UNIVERSIDAD NACIONAL DE TRUJILLO

Facultad de ingeniería
Programa de ingeniería mecatrónica

LABORATORIO 3 – OpenCV y Operaciones de Punto

PROCESAMIENTO DIGITAL DE SEÑALES E IMÁGENES

ESTUDIANTE(S) :
CORTEZ GOMEZ BRAJAN LEONEL
ESPINOZA LEÓN KARL ALEJANDRO
GUTIÉRREZ GUTIÉRREZ ITALO AARÓN

DOCENTE :
MS. ING. EMERSON MÁXIMO ASTO RODRIGUEZ

CICLO :
2023 - II

Trujillo, Perú
2023

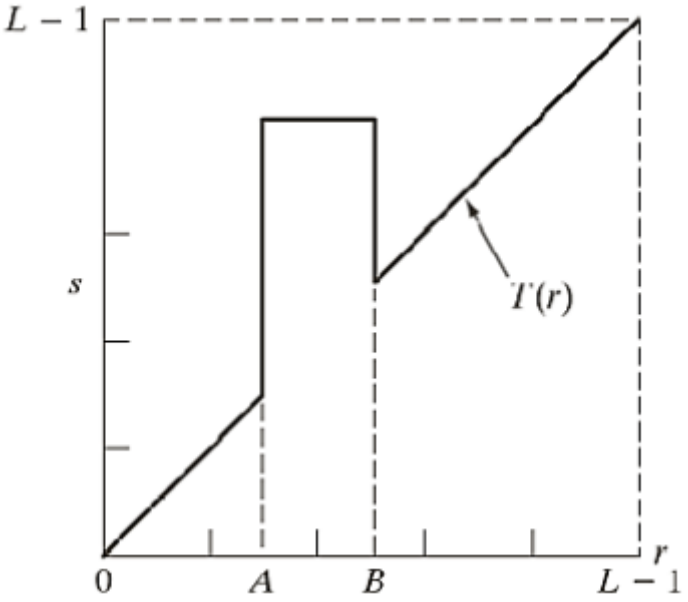
RESUMEN

En el presente informe se resuelven ejercicios de procesamiento de imágenes, siendo estos la modificación de una imagen usando una función arbitraria, encriptación de un mensaje en una imagen, la especificación de dos histogramas de dos imágenes, así como averiguar sobre histograma en 2d con OpenCV, definición de una función para expandir el rango y una función para aumentar el brillo de una imagen.

ÍNDICE

1. RESULTADOS ESPERADOS DE LA EXPERIENCIA

a. Implementación de una función con la forma mostrada en la siguiente figura para modificar una imagen arbitraria.



Resolución:

Tenemos como imagen original a la siguiente imagen:

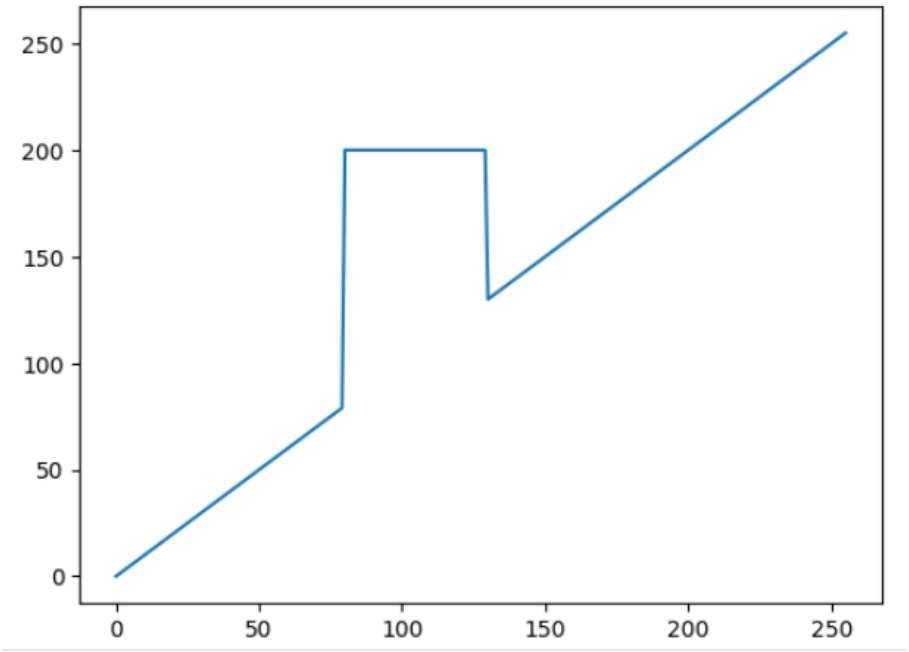
Implementamos la función en un código de Python.
Importamos las librerías “cv2”, “numpy”, “matplotlib.pyplot” en ese orden.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Creamos una función de mapeo de intensidad en una matriz Numpy, luego visualizamos la función de mapeo en un gráfico.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

funcion = np.arange(256)
funcion[80:130] = 200

plt.plot(funcion)
```

Ejecutando el código (con los datos asignados, es idéntica a la gráfica mostrada en el problema):



Asignamos la imagen a modificar y la leemos, además declaramos una variable que será la imagen de salida, es decir, la muestra de la imagen modificada.

```
imgYN = 'yourname.jpg'
img = cv2.imread(imgYN,0)
img_out = np.empty_like(img)

plt.plot(funcion)
```

Mostramos la forma de la imagen en una tupla, con el código img.shape, luego en un bucle for, recorremos la variable 'm' por las filas de la imagen, después recorremos la

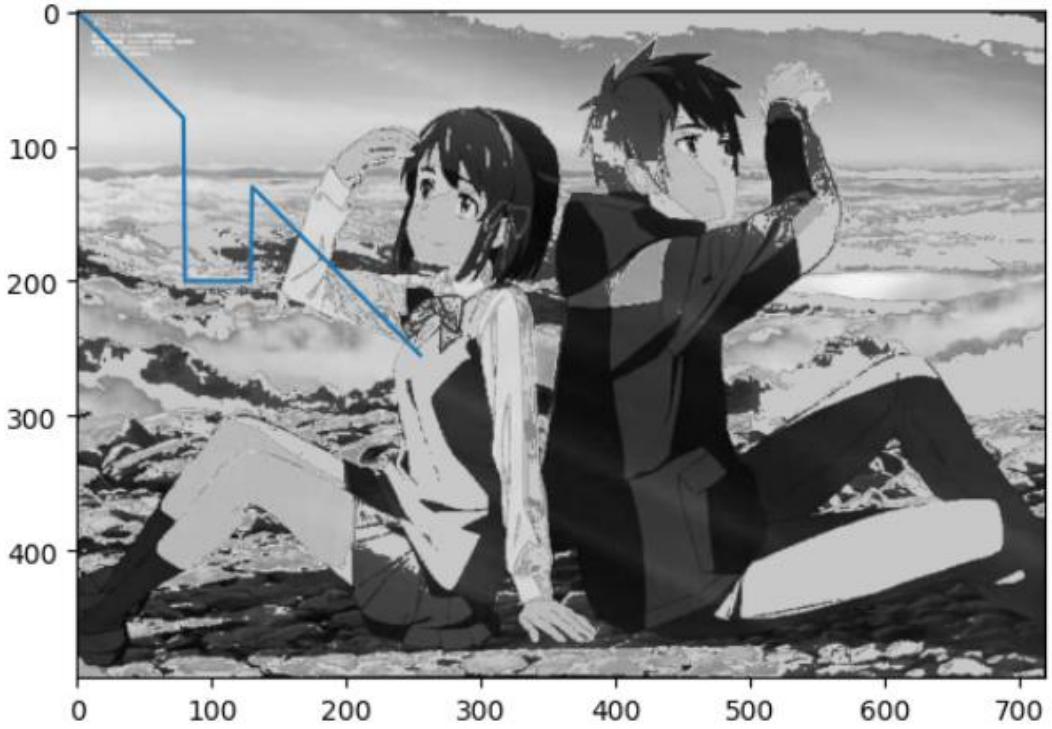
variable 'n' por las columnas de la misma imagen para que se pueda ejecutar la función de mapeo a cada píxel de la imagen con el código que está dentro del segundo bucle for, el cual mostrará los resultados en la variable de imagen de salida.

```
img_out = np.empty_like(img)

img.shape

for m in range(img.shape[0]):
  for n in range(img.shape[1]):
    img_out[m,n] = funcion[img[m,n]]
```

Mostramos la imagen de salida con la escala de grises.

```
plt.plot(funcion)
plt.imshow(img_out, cmap = "gray", vmin=0, vmax=255)
```

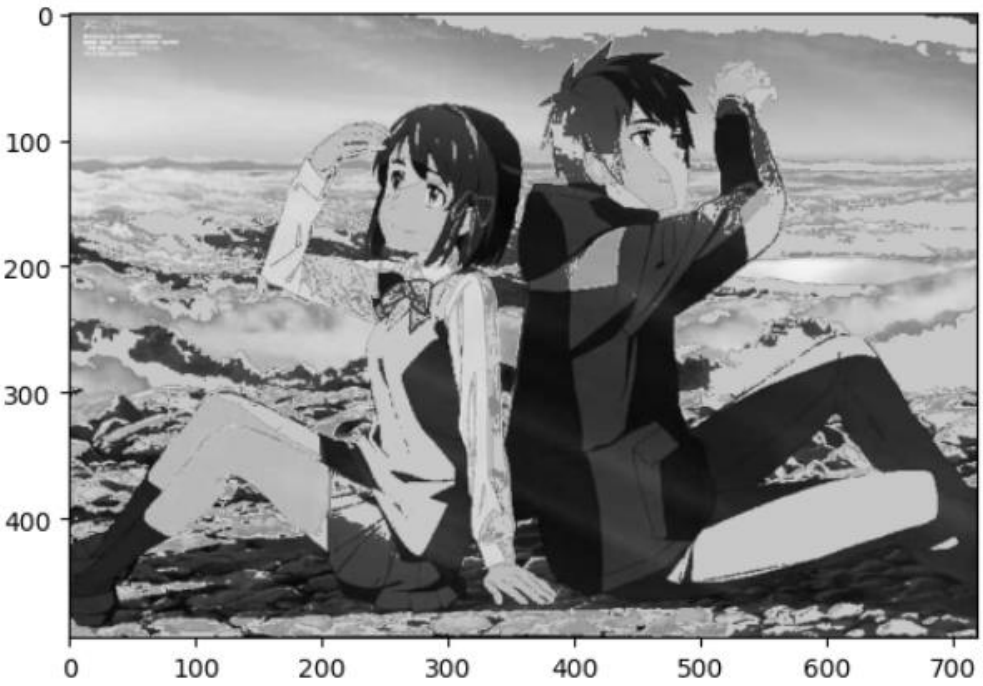
Ejecutando el código:



Para que no se note la línea que representa la gráfica de la función en sentido contrario, comentamos la siguiente línea de código:

```
2 s
#plt.plot(funcion)
plt.imshow(img_out, cmap = "gray", vmin=0, vmax=255)
```

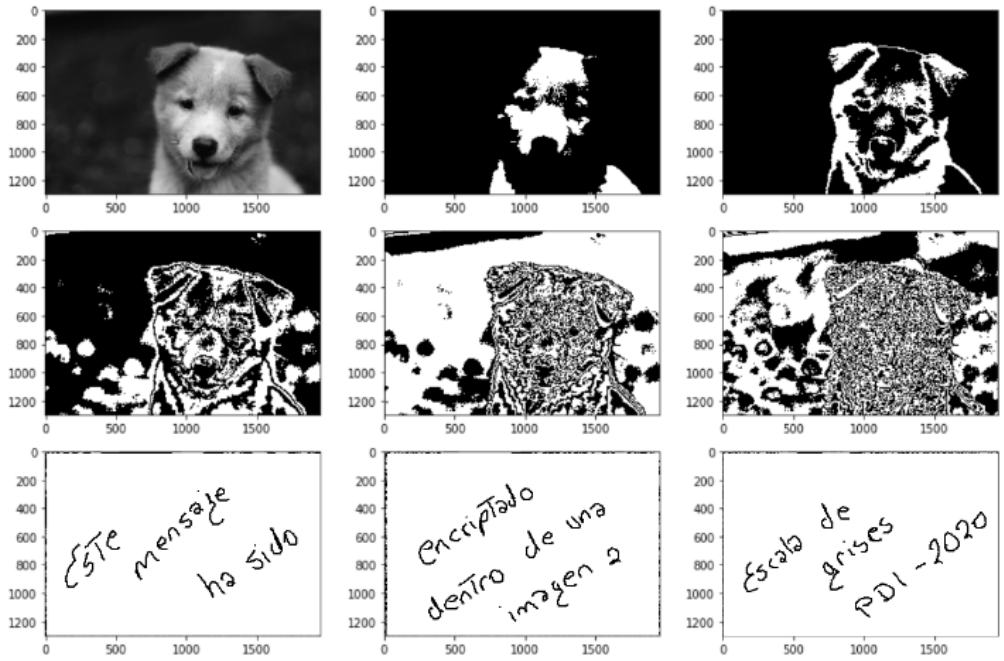
La imagen modificada es la siguiente:



El código completo es el siguiente:

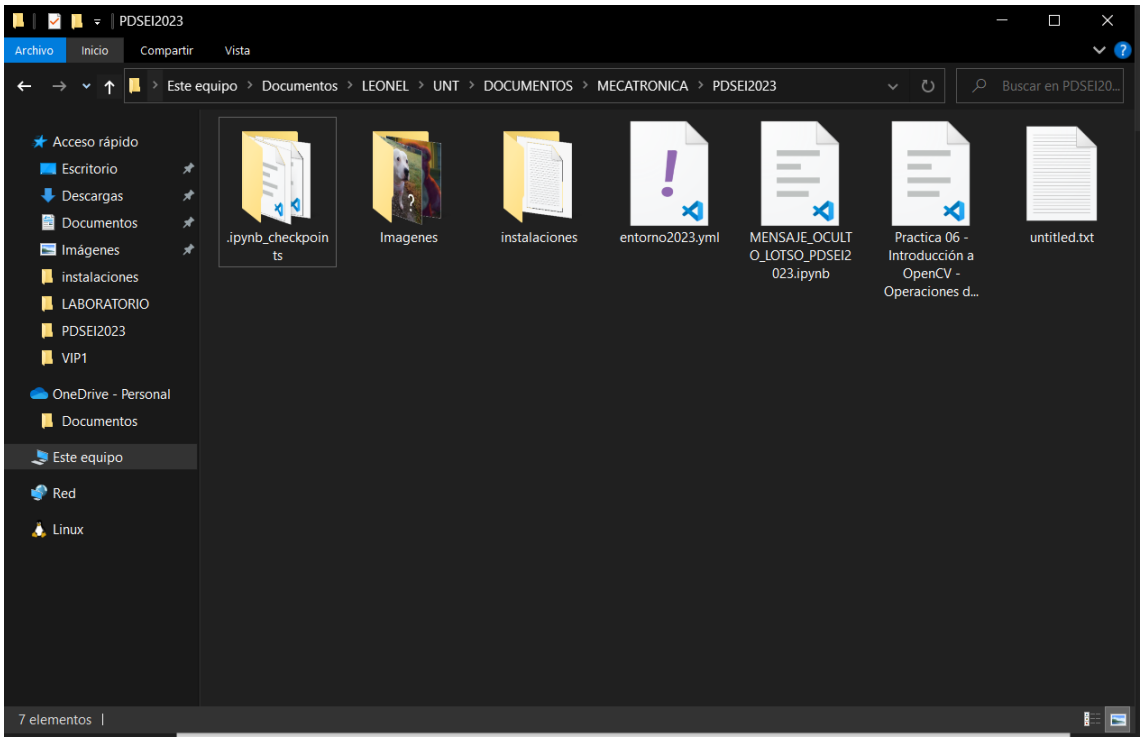
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

funcion = np.arange(256)
funcion[80:130] = 200

imgYN = 'yourname.jpg'
img = cv2.imread(imgYN,0)
img_out = np.empty_like(img)

img.shape

for m in range(img.shape[0]):
  for n in range(img.shape[1]):
    img_out[m,n] = funcion[img[m,n]]


#plt.plot(funcion)
plt.imshow(img_out, cmap = "gray", vmin=0, vmax=255)
```

b. Encriptar un mensaje dentro de una imagen usando descomposición en capa de bits.

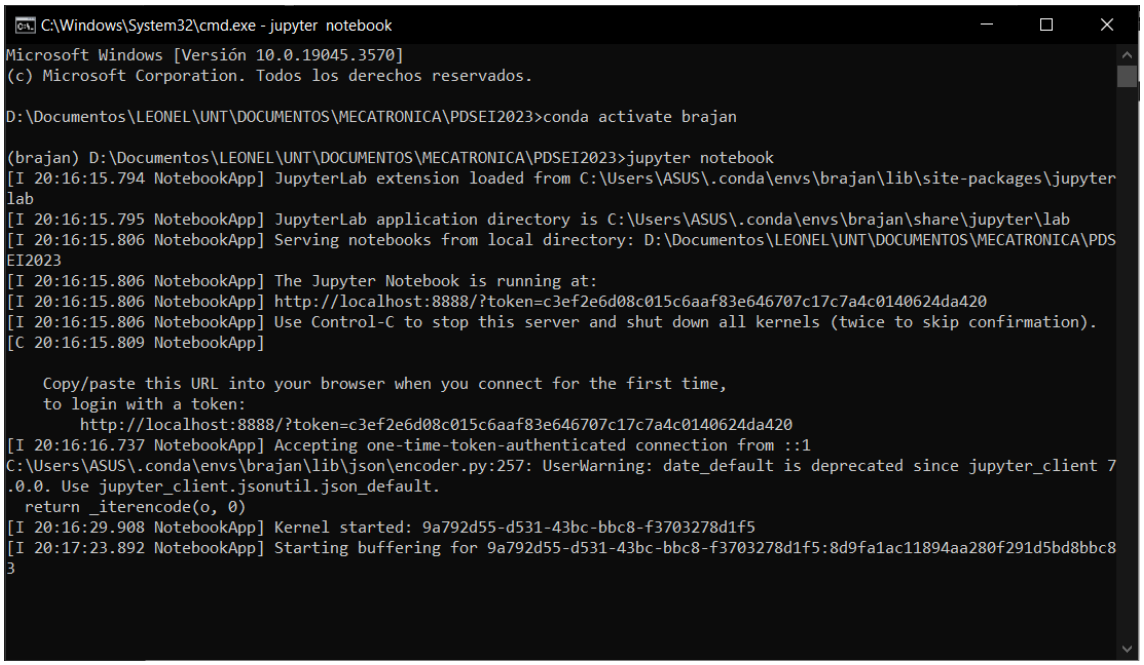


Resolución:

Para esto, después de tener todos los requerimientos previos necesarios para el funcionamiento, abrimos el cmd en la carpeta donde se encuentre nuestro entorno virtual



Y con eso, una vez abierta la ventana de comandos, escribimos conda actívate –name” seguido del nombre de nuestro entorno y luego una vez ubicados en ella, abrimos nuestro “jupyter notebook”:



Se adjunta el código:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Importamos la imagen:
imagen =
cv2.imread("/Documentos/LEONEL/UNT/DOCUMENTOS/MECATRONICA/PDSEI2023/imagenes/lotso.jpg", 0)

# Creamos una imagen blanca con las dimensiones de nuestra imagen
imagen_w = np.ones(imagen.shape, dtype=np.uint8) * 255

# Descomponemos la imagen en 8 capas:
imgg_b7 = np.bitwise_and(imagen, 128)
imgg_b6 = np.bitwise_and(imagen, 64)
imgg_b5 = np.bitwise_and(imagen, 32)
imgg_b4 = np.bitwise_and(imagen, 16)
imgg_b3 = np.bitwise_and(imagen, 8)
imgg_b2 = np.bitwise_and(imagen, 4)
imgg_b1 = np.bitwise_and(imagen, 2)
imgg_b0 = np.bitwise_and(imagen, 1)

# Descomponemos la imagen blanca en 8 capas:
imgw_b7 = np.bitwise_and(imagen_w, 128)
imgw_b6 = np.bitwise_and(imagen_w, 64)
imgw_b5 = np.bitwise_and(imagen_w, 32)
imgw_b4 = np.bitwise_and(imagen_w, 16)
imgw_b3 = np.bitwise_and(imagen_w, 8)
imgw_b2 = np.bitwise_and(imagen_w, 4)
imgw_b1 = np.bitwise_and(imagen_w, 2)
```

```
imgw_b0 = np.bitwise_and(imagen_w, 1)

# Escribimos el texto en las primeras 3 capas de la imagen blanca:
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(imgw_b2, "desde aquel dia", (200, 350), font, 3, (0, 255, 255), 8)
cv2.putText(imgw_b1, "algo cambio", (250, 350), font, 3, (0, 255, 255), 8)
cv2.putText(imgw_b0, "dentro de lotso :'v'", (200, 350), font, 3, (0, 255, 255), 8)
# Intercambiamos las primeras 3 capas de la imagen con las de la imagen blanca
imgg_b2 = imgw_b2
imgg_b1 = imgw_b1
imgg_b0 = imgw_b0

# Imprimimos todas las capas de la imagen
plt.figure(figsize=(15, 10))
plt.subplot(331)
plt.imshow(imagen, cmap="gray")
plt.subplot(332)
plt.imshow(imgg_b7, cmap="gray")
plt.subplot(333)
plt.imshow(imgg_b6, cmap="gray")
plt.subplot(334)
plt.imshow(imgg_b5, cmap="gray")
plt.subplot(335)
plt.imshow(imgg_b4, cmap="gray")
plt.subplot(336)
plt.imshow(imgg_b3, cmap="gray")
plt.subplot(337)
plt.imshow(imgw_b2, cmap="gray")
plt.subplot(338)
plt.imshow(imgg_b1, cmap="gray")
plt.subplot(339)
plt.imshow(imgg_b0, cmap="gray")
plt.show()

plt.figure(figsize=(6, 6))
# Sumamos las capas de la imagen para ver el mensaje encriptado
mensaje_encriptado = imgg_b7 + imgg_b6 + imgg_b5 + imgg_b4 + imgg_b3 +
imgg_b2 + imgg_b1 + imgg_b0
plt.imshow(mensaje_encriptado, cmap="gray")
plt.title("Imagen Encriptada")
plt.show()
```

Se adjunta también capturas del código:

jupyter MENSAJE_OCULTO_LOTSO_PDSEI2023 Last Checkpoint: hace 2 horas (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

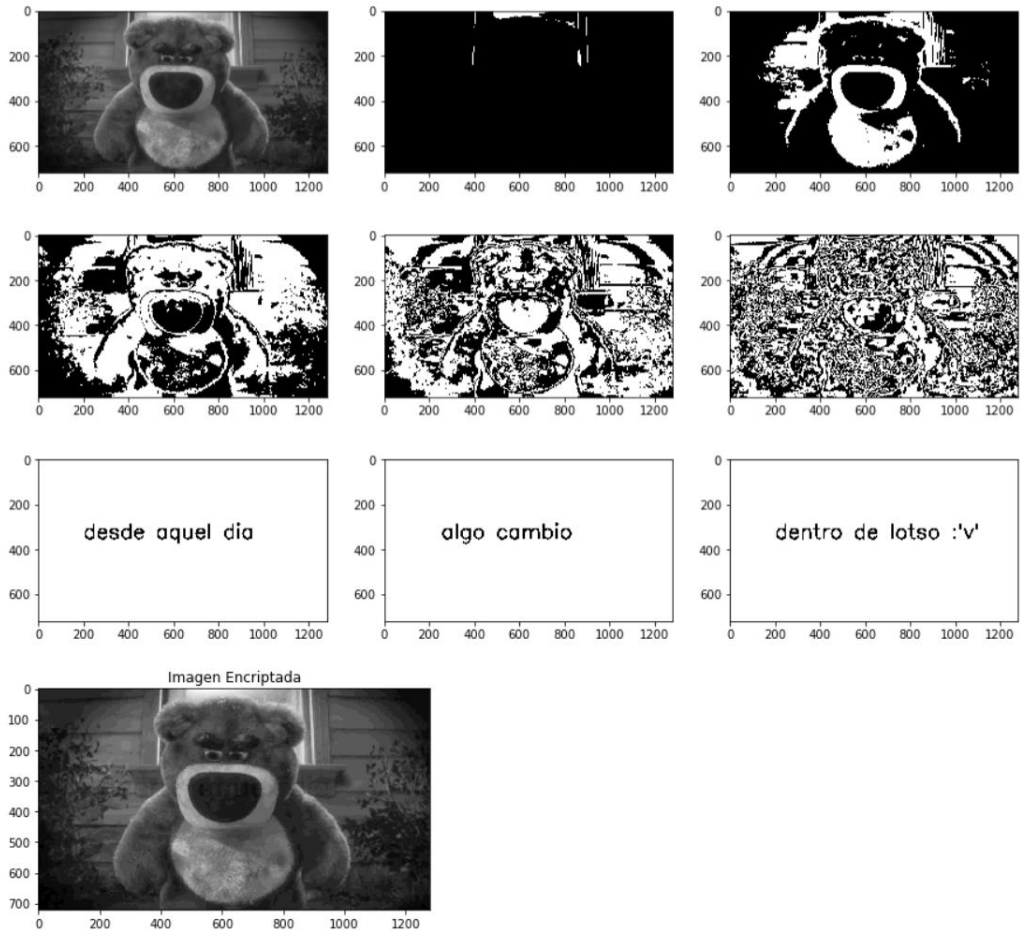
Trusted Python 3

Run Code

In [11]:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Importamos la imagen:
imagen = cv2.imread("/Documentos/LEONEL/UNT/DOCUMENTOS/MECATRONICA/PDSEI2023/imagenes/lotso.jpg", 0)

# Creamos una imagen blanca con las dimensiones de nuestra imagen
imagen_w = np.ones(imagen.shape, dtype=np.uint8) * 255

# Descomponemos la imagen en 8 capas:
imgg_b7 = np.bitwise_and(imagen, 128)
imgg_b6 = np.bitwise_and(imagen, 64)
imgg_b5 = np.bitwise_and(imagen, 32)
imgg_b4 = np.bitwise_and(imagen, 16)
imgg_b3 = np.bitwise_and(imagen, 8)
imgg_b2 = np.bitwise_and(imagen, 4)
imgg_b1 = np.bitwise_and(imagen, 2)
imgg_b0 = np.bitwise_and(imagen, 1)

# Descomponemos la imagen blanca en 8 capas:
imgw_b7 = np.bitwise_and(imagen_w, 128)
imgw_b6 = np.bitwise_and(imagen_w, 64)
imgw_b5 = np.bitwise_and(imagen_w, 32)
imgw_b4 = np.bitwise_and(imagen_w, 16)
imgw_b3 = np.bitwise_and(imagen_w, 8)
imgw_b2 = np.bitwise_and(imagen_w, 4)
imgw_b1 = np.bitwise_and(imagen_w, 2)
imgw_b0 = np.bitwise_and(imagen_w, 1)

# Escribimos el texto en las primeras 3 capas de la imagen blanca:
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(imgw_b2, "desde aquel dia", (200, 350), font, 3, (0, 255, 255), 8)
cv2.putText(imgw_b1, "algo cambio", (250, 350), font, 3, (0, 255, 255), 8)
cv2.putText(imgw_b0, "dentro de lotso :'v'", (200, 350), font, 3, (0, 255, 255), 8)
# Intercambiamos las primeras 3 capas de la imagen con las de la imagen blanca
imgg_b2 = imgw_b2
imgg_b1 = imgw_b1
imgg_b0 = imgw_b0
```

```python
# Imprimimos todas las capas de la imagen
plt.figure(figsize=(15, 10))
plt.subplot(331)
plt.imshow(imagen, cmap="gray")
plt.subplot(332)
plt.imshow(imgg_b7, cmap="gray")
plt.subplot(333)
plt.imshow(imgg_b6, cmap="gray")
plt.subplot(334)
plt.imshow(imgg_b5, cmap="gray")
plt.subplot(335)
plt.imshow(imgg_b4, cmap="gray")
plt.subplot(336)
plt.imshow(imgg_b3, cmap="gray")
plt.subplot(337)
plt.imshow(imgw_b2, cmap="gray")
plt.subplot(338)
plt.imshow(imgg_b1, cmap="gray")
plt.subplot(339)
plt.imshow(imgg_b0, cmap="gray")
plt.show()

plt.figure(figsize=(6, 6))
# Sumamos las capas de la imagen para ver el mensaje encriptado
mensaje_encriptado = imgg_b7 + imgg_b6 + imgg_b5 + imgg_b4 + imgg_b3 + imgg_b2 + imgg_b1 + imgg_b0
plt.imshow(mensaje_encriptado, cmap="gray")
plt.title("Imagen Encriptada")
plt.show()
```

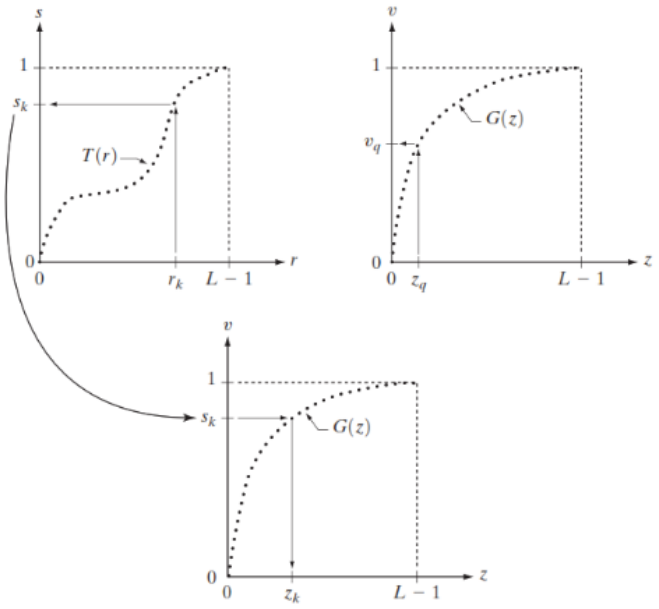


Teniendo en cuenta que esto se estará ejecutando miestras lo estemos utilizando, y que para poder cerrarla utilizaremos “ctrl +c”.

c. Implementar un algoritmo de especificación de histograma de una imagen en escala de grises, cuyo procedimiento se resume en la siguiente figura.

a b
c

FIGURE 3.19
(a) Graphical interpretation of mapping from r_k to s_k via $T(r)$. (b) Mapping of z_q to its corresponding value v_q via $G(z)$. (c) Inverse mapping from s_k to its corresponding value of z_k.



Resolución:

Primero, importamos las librerías de OpenCV, matplotlib, numpy y math

IMPORTACION DE LIBRERIAS

In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import math as m
```

Realizamos la lectura de la imagen 1 “Scarlett_oc.png” usando el comando imread de la librería OpenCV y luego la pasamos a escala de grises añadiendo un 0 al final

LECTURA Y CREACIÓN DE IMAGEN 1 "SCARLET"

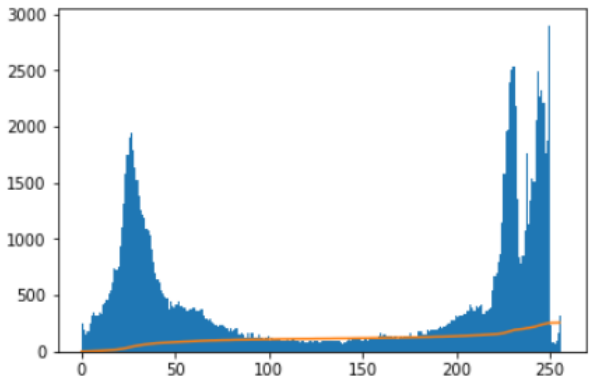
In [9]:
```
img = cv2.imread("../Imagenes/scarlett_oc.png")[..., ::-1]
img_gray = img[..., 0]
```

Creamos el histograma de la imagen 1 en la variable “hist_” con el comando “plt.hist” estableciendo que el eje x sea de 0 a 256. Cabe resaltar que hist_ no está normalizado. Luego, creamos una segunda variable “hist” la cuál es la normalización del histograma. Después, creamos el cdf del histograma con el comando “np.cumsum” y lo almacenamos en la variable “cdf”. “Chancamos” el valor de cdf con la extensión del rango de cdf, pasando de 0 a 1 a de 0 a 256. Por último, redondeamos el valor de cada elemento del vector cdf y graficamos cdf.

CREACIÓN DE HISTOGRAMA Y CDF DE IMAGEN 1

In [10]:
```
hist_ = plt.hist(img_gray.ravel(), 256, [0,256])

hist = hist_[0]/sum(hist_[0])

cdf = np.cumsum(hist)

cdf = cdf*hist_[1].max()

for i in range(len(cdf)):
    cdf[i] = m.ceil(cdf[i])
plt.plot(cdf)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x14efae5e390>]
```

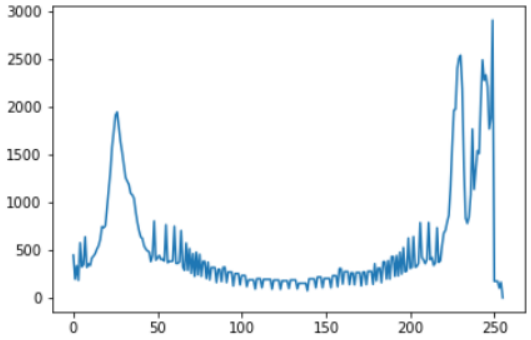


Creamos un vector de la misma dimensión que el del histograma con elementos con valor 0 (matriz de ceros), siendo este el histograma de ecualización de nivel de la imagen 1. Inicializamos con valor 0 dos variables auxiliares, siendo éstas “val” y “contador”. Las siguientes líneas de código de condicionales e iteraciones son para almacenar la ecualización del histograma en la variable “col”. Y por último, graficamos la ecualización.

EQUALIZACIÓN DE NIVEL DE HISTOGRAMA DE IMAGEN 1

In [11]:
```
col = np.zeros_like(hist_[0])

val = 0
contador = 0
for i in range(len(col)):
    if cdf[i] == cdf[i+1]:
        val = hist_[0][i] + hist_[0][i+1]
        contador = contador + 1
    else:
        for j in range(contador):
            col[i-1-j] = val
        val = 0
        contador = 0
        col[i] = hist_[0][i]
    if i == 254:
        break

plt.plot(col)
```

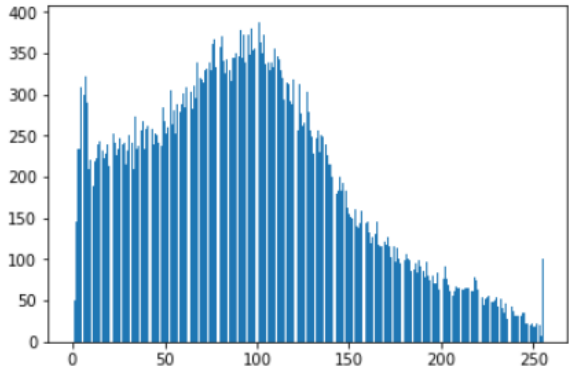
Out[11]:
```
[<matplotlib.lines.Line2D at 0x14efb11ee48>]
```



Los siguientes pasos son los mismos que los anteriores solo que extrapolados para la segunda imagen “paisaje.jpg”

LECTURA Y CREACIÓN DE IMAGEN 2 "PAISAJE" ¶

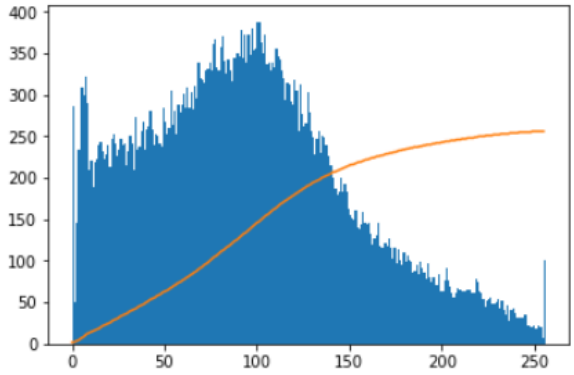
In [12]:
```
img_2 = cv2.imread("../Imagenes/paisaje.jpg")[..., ::-1]
img_2_gray = img_2[..., 0]
```



CREACIÓN DE HISTOGRAMA Y CDF DE IMAGEN 2

In [13]:
```
hist_2_ = plt.hist(img_2_gray.ravel(), 256, [0,256])

hist_2 = hist_2_[0]/sum(hist_2_[0])

cdf_2 = np.cumsum(hist_2)

cdf_2 = cdf_2*hist_2_[1].max()

for i in range(len(cdf_2)):
    cdf_2[i] = m.ceil(cdf_2[i])
plt.plot(cdf_2)
```

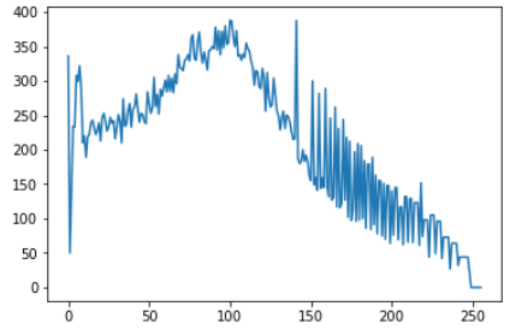
Out[13]:
```
[<matplotlib.lines.Line2D at 0x14efa98f2b0>]
```



EQUALIZACIÓN DE NIVEL DE HISTOGRAMA DE IMAGEN 2

In [14]:
```
col_2 = np.zeros_like(hist_2_[0])

val = 0
contador = 0
for i in range(len(col_2)):
    if cdf_2[i] == cdf_2[i+1]:
        val = hist_2_[0][i] + hist_2_[0][i+1]
        contador = contador + 1
    else:
        for j in range(contador):
            col_2[i-1-j] = val
        val = 0
        contador = 0
        col_2[i] = hist_2_[0][i]
    if i == 254:
        break

plt.plot(col_2)
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x14efc40a128>]
```

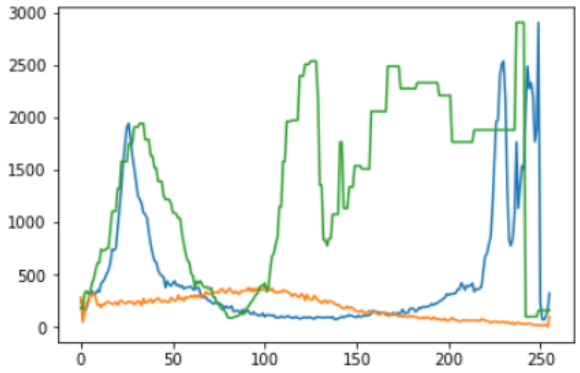


En esta última parte, se realiza el “matching” o especificación de los dos histogramas, básicamente, se comparan los vectores cdf y cdf_2 de tal manera que el histograma de la imagen 2 (imagen a modificar) tome los valores del histograma de la imagen 1 (base) según las posiciones donde los elementos del vector cdf sean igual al vector cdf_2. Para una explicación más profunda ver el video de la bibliografía.

MATCHING O ESPECIFICACIÓN DE HISTOGRAMAS

In [15]:
```
col_match = np.zeros(256)

for i in range(len(cdf)):
    if cdf_2[i] == 0:
        col_match[i] = 0
    for j in range(len(cdf_2)):
        if cdf_2[i] - cdf[j] >=0:
            col_match[i] = col[j]

plt.plot(hist_[0]) #IMAGEN BASE AZUL
plt.plot(hist_2_[0]) #IMAGEN A MODIFICAR NARANJA
plt.plot(col_match) #HISTOGRAMA DE IMAGEN MODIFICADA VERDE
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x14efc45f208>]
```



2. TEST DE COMPROBACIÓN

a. Averiguar como crear un histograma en 2d con OpenCV.

Resolución:

Un histograma en 2D o "histograma bidimensional", es una representación gráfica de la distribución conjunta de dos variables en un conjunto de datos. En una imagen en escala de grises, un histograma en 2D muestra la cantidad de píxeles que tienen cierta intensidad en el eje X y cierta intensidad en el eje Y.
Con la librería OpenCV de Python, se puede crear un histograma en 2D de la siguiente manera:

Primero, se tiene como imagen a modificar a la siguiente imagen:

Importamos las librerías necesarias:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

Cargamos la imagen en escala de grises:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread('chessp.jpg', 0)
```

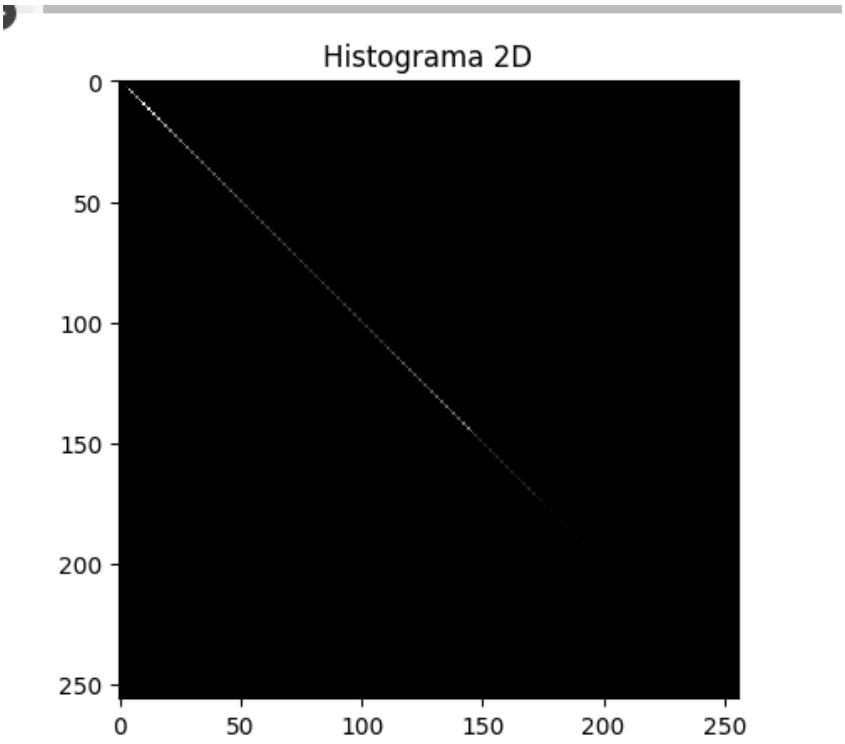
Calculamos un histograma en 2D manualmente:

```
image = cv2.imread('chessp.jpg', 0)

hist_2D, x_edges, y_edges = np.histogram2d(image.ravel(), image.ravel(), bins=(256, 256), range=([0, 256], [0, 256]))
```

Mostramos el histograma en 2D:

```
hist_2D, x_edges, y_edges = np.histogram2d(image.ravel(),

plt.imshow(hist_2D, cmap='gray',interpolation='nearest')
plt.title('Histograma 2D')
plt.show()
```

El resultado es el siguiente:



El código completo es:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread('chessp.jpg', 0)

hist_2D, x_edges, y_edges = np.histogram2d(image.ravel(), image.ravel(), bins=(256, 256), range=([0, 256], [0, 256]))

plt.imshow(hist_2D, cmap='gray',interpolation='nearest')
plt.title('Histograma 2D')
plt.show()
```

b. Definir una función que ingresando un rango de entrada (ej. 100, 120), se cree una función para expandir ese rango, en todo el rango dinámico de 0 a 255.

Resolución:

Para lograr eso, debemos tener en cuentas la importación de las librerías necesarias. Nuestra función tomará la imagen y tendrá en consideración los valores mínimo y máximo del rango que deseamos expandir.
A continuación, un ejemplo de una función implementada que cumple con lo solicitado, teniendo en cuenta que en donde dice “imagen.jpg” iría nuestra imagen a expandir:

```
import cv2
import numpy as np

def expandir_rango(input_image, min_range, max_range):
    #verificamos que los valores estén en el rango válido (0-255)
    min_range = max(0, min_range)
    max_range = min(255, max_range)
```

```
    # Calculamos la amplitud del rango de entrada
    input_range = max_range - min_range

    # Verificamos que la amplitud sea mayor que cero para evitar divisiones por cero
    if input_range > 0:
        # Aplicamos la expansión del rango
        output_image = ((input_image - min_range) / input_range) * 255
        # verificamos que los valores estén en el rango válido (0-255)
        output_image = np.clip(output_image, 0, 255).astype(np.uint8)
        return output_image
    else:
        # Si el rango de entrada es igual o menor a cero, devuelve la imagen original
        return input_image

# Cargamos una imagen de entrada
input_image = cv2.imread('imagen.jpg', cv2.IMREAD_GRAYSCALE)

# Definimos el rango de entrada que deseamos expandir (ejemplo: 100 a 120)
min_range = 100
max_range = 120

# Aplicamos la expansión del rango
output_image = expandir_rango(input_image, min_range, max_range)

# Guardamos la imagen resultante
cv2.imwrite('imagen_expandida.jpg', output_image)
```

c. Definir una función para aumentar o reducir el brillo de una imagen, según valor especificado.

Resolución:

Importamos las librerías OpenCV, matplotlib, numpy y math.

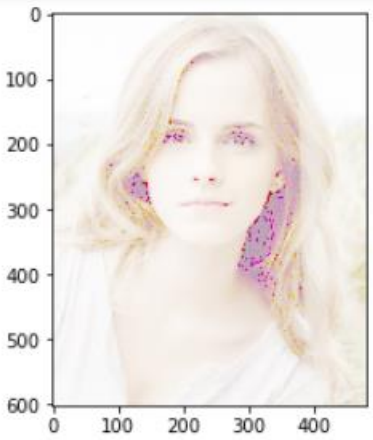
```
In [2]: import cv2
        import matplotlib.pyplot as plt
        import numpy as np
        import math as m
```

Realizamos la lectura de la imagen “emma.jpg” ya la almacenamos en la variable “imgSRC”.

```
In [3]: imgSRC = '../Imagenes/emma.jpg'
```

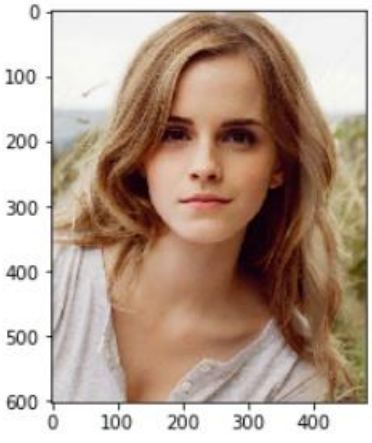
Definimos una variable “PL”, la cuál es nuestra variable de entrada para definir un porcentaje de brillo (de 0 a 100%). Luego, definimos una variable “e”, la cuál realiza un mapeo del rango de 0 a 100 a un rango de 1 a 0.1 respectivamente. Después, creamos la variable “r” en la cuál hacemos la lectura de la imagen “imgSCR”. Además, creamos la variable “s”, la cuál es la definición de la Transformación Gaussiana, tomando como entradas las variables r y e. Por último, graficamos la imagen.

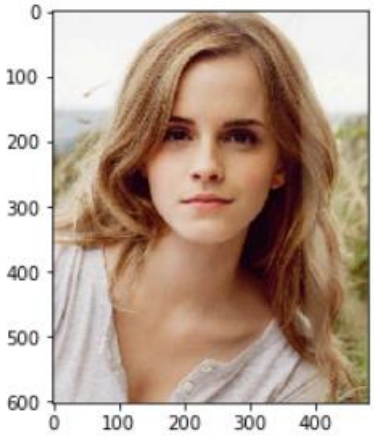
In [4]:
```
PL = 100 #Porcentaje de luz (En este caso 50%)
#En caso de querer bajar el brillo, se debe colocar un porcentaje negativo ejm:(-100)
e = PL*(-0.9/100)+1
r = cv2.imread(imgSRC)[..., ::-1]/255
s = r**e
plt.imshow(s)
plt.show()
```

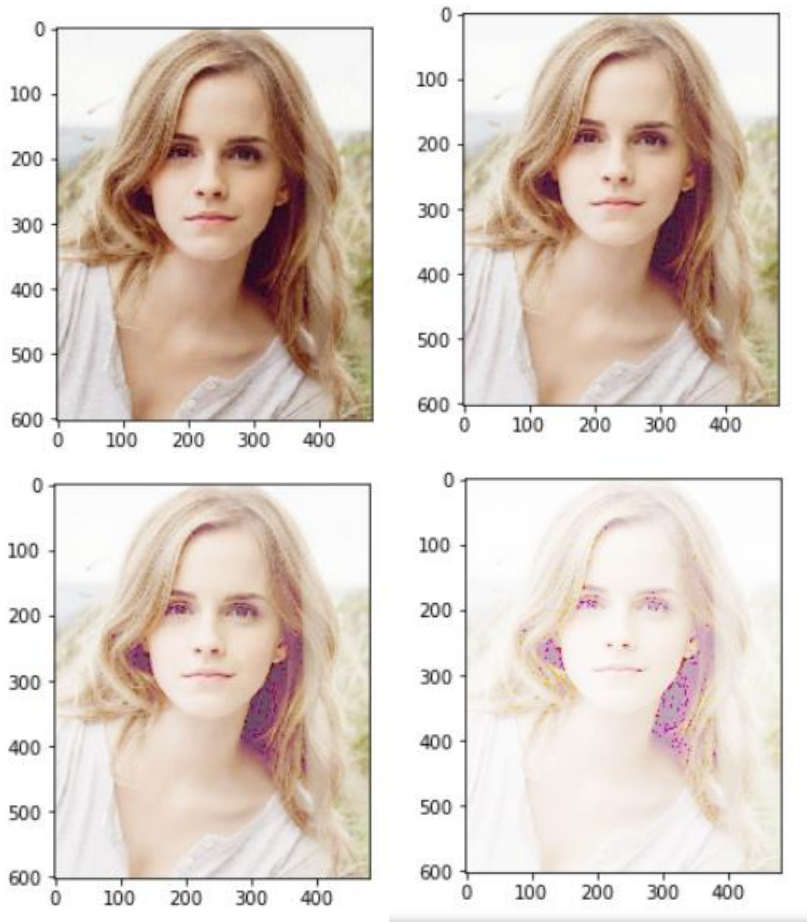


Las siguientes líneas de código son pruebas de la función gaussiana con porcentajes de brillo 0, 10, 40, 60, 80 y 100.

```
#DIFERENTES PORCENTAJES DE LUZ
PL_0 = 0
PL_1 = 10
PL_2 = 40
PL_3 = 60
PL_4 = 80
PL_5 = 100
e_0 = PL_0*(-0.9/100)+1
e_1 = PL_1*(-0.9/100)+1
e_2 = PL_2*(-0.9/100)+1
e_3 = PL_3*(-0.9/100)+1
e_4 = PL_4*(-0.9/100)+1
e_5 = PL_5*(-0.9/100)+1

s_0 = r**e_0
s_1 = r**e_1
s_2 = r**e_2
s_3 = r**e_3
s_4 = r**e_4
s_5 = r**e_5

plt.imshow(s_0)
plt.show()

plt.imshow(s_1)
plt.show()

plt.imshow(s_2)
plt.show()

plt.imshow(s_3)
plt.show()

plt.imshow(s_4)
plt.show()

plt.imshow(s_5)
plt.show()
```







3. BIBLIOGRAFÍA

Friendly, C. (2 de Marzo de 2021). "*Histogram matching in digital image processing*". Obtenido de Youtube: https://www.youtube.com/watch?v=r565euxWZBs&ab_channel=CollegeFriendly