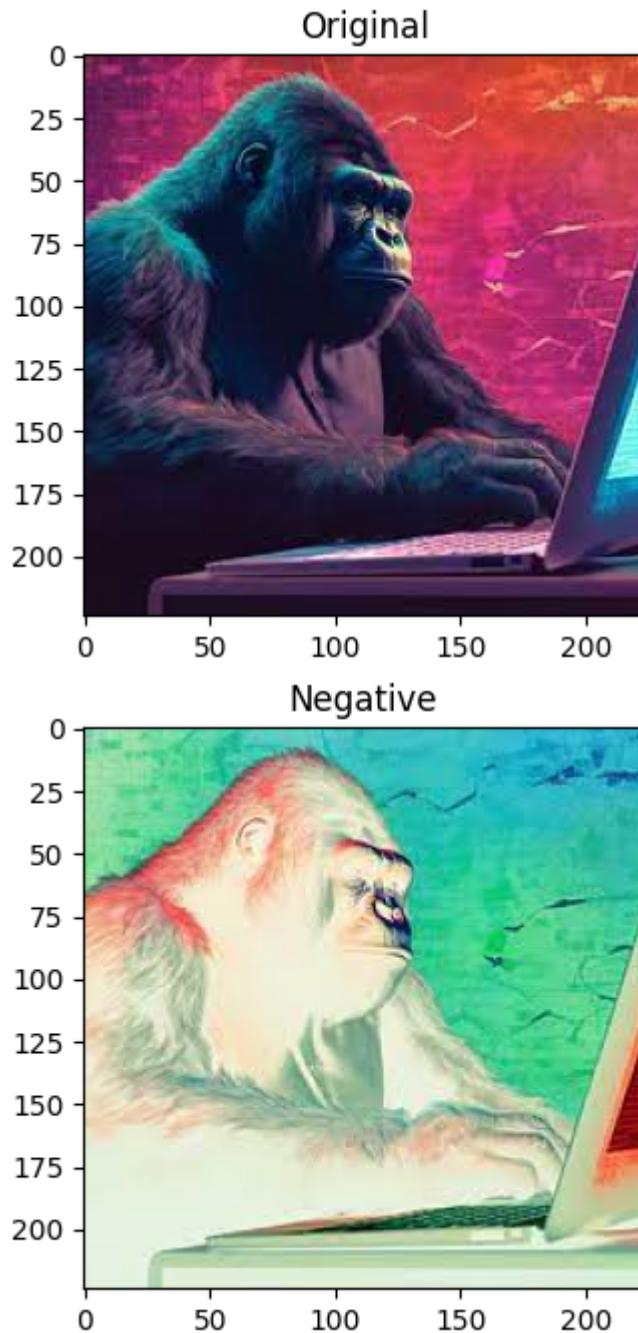
```
In [19]: orig_img = cv2.imread('../data/semana_2/Gorila.jpeg')
orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)
img_neg = 255 - orig_img
```

```
In [20]: fig = plt.figure(figsize=(8, 8))

# Subplot for original image
a=fig.add_subplot(2,1,1)
imgplot = plt.imshow(orig_img)
a.set_title('Original')

a = fig.add_subplot(2,1,2)
imgplot = plt.imshow(img_neg)
a.set_title('Negative')

plt.show()
```



## Logarithmic Transformation

$$S = c * \log(1 + r)$$

where,

- $R$  = input pixel value
- $C$  = scaling constant and
- $S$  = output pixel value

The value of  $c$  is chosen such that we get the maximum output value corresponding to the bit size used. So, the formula for calculating  $c$  is as follows:

$$c = 255 / (\log(1 + \text{max input pixel value}))$$

```
In [23]: orig_img = cv2.imread('../data/semana_2/Gorila.jpeg')
orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

# Apply log transformation method
c = 255 / np.log(1 + np.max(orig_img))
log_img = c * (np.log(orig_img + 1))

# Specify the data type so that
# float value will be converted to int
log_img = np.array(log_img, dtype = np.uint8)
```

```
/tmp/ipykernel_153889/928694418.py:6: RuntimeWarning: divide by zero encountered in log
    log_img = c * (np.log(orig_img + 1))
/tmp/ipykernel_153889/928694418.py:10: RuntimeWarning: invalid value encountered in cast
    log_img = np.array(log_img, dtype = np.uint8)
```

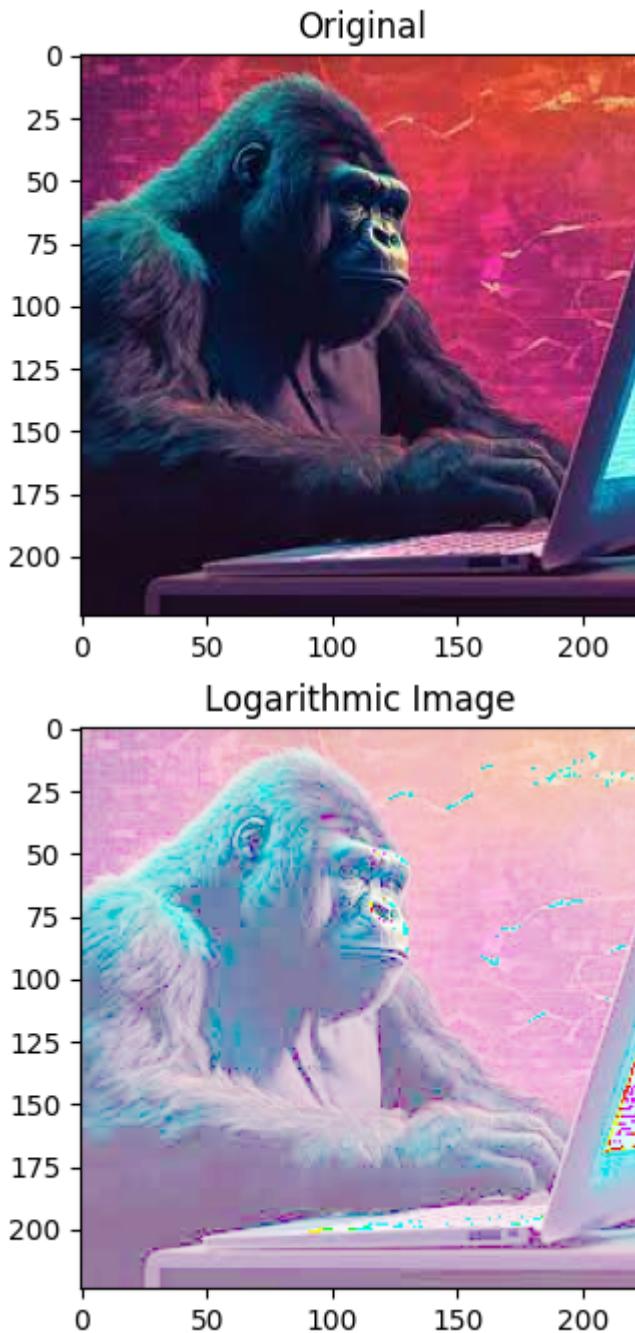
Log transformation of gives actual information by enhancing the image. If we apply this method in an image having higher pixel values then it will enhance the image more and actual information of the image will be lost.

```
In [24]: fig = plt.figure(figsize=(8, 8))

# Subplot for original image
a=fig.add_subplot(2,1,1)
imgplot = plt.imshow(orig_img)
a.set_title('Original')

a = fig.add_subplot(2,1,2)
imgplot = plt.imshow(log_img)
a.set_title('Logarithmic Image')

plt.show()
```



## Image Binarizer (Thresholding)

Binarize pixels (set pixel values to 0 or 1) according to a threshold.

```
In [26]: orig_img = cv2.imread('../data/linear_gradient.png')
orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

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

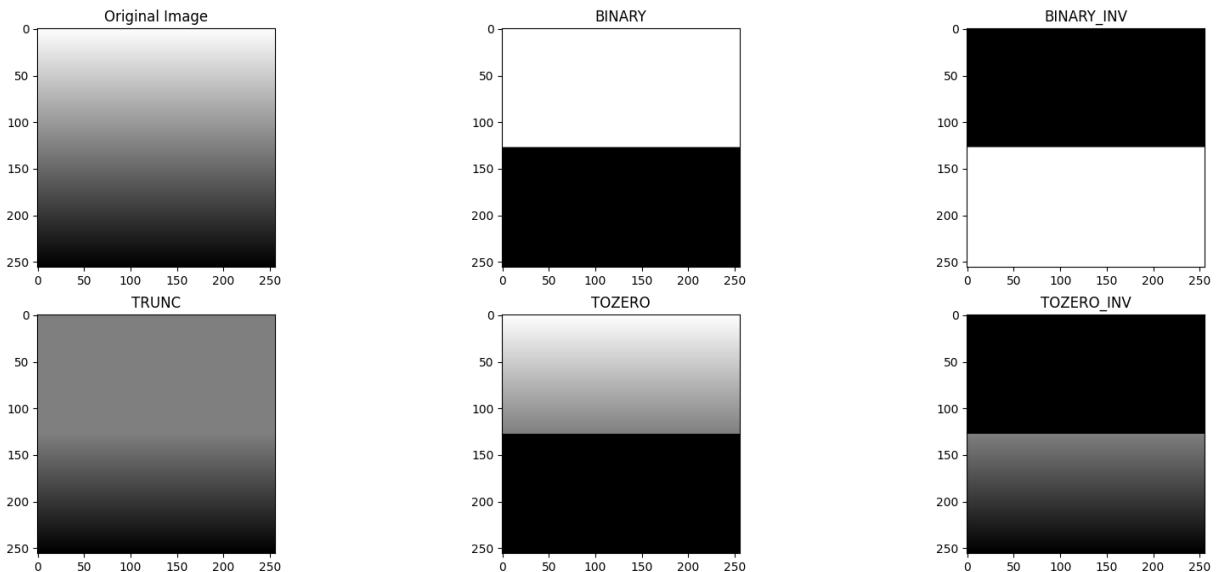
```

titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
images = [orig_img, thresh1, thresh2, thresh3, thresh4, thresh5]
n = np.arange(6)

fig = plt.figure(figsize=(20, 8))
for i in n:
    plt.subplot(2,3,i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])

plt.show()

```



## Image Quantizer

```

In [27]: orig_img = cv2.imread('../data/semana_2/Gorila.jpeg')

Z = orig_img.reshape((-1,3))

# convert to np.float32
Z = np.float32(Z)

# define criteria, number of clusters(K) and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
def colorQuant(Z, K, criteria):

    ret,label,center=cv2.kmeans(Z,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

    # Now convert back into uint8, and make original image
    center = np.uint8(center)
    res = center[label.flatten()]
    res2 = res.reshape((orig_img.shape))
    return res2
res1 = colorQuant(Z, 2, criteria)
res2 = colorQuant(Z, 5, criteria)
res3 = colorQuant(Z, 8, criteria)

fig = plt.figure(figsize=(12, 8))

```

```

plt.subplot(221),plt.imshow(cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

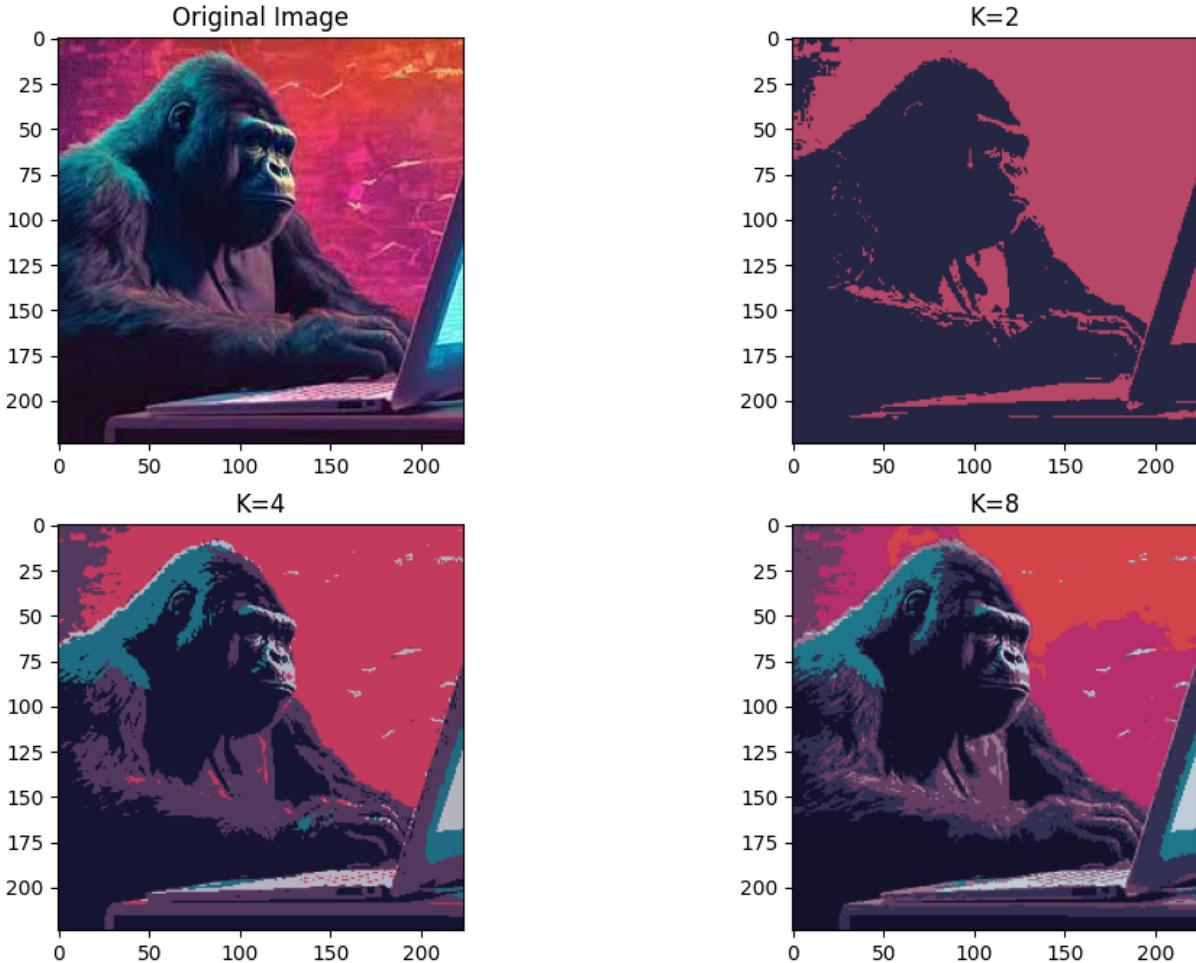
plt.subplot(222),plt.imshow(cv2.cvtColor(res1, cv2.COLOR_BGR2RGB))
plt.title('K=2')

plt.subplot(223),plt.imshow(cv2.cvtColor(res2, cv2.COLOR_BGR2RGB))
plt.title('K=4')

plt.subplot(224),plt.imshow(cv2.cvtColor(res3, cv2.COLOR_BGR2RGB))
plt.title('K=8')

plt.show()

```



In [28]: Z.shape, orig\_img.shape

Out[28]: ((50176, 3), (224, 224, 3))

## 2. Simple Image Processing

### Table of Contents

1. Solución
2. Pregunta 1
  - 1.1. Contrast Stretching
  - 1.2. Gray-level Slicing
  - 1.3. Intensity level slicing
  - 1.4. Masking, thresholgind

3. Pregunta 2

- 2.1. Negative

4. Pregunta 3

- 3.1. Power low transformation

5. Pregunta 4

- 4.1. Sustracción
  - 4.2. Power low
  - 4.3. Adición
  - 4.4. Multiplicación

6. Conclusiones

# Solución de la práctica

## Pregunta 1

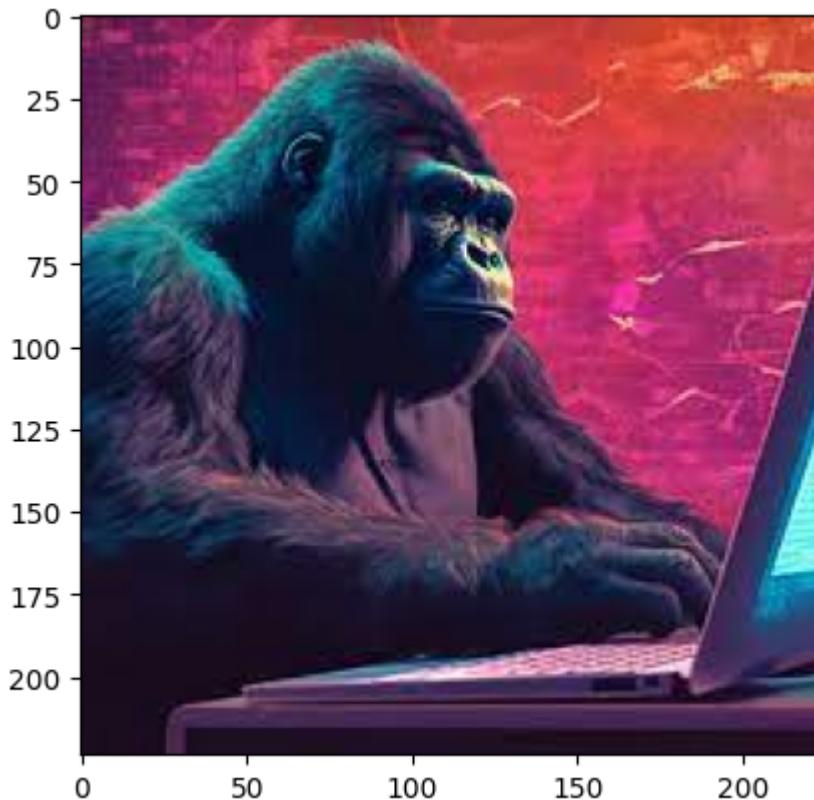
1. Las transformaciones pixel a pixel son sumamente utilizadas para aumentar la cantidad de imágenes para entrenar modelos de inteligencia artificial, sobre todo aquellas de tipo fotométrico. Investiga 3 tipos de transformaciones y aplicarlas en el proyecto de Google Collab sobre imágenes propias.

```
In [2]: from PIL import Image, ImageOps
from matplotlib import image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

A continuación se carga una imagen real la cual fue creada usando modelos de inteligencia artificial o los famosos Large vision models

```
In [4]: img = mpimg.imread('./data/semana_2/Gorila.jpeg')
plt.imshow(img)
```

```
print(type(img))  
<class 'numpy.ndarray'>
```



## Contrast Stretching

Este método amplía el rango de intensidades, aumentando el rango dinámico de los niveles de gris en la imagen. Las imágenes de bajo contraste debido a una iluminación inadecuada se pueden mejorar con este método, aumentando así el contraste general.

```
In [6]: def Contrast_stretch(p, r1, s1, r2, s2):  
    if (0 <= p and p <= r1):  
        equation = (s1 / r1)*p  
    elif (r1 < p and p <= r2):  
        equation = ((s2 - s1)/(r2 - r1))*(p - r1)+s1  
    else:  
        equation = ((255 - s2)/(255 - r2))*(p - r2)+s2  
    return equation  
  
# Initialize range  
r1 = 55  
s1 = 40  
r2 = 140  
s2 = 200  
  
pixelVal_vec = np.vectorize(Contrast_stretch)
```

```
# Contrast stretching
contrast = pixelVal_vec(img, r1, s1, r2, s2)
```

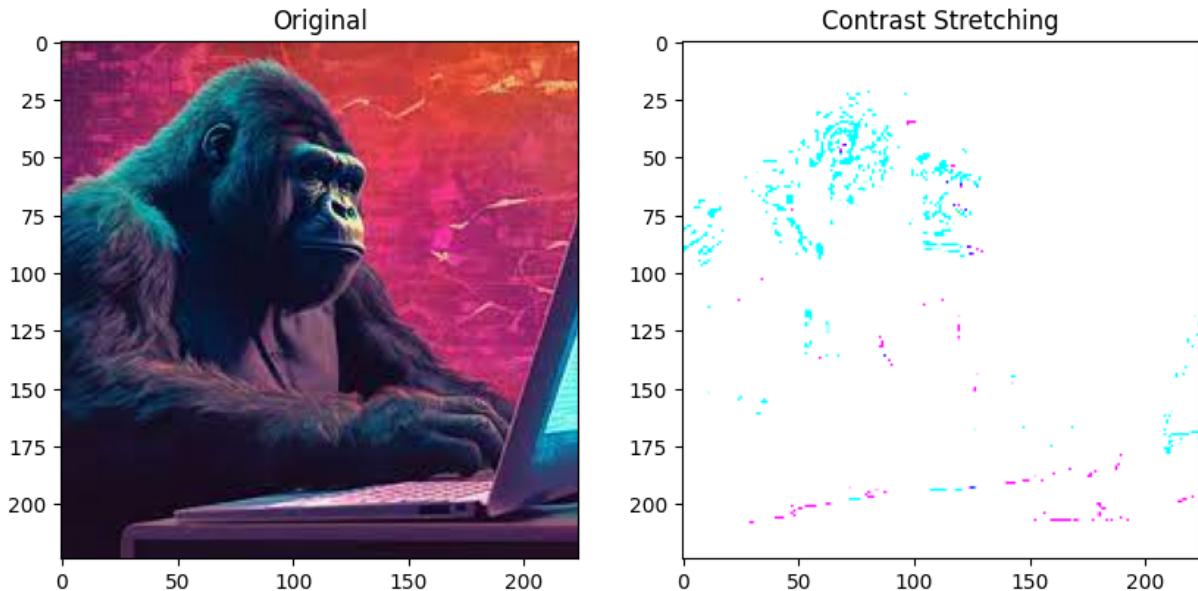
```
In [7]: fig = plt.figure(figsize=(10, 10))

# Subplot for original image
a=fig.add_subplot(121)
imgplot = plt.imshow(img)
a.set_title('Original')

a = fig.add_subplot(122)
imgplot = plt.imshow(contrast)
a.set_title('Contrast Stretching')

plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



## Gray-level Slicing

Se centra en mejorar un rango específico de niveles de grises en una imagen. Los intervalos están predefinidos y se manipulan los píxeles que se encuentran en ese rango. Esto se puede utilizar para iluminar el rango deseado de niveles de grises y al mismo tiempo preservar la calidad del fondo en el rango.

```
In [8]: # Find width and height of image
img_test = img[:, :, 0]
row, column = img_test.shape

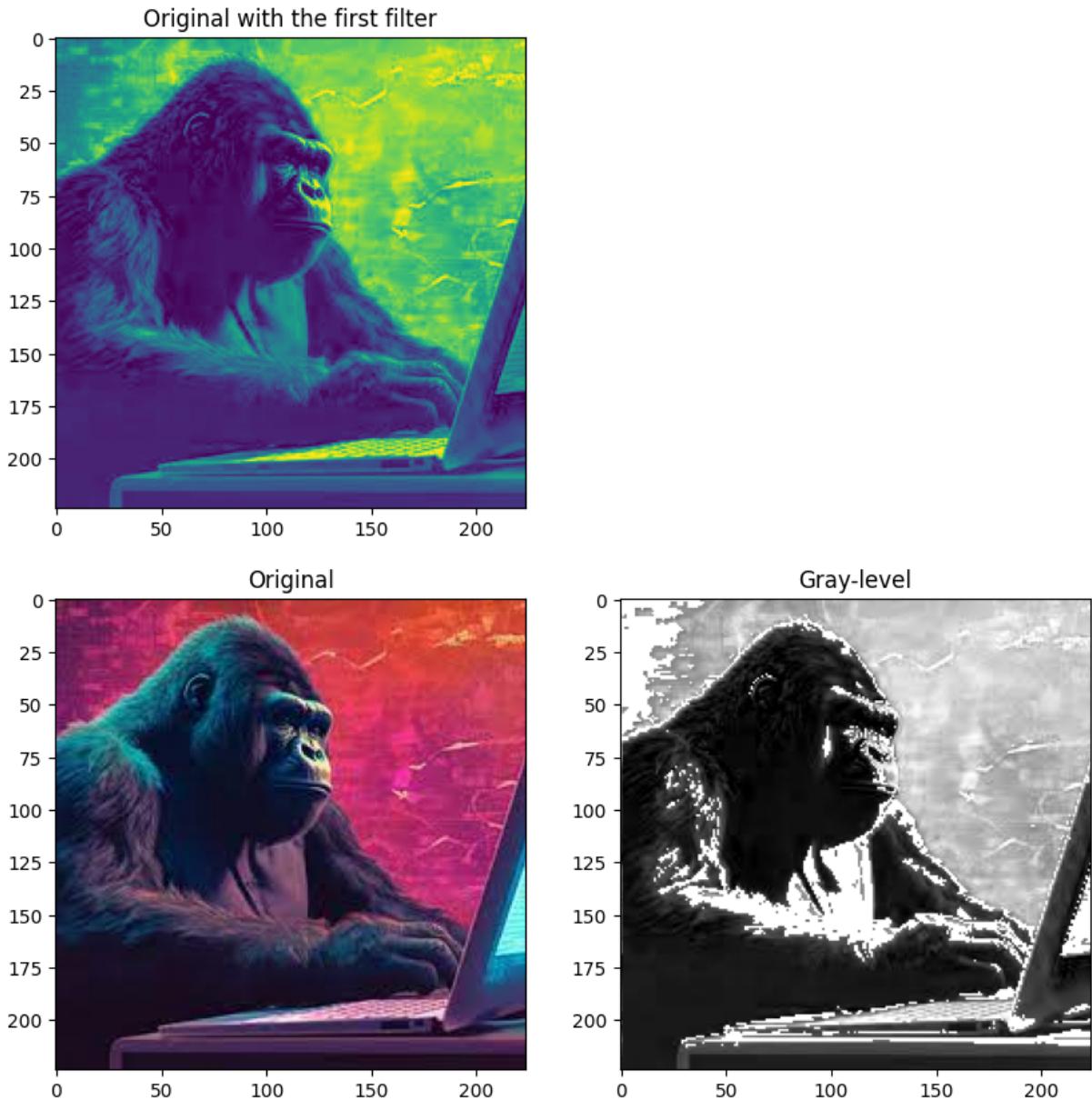
# Create an zeros array to store the sliced image
img1 = np.zeros((row,column),dtype = 'uint8')

# Specify the min and max range
```

```
min_range = 80  
max_range = 140
```

```
In [9]: # Recorra la imagen de entrada y, si el valor de píxel se encuentra en el rango deseado, configúrelo al valor deseado  
# de lo contrario, configúrelo al valor deseado  
for i in range(row):  
    for j in range(column):  
        if img_test[i,j] > min_range and img_test[i,j] < max_range:  
            img1[i,j] = 255  
        else:  
            img1[i,j] = img_test[i-1,j-1]
```

```
In [10]: fig = plt.figure(figsize=(10, 20))  
  
# Subplot for original image  
a=fig.add_subplot(121)  
imgplot = plt.imshow(img)  
a.set_title('Original')  
  
a = fig.add_subplot(122)  
imgplot = plt.imshow(img1, cmap="gray")  
a.set_title('Gray-level')  
  
a = fig.add_subplot(221)  
imgplot = plt.imshow(img_test)  
a.set_title('Original with the first filter')  
  
plt.show()
```



## Intensity level slicing

El método, llamado corte por niveles de intensidad, se puede implementar de varias maneras, pero la mayoría son variaciones de dos temas básicos. Un enfoque es mostrar en un valor (digamos, blanco) todos los valores en el rango de interés y en otro (digamos, negro) todas las demás intensidades.

```
In [11]: img1 = img/2
img2 = 2 * img
img3 = img**2

dict_images = {'Halve the intensity':img1, 'Double the intensity':img2, 'Squ
```

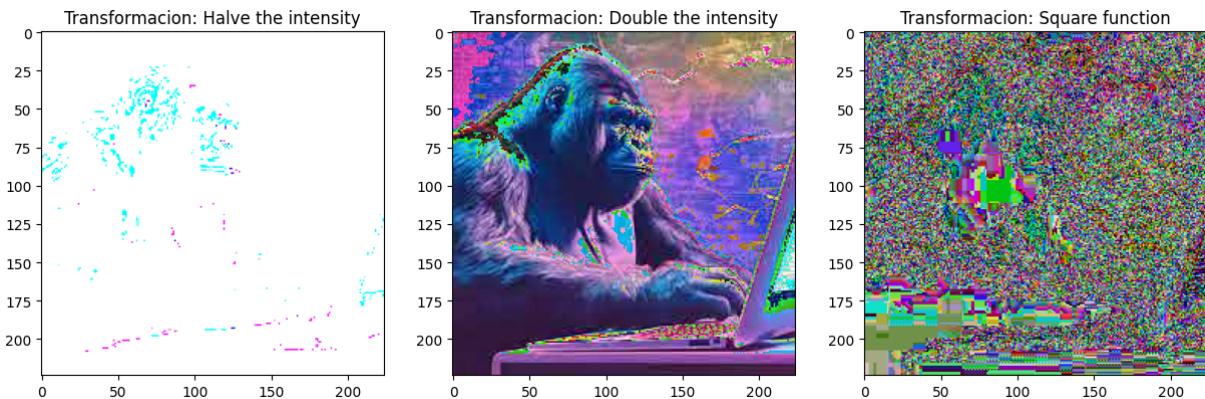
```
In [34]: def display(dict=None, nrow=None, ncol=None):
    fig = plt.figure(figsize=(10, 10))
```

```
for raw, col in enumerate(dict):

    plt.subplot(nrow,ncol,raw+1)
    plt.title(f'Transformacion: {col}')
    plt.imshow(dict[col])
```

```
In [22]: display(dict_images, nrow=1, ncol=3)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



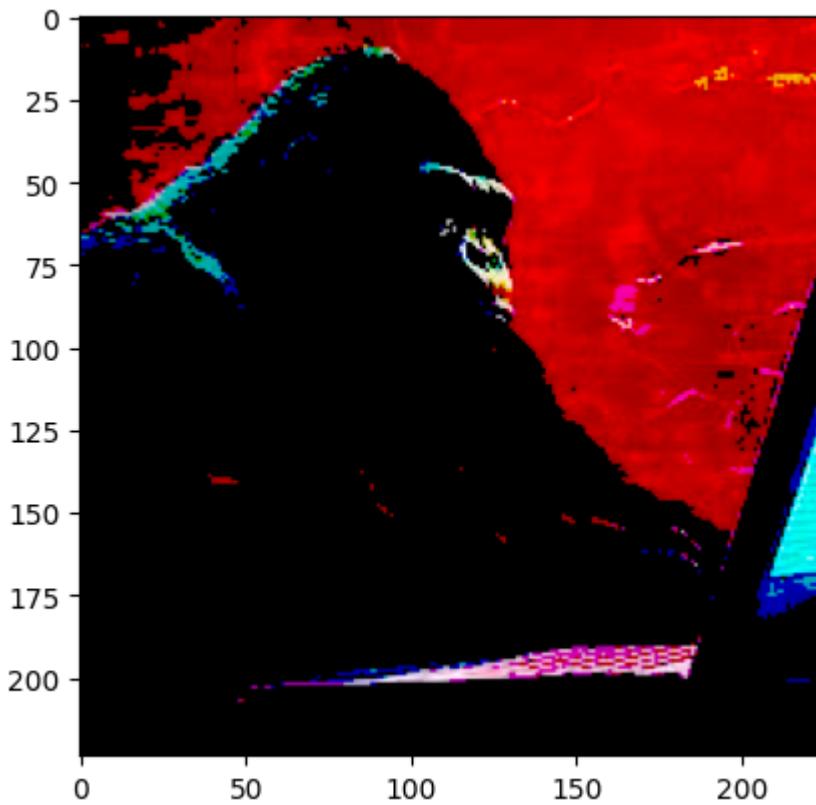
## Masking, thresholding

Pruebas binarias para enmascar imágenes usando un umbral respectivo

```
In [24]: img4 = img.copy()
img4[img4 < 150] = 0
```

```
In [25]: plt.imshow(img4)
print(type(img4))
```

```
<class 'numpy.ndarray'>
```



## Pregunta 2

2. Investiga una aplicación donde obtener el negativo de imagen tenga un valor específico e integra el código en en una fila de google collab, justificar brevemente tu investigación y haciendo una demo sencilla..

## Negative

```
In [36]: img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.imread('/home/yvillamil/Pictures/Gorila.jpeg', 0)

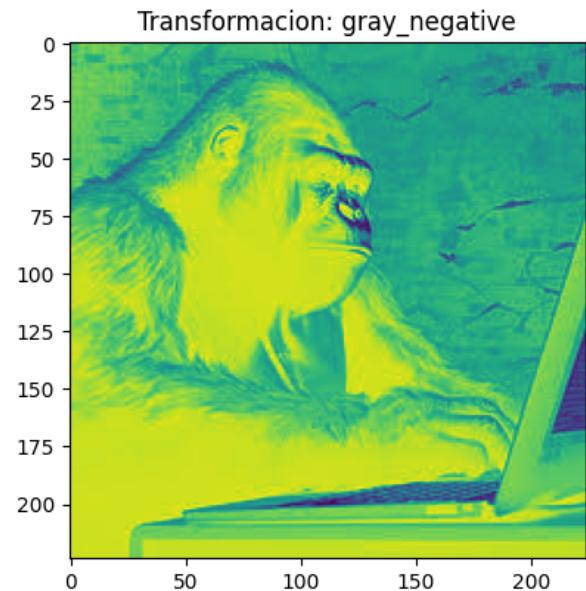
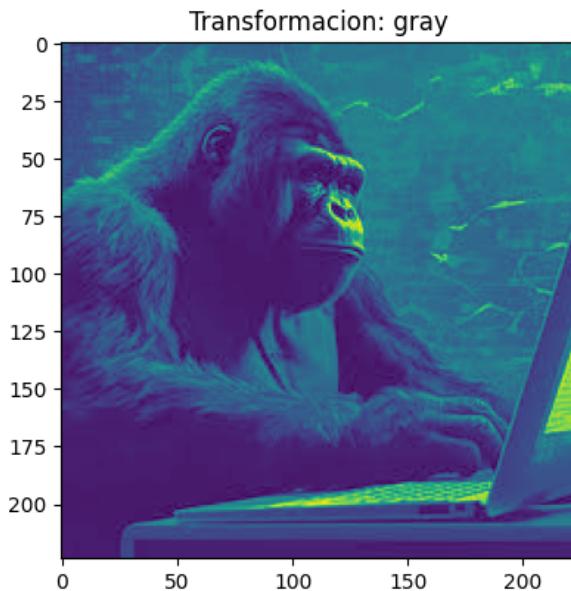
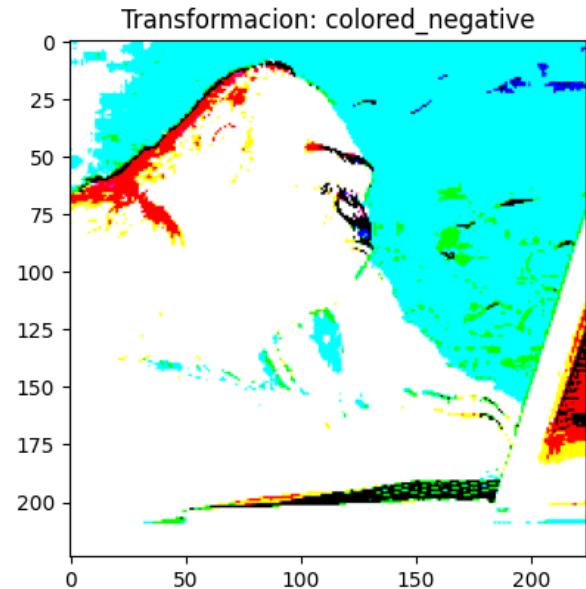
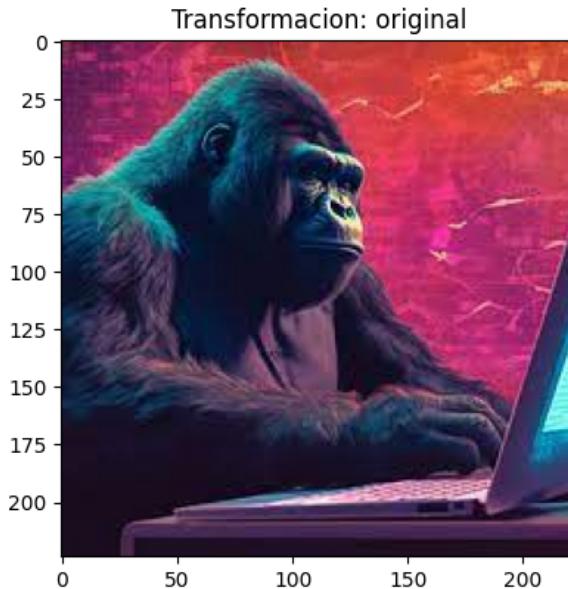
# Max intensity based on quantization
max_pixel = img1.max()

colored_negative = max_pixel-img
gray_negative = max_pixel-img_gray

dict_negative = {
    'original':img,
    'colored_negative': colored_negative,
    'gray': img_gray,
    'gray_negative': gray_negative
}
```

```
In [37]: display(dict_negative, nrow=2, ncol=2)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



## Pregunta 3

3. Investiga una aplicación donde se puede aplicar la corrección de gamma en una imagen. Integra el código en en una fila de google collab, justifica brevemente tu investigación y haz una demo sencilla.

## Power low transformation

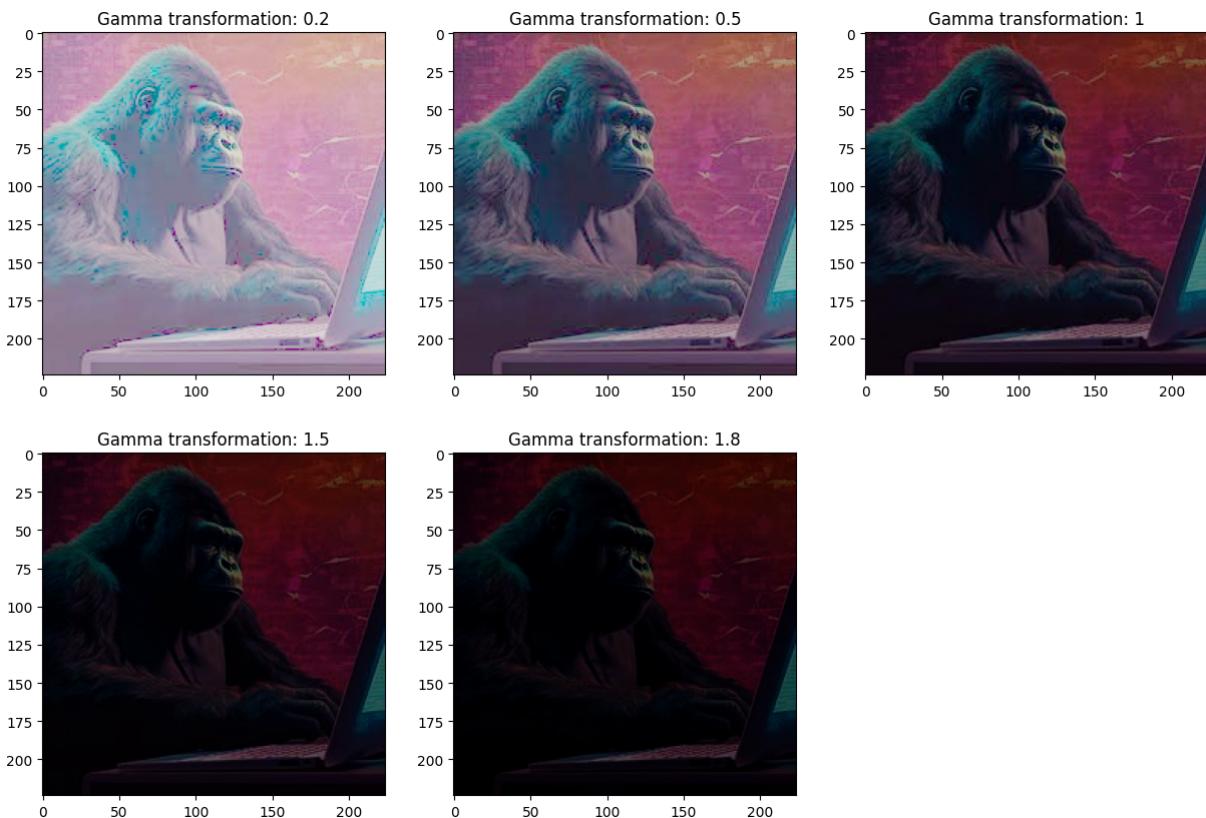
$$S = c * r^\gamma$$

También conocida como correlación gamma, se ocupa de los niveles gamma ( $\gamma$ ) en las imágenes, ampliamente utilizada para mejorar el contraste. La ecuación de transformación es como se muestra a continuación.

```
In [229...]: fig = plt.figure(figsize=(15, 10))
rows = 2
for raw, gamma in enumerate([0.2, 0.5, 1, 1.5, 1.8]):
```

*# Apply gamma correction.*  
gamma\_transformation = np.array(255\*(img1 / 255) \*\* gamma, dtype = 'uint8')

```
plt.subplot(rows, 3, raw+1)
plt.title(f'Gamma transformation: {gamma}')
plt.imshow(gamma_transformation)
```



## Pregunta 4

**4. Investiga una aplicación donde se puede usar la sustracción de imágenes e integra el código en en una fila de google collab, justificar brevemente tu investigación, haciendo una demo sencilla.**

Adicional a la substracción, se presentarán una serie de operaciones aritméticas que pueden ser realizadas con OpenCV

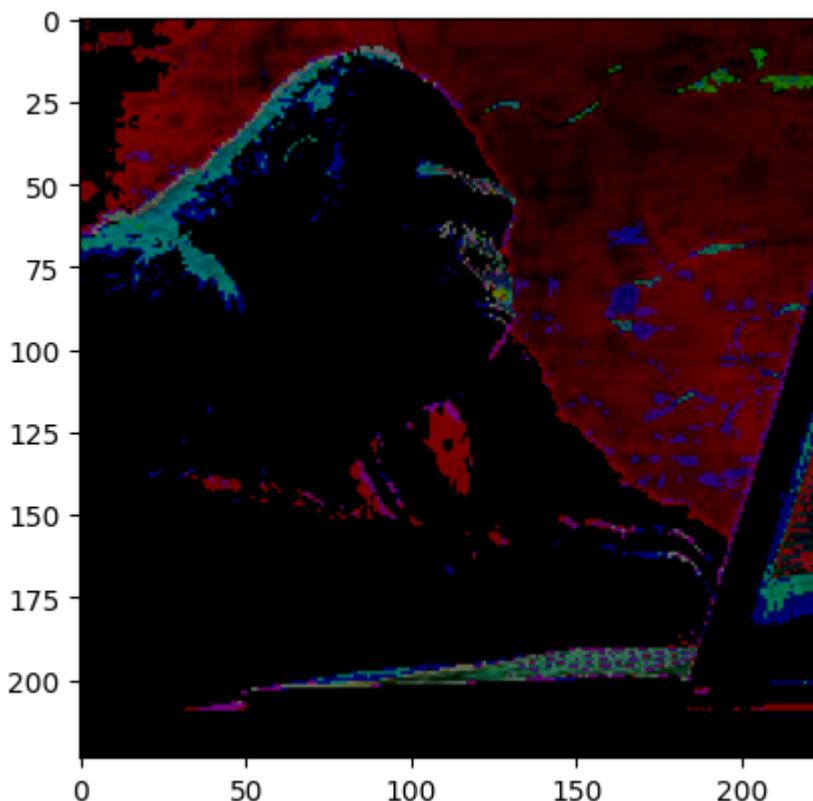
### Sustracción

A continuación se muestra como sustraer los valores de la imagen original y una imagen que fue generada en el paso de operaciones de intensidad (específicamente el doble de la intensidad)

```
In [197... img_sub = cv2.subtract(img, img2)
```

```
In [198... plt.imshow(img_sub)
```

```
Out[198... <matplotlib.image.AxesImage at 0x7f583a359d50>
```

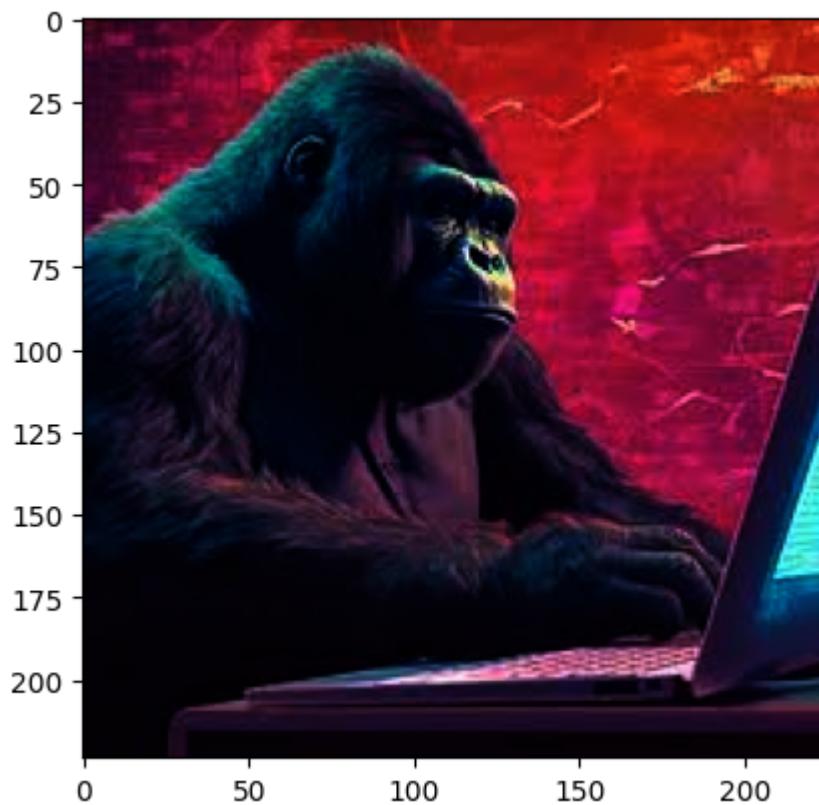


## Power law

Usando openCV es posible crear funciones del tipo transformaciones a la potencia para la imagen original

```
In [240... img_plot = img.copy()
pow_law = np.array(cv2.pow((img_plot / img_plot.max()), 1.8) * img_plot.max())
plt.imshow(pow_law)
```

```
Out[240... <matplotlib.image.AxesImage at 0x7f5837942cd0>
```

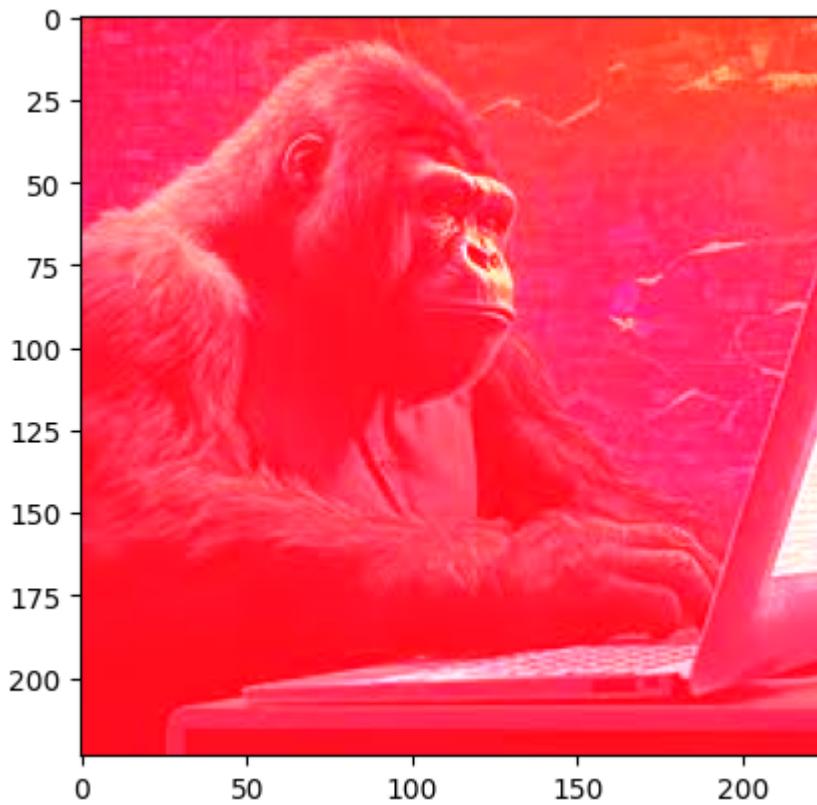


## Adición

Es adicionado un valor a cada pixel mediante sumatoria. Se observan los cambios de color de la imagen original

```
In [244...]: img_sub = cv2.add(img, 252)  
plt.imshow(img_sub)
```

```
Out[244...]: <matplotlib.image.AxesImage at 0x7f583785d7d0>
```



## Multiplicación

Calcula el producto escalado por elemento de dos matrices o una matriz con un escalar

```
In [273...]: i = cv2.multiply(img, 1, dtype=-1, scale=2)
plt.imshow(np.array(i, dtype='uint8'))
```

```
Out[273...]: <matplotlib.image.AxesImage at 0x7f58373f57d0>
```



## Conclusiones

1. Se han implementando tres operaciones básicas de punto (point operations), las cuales permiten realizar cambios directos en los valores de pixeles, permitiendo la transformación de las imágenes. Las usadas y expuestas en código son las siguientes:
  - Contrast Stretching: Permite cambios de intensidad altas en las imágenes en escala de grises. En la imagen que vemos representada se observa que el color de la transformación es muy diferente al original a lo esperado en los ejemplo pero esto es debido a la iluminación de la imagen.
  - Gray-level: De manera sencilla se pueden observar las escalas de grises de la imagen original y se expone una imagen de un solo canal (siempre el primero) y su escala es grises.
  - Nivel de intensidades: Por medio de operaciones sencillas es posible cambiar las intensidades de imágenes, tales como el doble de los pixeles o

la raíz cuadrada. Siendo esta última la que genera cambios que se evidencian de manera significativa en la figura que lo representa.

2. El negativo tiene una forma sencilla se extraerse y es por medio del valor máximo de pixeles y restando cada uno de estos con este valor. Para dar más valor se muestra el negativo a color y en escala de grises sombras que en el caso de escala de grises no es posible identificar a grandes rasgos y en su negativo da más brillo y de igual manera, para el caso coloridad se muestra que aquellas que son más brillantes de la imagen original esta tiende a ser más oscura en su negativo. Transformación interesante que puede ser de valor en proceso de desarrollos analíticos específicos.
3. Para la transformación `power law` se considera de manera sencilla usar una operación de arrays para poder calcular lo más conocido como correlación gamma. Esta permite ver los cambios de contraste de la imagen original, haciendola más intensa cuando el valor del exponente es mayor.
4. Finalmente y mediante el uso de openCV, se emplean funciones propias de la librería o solo para realizar una sustracción de pixeles, aquí también son presentando ejemplos para potencias, multiplicación, adición donde se evidencia las transformaciones y contraste que toma la imagen para ciertos detalles como el caso de la multiplicación de matriz-matriz o matriz-vector. La parte de la nariz tiene detalles interesantes que pueden ser resaltados.