

Procesamiento de imagen usando geometría vectorial

Prof. Francisco Alejandro Medina

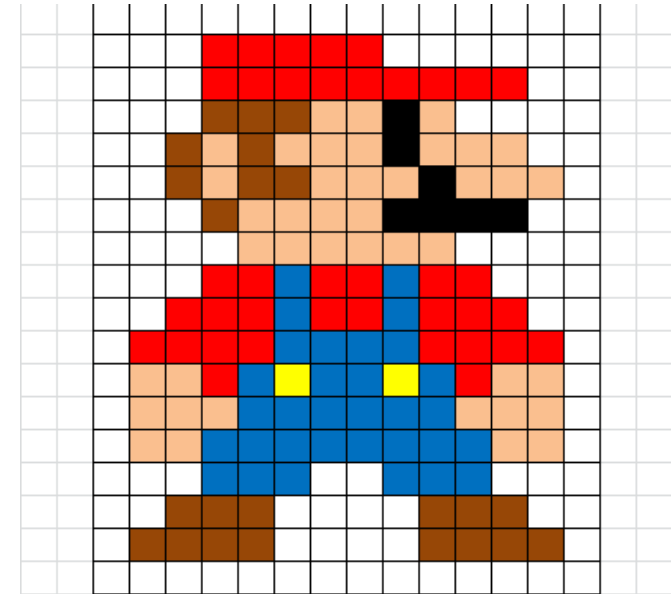
La Imagen

Una imagen es una representación esculpida, pintada o fotografiada de un objeto o escena. Actualmente estas pueden ser representadas sobre un monitor electrónico como una pantalla Led.



Imagen Digital

Una imagen digital está compuesta por píxeles, que son las unidades mínimas de información. Cada píxel almacena un valor numérico que representa su color y brillo. La cantidad de píxeles determina la resolución y la cantidad de información de la imagen.



Leer una imagen



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 =plt.imread('Imagen.jpg')/255
5
6  plt.figure("Imagen Original")
7  plt.imshow(img1)
8  plt.axis('off')
9  plt.show()
10
```



Universidad
Tecnológica
de Pereira

Imagen Digital

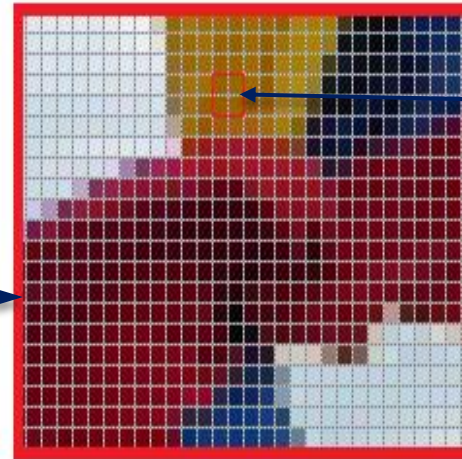
Una imagen digital es una celda compuesta por unos elementos llamados píxeles, que son los componentes más pequeños de una imagen digital.



Imagen Digital

Cada cuadro de la imagen se denomina pixel (*picture element*)

Una imagen a color tiene 3 valores de color por cada pixel.



R:146 G: 93 B: 0	R:148 G: 95 B: 0
R:160 G: 91 B: 14	R:161 G: 92 B: 14

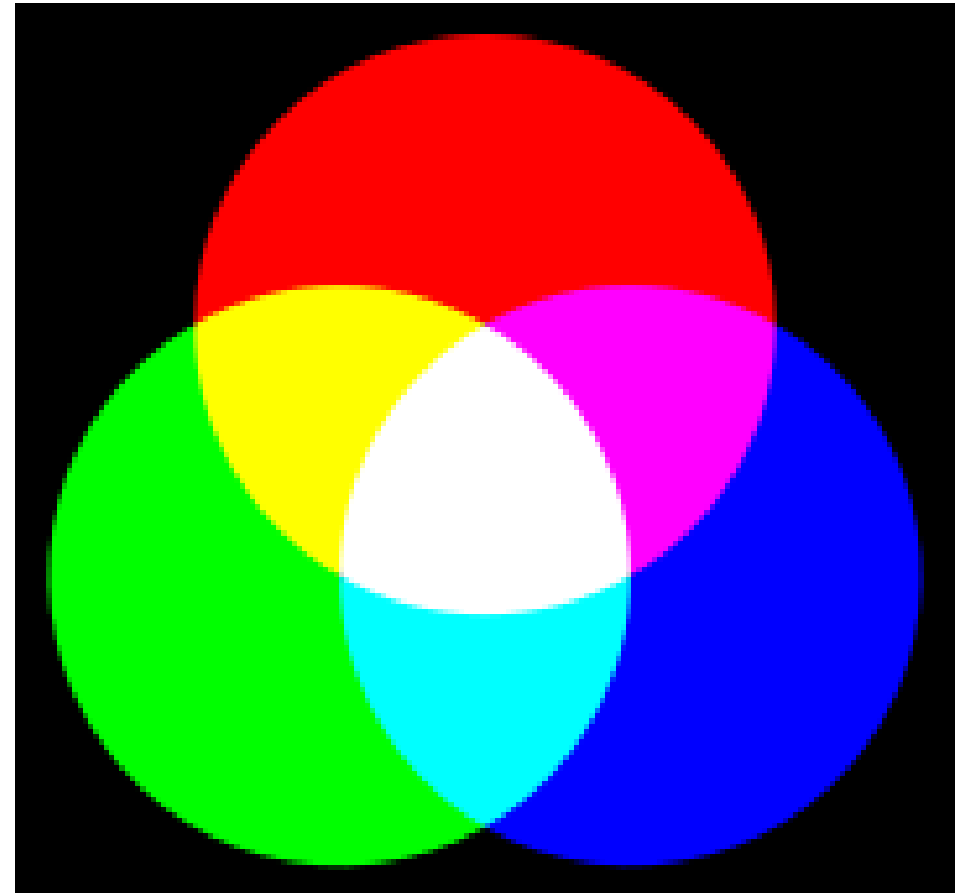
Una imagen en blanco y negro tiene solo un valor de color por cada pixel.

Componentes de color

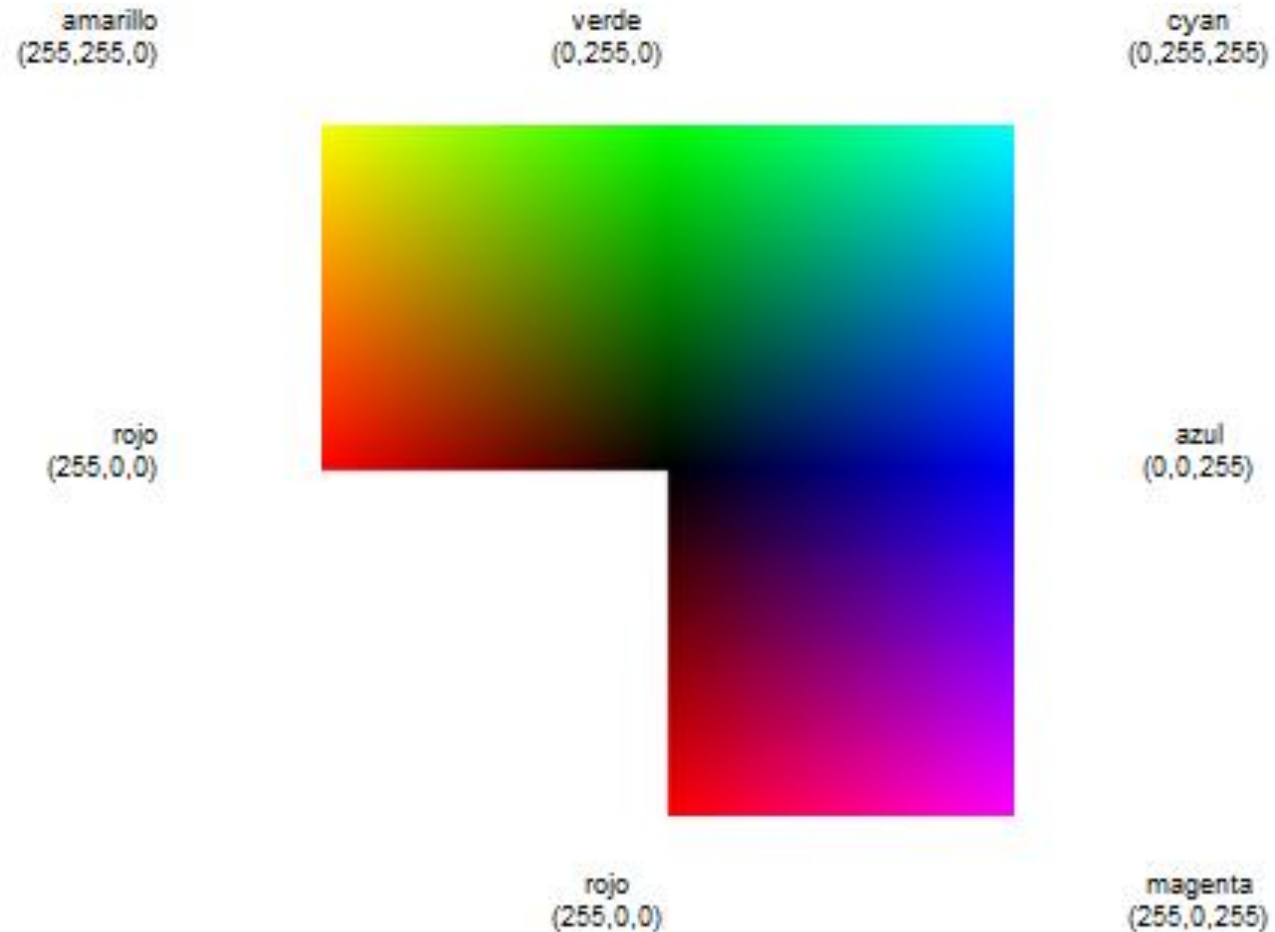
- El RGB (Red, Green, Blue) representa a los colores en función de la intensidad de los colores primarios.
- Es posible representar un color mediante la adición de los tres colores luz primarios.

0 → No interviene en la mezcla

255 → Aporta más intensidad.

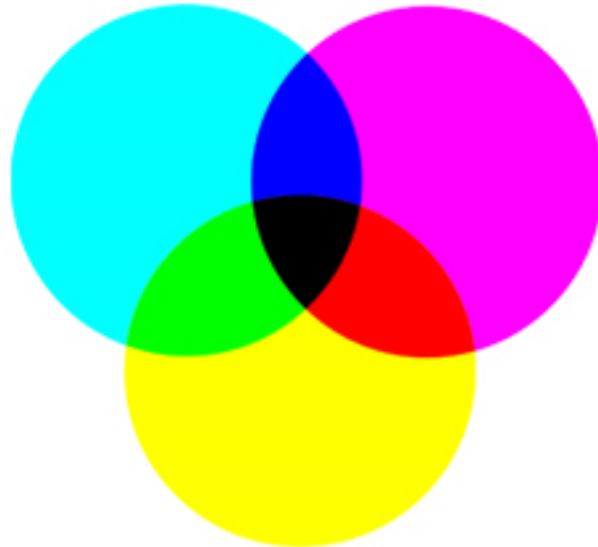


Componentes de color



Comparación RGB y CMYK

CMYK



RGB



RGB

- Cualquier color puede ser representado mediante la combinación de los colores rojo, verde y azul, cada uno en diferente proporción.
- La combinación RGB estándar indica 256 niveles por cada canal, es decir por cada color rojo, verde o azul. ($256 \times 256 \times 256 = 16,777,216$).

666633 R: 102 G: 102 B: 051	999966 R: 153 G: 153 B: 102	CCCC99 R: 204 G: 204 B: 153	FFFFCC R: 255 G: 255 B: 204	FFFF99 R: 255 G: 255 B: 153	FFFF66 R: 255 G: 255 B: 102	FFFF33 R: 255 G: 255 B: 051	FFFF00 R: 255 G: 255 B: 000
993300 R: 153 G: 051 B: 000	CC6633 R: 204 G: 102 B: 051	663300 R: 102 G: 051 B: 000	FF9966 R: 255 G: 153 B: 102	FF6633 R: 255 G: 102 B: 051	FF9933 R: 255 G: 153 B: 051	FF6600 R: 255 G: 102 B: 000	CC3300 R: 204 G: 051 B: 000
3333CC R: 051 G: 051 B: 204	0066FF R: 000 G: 102 B: 255	0033FF R: 000 G: 051 B: 255	3366FF R: 051 G: 102 B: 255	3366CC R: 051 G: 102 B: 204	000066 R: 000 G: 000 B: 102	000033 R: 000 G: 000 B: 051	0000FF R: 000 G: 000 B: 255
33FF33 R: 051 G: 255 B: 051	00CC33 R: 000 G: 204 B: 051	33CC33 R: 051 G: 204 B: 051	66FF33 R: 102 G: 255 B: 051	00FF00 R: 000 G: 255 B: 000	66CC33 R: 102 G: 204 B: 051	006600 R: 000 G: 102 B: 000	003300 R: 000 G: 051 B: 000
FF3333 R: 255 G: 051 B: 051	CC3333 R: 204 G: 051 B: 051	FF6666 R: 255 G: 102 B: 102	660000 R: 102 G: 000 B: 000	990000 R: 153 G: 000 B: 000	CC0000 R: 204 G: 000 B: 000	FF0000 R: 255 G: 000 B: 000	FF3300 R: 255 G: 051 B: 000
CC9966 R: 204 G: 153 B: 102	FFCC99 R: 255 G: 204 B: 153	FFFFFF R: 255 G: 255 B: 255	CCCCCC R: 204 G: 204 B: 204	999999 R: 153 G: 153 B: 153	666666 R: 102 G: 102 B: 102	333333 R: 051 G: 051 B: 051	000000 R: 000 G: 000 B: 000

Descomposición RGB

Imagen Original



Capa Roja



Capa Verde



Capa Azul



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 imgUTP=np.array(plt.imread("Imagen.jpg"))/255
5
6 plt.subplot(2,2,1)
7 plt.imshow(imgUTP)
8 plt.axis('off')
9 plt.title("Imagen Original")
10
11 CapaR=np.copy(imgUTP)
12 CapaR[:, :, 1]= CapaR[:, :, 2]=0
13 plt.subplot(2,2,2)
14 plt.imshow(CapaR)
15 plt.axis('off')
16 plt.title("Capa Roja")
17
18 CapaG=np.copy(imgUTP)
19 CapaG[:, :, 0]= CapaG[:, :, 2]=0
20 plt.subplot(2,2,3)
21 plt.imshow(CapaG)
22 plt.axis('off')
23 plt.title("Capa Verde")
24
25 CapaB=np.copy(imgUTP)
26 CapaB[:, :, 0]= CapaB[:, :, 1]=0
27 plt.subplot(2,2,4)
28 plt.imshow(CapaB)
29 plt.axis('off')
30 plt.title("Capa Azul")
31
32 plt.show()
```

Canales CMYK

Imagen Original



Canal Cian



Canal Magenta



Canal Amarillo



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 imgUTP=np.array(plt.imread("Imagen.jpg"))/255
5
6 plt.subplot(2,2,1)
7 plt.imshow(imgUTP)
8 plt.axis('off')
9 plt.title("Imagen Original")
10
11 CapaR=np.copy(imgUTP)
12 CapaR[:, :, 1]= CapaR[:, :, 2]=1
13 plt.subplot(2,2,2)
14 plt.imshow(CapaR)
15 plt.axis('off')
16 plt.title("Canal Cian")
17
18 CapaG=np.copy(imgUTP)
19 CapaG[:, :, 0]= CapaG[:, :, 2]=1
20 plt.subplot(2,2,3)
21 plt.imshow(CapaG)
22 plt.axis('off')
23 plt.title("Canal Magenta")
24
25 CapaB=np.copy(imgUTP)
26 CapaB[:, :, 0]= CapaB[:, :, 1]=1
27 plt.subplot(2,2,4)
28 plt.imshow(CapaB)
29 plt.axis('off')
30 plt.title("Canal Amarillo")
31
32 plt.show()
```

Inversión Color

La inversión de una imagen consiste en convertir cada canal RGB a su negativo, como en una película fotográfica: blanco se vuelve negro, azul a amarillo, verde a magenta y rojo a cian. Este filtro se usa principalmente para digitalizar negativos fotográficos.

$$I_{\text{invertida}} = 255 - I_{\text{original}}$$



Universidad
Tecnológica
de Pereira

I_{original}



$I_{\text{invertida}}$



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 = plt.imread('Imagen.jpg')/255
5
6  ImgNegativo = 1 - img1
7  plt.figure("Negativo de una imagen")
8  plt.imshow(ImgNegativo)
9  plt.axis('off')
10 plt.show()
11
```

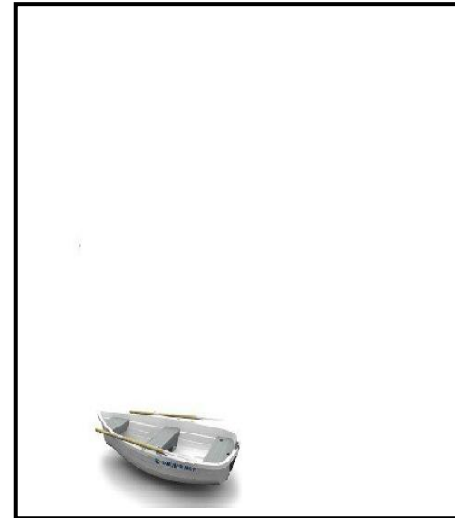
Suma y Resta de Matrices

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

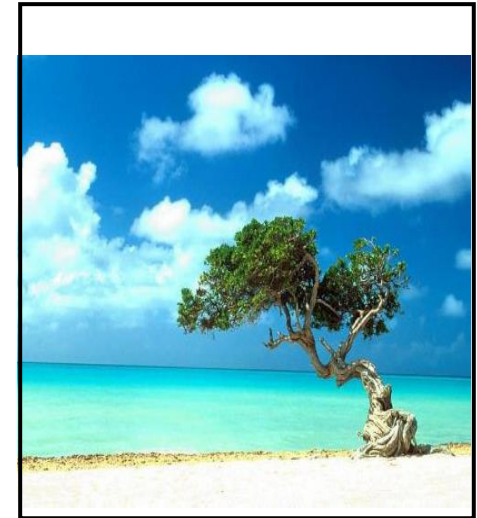
$$A + B = \begin{pmatrix} 2+1 & 0+0 & 1+1 \\ 3+1 & 0+2 & 0+1 \\ 5+1 & 1+1 & 1+0 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix}$$

$$A - B = \begin{pmatrix} 2-1 & 0-0 & 1-1 \\ 3-1 & 0-2 & 0-1 \\ 5-1 & 1-1 & 1-0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -2 & -1 \\ 4 & 0 & 1 \end{pmatrix}$$

Bote



Mar



$$R = \text{Mar} + \text{Bote}$$

Suma y Resta de Matrices



Imágenes Fusionadas



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 = plt.imread('Bote.jpg')/255
5  img2 = plt.imread('Mar.jpg')/255
6
7  SumaImg = img1 + img2
8  plt.figure("Suma de imagenes")
9  plt.imshow(SumaImg)
10 plt.axis('off')
11 plt.show()
12
```


Suma y Resta de Matrices



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 = plt.imread('Bote.jpg')/255
5  img2 = plt.imread('Mar.jpg')/255
6
7  Factor=0.2
8  SumaImg = img1*Factor + img2*(1-Factor)
9  plt.figure("Suma de imagenes")
10 plt.imshow(SumaImg)
11 plt.axis('off')
12 plt.show()
13
```

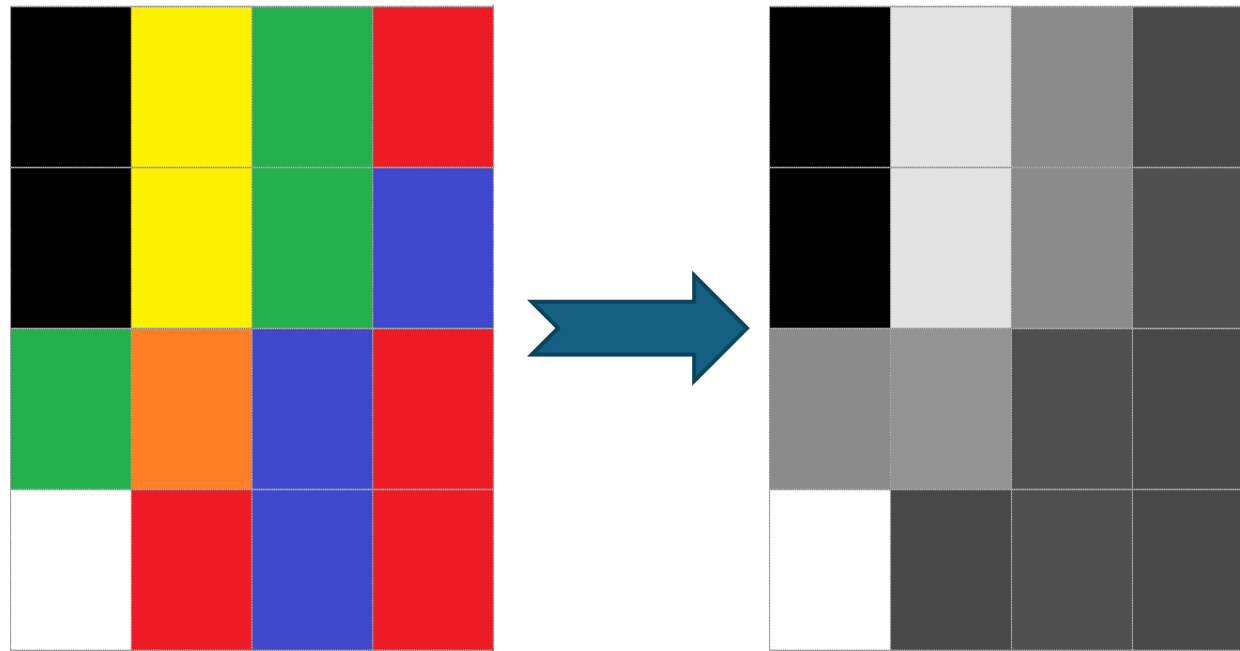


$R = \text{Mar} * \text{factor} + (1 - \text{factor}) * \text{Bote};$

Imágenes Fusionadas y Ecualizadas

Escala de Grises

Una imagen en escala de grises es un arreglo matricial de dos dimensiones que aporta información de la intensidad de la luz presente para cada punto de la imagen.



Técnica de promedio (Average)

- La forma más simple de lograrlo es mediante la suma de las componentes RGB de cada capa pixel a pixel y dividirles por la cantidad (3).

$$I_{gris} = \frac{R+G+B}{3}$$



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 = plt.imread('Imagen.jpg')/255
5
6 ImgGris = (img1[:, :, 0]+img1[:, :, 1]+img1[:, :, 2])/3
7 plt.figure("Escala de grises (Promedio)")
8 plt.imshow(ImgGris, cmap='gray')
9 plt.axis('off')
10 plt.show()
11
```

Técnica de Luminosidad (Luminosity)

Este método mejora el promedio al ponderar los canales RGB según la sensibilidad del ojo humano, siendo más sensible al verde. Los coeficientes provienen del estándar Rec. 601 NTSC de la UIT-R, ampliamente adoptado en televisión a color compatible con blanco y negro.

$$I_{gris} = 0.299R + 0.587G + 0.114B$$



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 = plt.imread('Imagen.jpg')/255
5
6 ImgGris = 0.299*img1[:, :, 0] + 0.587*img1[:, :, 1] + 0.114*img1[:, :, 2]
7 plt.figure("Escala de grises (Luma (Percepción humana de brillo))")
8 plt.imshow(ImgGris, cmap='gray')
9 plt.axis('off')
10 plt.show()
11
```

Técnica de La tonalidad (Midgray)

En el modelo HSL (del inglés *Hue*, *Saturation*, *Lightness* con su equivalencia en español Tonalidad, Saturación, Luminancia o intensidad) la tonalidad se define como parte de la media de las componentes de color máxima y mínima.

$$I_{gris} = \frac{\max(R,G,B) + \min(R,G,B)}{2}$$



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 = plt.imread('Imagen.jpg')/255
5
6 ImgGris = (np.maximum(img1[:, :, 0], img1[:, :, 1], img1[:, :, 2]) + np.minimum(img1[:, :, 0], img1[:, :, 1], img1[:, :, 2]))/2
7 plt.figure("Escala de grises (Midgray)")
8 plt.imshow(ImgGris, cmap='gray')
9 plt.axis('off')
10 plt.show()
```

Ajuste de Brillo

El ajuste de brillo modifica la luminancia de una imagen. Va del 0 % (negro) al 100 % (blanco) y es uno de los controles más básicos y utilizados en el procesamiento de imágenes.

$$I_{mod} = I + b, \text{ donde } b \text{ es el ajuste de brillo.}$$



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 = plt.imread('Imagen.jpg')/255
5
6 plt.figure("Ajuste de brillo")
7
8 plt.subplot(1,2,1)
9 plt.axis('off')
10 plt.title('Imagen Original')
11 plt.imshow(img1)
12
13
14 plt.subplot(1,2,2)
15 brillo = 0.5
16 imgBrillo = img1 + brillo
17 plt.axis('off')
18 plt.title('Imagen con Brillo')
19 plt.imshow(imgBrillo)
20
21
22 plt.show()
23
```

Ajuste de Canal

Consiste en manipular la intensidad de únicamente un canal a la vez, ya sea el rojo, verde, o azul independientemente.

Imagen Original



Ajuste del Canal Rojo



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 =plt.imread('Imagen.jpg')/255
5
6 plt.figure("Ajuste de brillo")
7
8 plt.subplot(1,2,1)
9 plt.axis('off')
10 plt.title('Imagen Original')
11 plt.imshow(img1)
12
13
14 plt.subplot(1,2,2)
15 brillo = 0.5
16 imgCanal=np.copy(img1)
17 Canal=0 #Canal Rojo
18 imgCanal[:, :, Canal]=img1[:, :, Canal]+brillo
19 plt.axis('off')
20 plt.title('Ajuste del Canal Rojo')
21 plt.imshow(imgCanal)
22
23
24 plt.show()
```

Ajuste de Contraste

El contraste, se refiere a la diferencia que existe entre las zonas oscuras y claras de la imagen.

La función logaritmo se utiliza para aumentar el contraste de las zonas oscuras en detrimento de las zonas claras.

$$IMB(i,j)=K*\text{Log}_{10}(1+IMA(i,j))$$

La función exponencial se utiliza para aumentar el contraste de las zonas claras en detrimento de las oscuras.

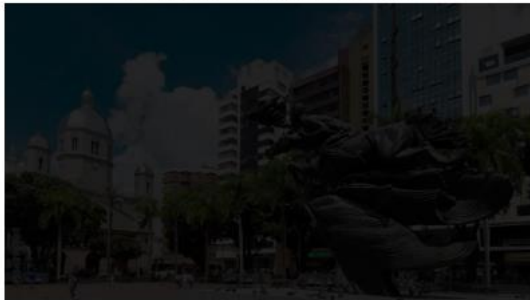
$$IMB(i,j)=K*\exp(IMA(i,j)-1)$$

Ejemplo de Contraste

Imagen Original



Contraste Zonas Oscuras



Contraste Zonas Claras



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 img1 =plt.imread('Imagen.jpg')/255
5
6 plt.figure("Ajuste del Contraste")
7
8 plt.subplot(3,1,1)
9 plt.axis('off')
10 plt.title('Imagen Original')
11 plt.imshow(img1)
12
13 Contraste = 0.5
14
15 plt.subplot(3,1,2)
16 ImgContraste= Contraste*np.log10(1+img1)
17 plt.axis('off')
18 plt.title('Contraste Zonas Oscuras')
19 plt.imshow(ImgContraste)
20
21 plt.subplot(3,1,3)
22 ImgContraste= Contraste*np.exp(img1-1)
23 plt.axis('off')
24 plt.title('Contraste Zonas Claras')
25 plt.imshow(ImgContraste)
26
27
28
29 plt.show()
30
```


Binarizar un imagen

La binarización convierte una imagen en blanco y negro. Los píxeles con valores menores al umbral se asignan a 0, y los mayores o iguales a 1, generando una imagen formada solo por ceros y unos.



Universidad
Tecnológica
de Pereira

$$IMB(i,j) = (IMA(i,j) \geq \text{umbral})$$



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 = plt.imread('Imagen.jpg')/255
5
6
7  Gris=(img1[:, :, 0]+img1[:, :, 1]+img1[:, :, 2])/3
8  Umbral=0.5
9  ImgBin=(Gris > Umbral)
10
11 plt.figure("Imagen Binarizada")
12 plt.imshow(ImgBin, cmap='gray')
13 plt.axis('off')
14 plt.show()
```

Trasladar una imagen

La traslación es una operación de geometría afín que **suma una cantidad fija a las coordenadas de cada píxel.**

Si (x,y) es la posición original de un píxel, la traslación se define como:

$$(x' , y')=(x + \Delta x , y + \Delta y)$$

Donde:

- Delta x: desplazamiento horizontal (positivo a la derecha, negativo a la izquierda)
- Delta y: desplazamiento vertical (positivo hacia abajo, negativo hacia arriba)

Ejemplo de Traslación

Imagen Original



Imagen Traslada



```
1  # Crear una nueva imagen del mismo tamaño llena de ceros (fondo negro)
2  trasladada = np.zeros_like(img)
3
4  # Calcular límites válidos para el copiado
5  h, w = img.shape[:2]
6  x_origen_inicio = 0
7  x_origen_fin = w - dx
8  y_origen_inicio = 0
9  y_origen_fin = h - dy
10
11 # Asignar los valores trasladados
12 trasladada[dy:h, dx:w] = img[y_origen_inicio:y_origen_fin, x_origen_inicio:x_origen_fin]
13
14 # Visualizar resultados
15 plt.figure("Traslación sin np.roll")
16 plt.subplot(1, 2, 1)
17 plt.title("Imagen Original")
18 plt.imshow(img)
19 plt.axis('off')
20
21 plt.subplot(1, 2, 2)
22 plt.title("Imagen Traslada")
23 plt.imshow(trasladada)
24 plt.axis('off')
25 plt.show()
26
```

Recorte de una imagen

Se extrae una porción específica de la imagen.

Imagen Original



Imagen Recortada



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 =plt.imread('Imagen.jpg')/255
5  print("Tamaño imagen original : ",img1.shape)
6  xIni = 50
7  xFin = 200
8  yIni = 50
9  yFin = 200
10  ImgRecorte = img1[xIni:xFin, yIni:yFin]
11  print("Tamaño imagen escalada : ",ImgRecorte.shape)
12
13  plt.figure("Escalado de una imagen")
14  plt.subplot(2, 1, 1)
15  plt.title("Imagen Original")
16  plt.imshow(img1)
17  plt.axis('off')
18
19  plt.subplot(2, 1, 2)
20  plt.title("Imagen Recortada")
21  plt.imshow(ImgRecorte)
22  plt.axis('off')
23
24  plt.show()
25
```

Rotar una imagen

Los algoritmos de giro son generalmente los más complejos y por lo tanto los más costosos en tiempo de procesamiento. Debido a esto, sólo se utilizan cuando es posible obtener una posición de giro que simplifique más posteriores procesos.

Dado un punto $IMB(i,j)$ y se rota θ grados, las coordenadas i' y j' del nuevo punto serán:

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix}$$

Por lo tanto, la versión girada de la imagen principal será:

$$IMB(i',j') = IMB(i \cos \theta - j \sin \theta, i \sin \theta + j \cos \theta) = IMA(i,j)$$

Imagen rotada



Ejemplo de rotación una imagen

```
def rotarImg(a, ang):  
    """  
    Rota una imagen en sentido antihorario por un ángulo dado.  
    Parámetros:  
    a (numpy.ndarray): Imagen de entrada representada como un arreglo 2D.  
    ang (float): Ángulo de rotación en grados. Debe estar en el rango (0,  $\pi$ ] radianes.  
    Devuelve:  
    numpy.ndarray: Imagen rotada con el mismo tipo de datos que la imagen de entrada.  
    Excepciones:  
    ValueError: Si el ángulo está fuera del rango esperado (0 < ang <=  $\pi$ ).  
    Notas:  
    - La función convierte el ángulo de grados a radianes internamente.  
    - La imagen de salida tendrá dimensiones ajustadas para contener la imagen rotada.  
    - La rotación se realiza en sentido antihorario.  
    """  
    ang = np.radians(ang) # Convertir a radianes  
    m, n = a.shape  
    cos_ang = np.cos(ang)  
    sin_ang = np.sin(ang)  
  
    if ang > 0 and ang <= np.pi / 2:  
        c = int(round(m * sin_ang + n * cos_ang)) + 1  
        d = int(round(m * cos_ang + n * sin_ang)) + 1  
        b = np.zeros((c, d), dtype=a.dtype)  
  
        for i in range(c):  
            for j in range(d):  
                iii = i - int(n * sin_ang) - 1  
                ii = int(round(j * sin_ang + iii * cos_ang))  
                jj = int(round(j * cos_ang - iii * sin_ang))  
                if 0 <= ii < m and 0 <= jj < n:  
                    b[i, j] = a[ii, jj]
```

```
    elif ang > np.pi / 2 and ang <= np.pi:  
        c = int(round(m * sin_ang - n * cos_ang)) + 1  
        d = int(round(m * sin_ang - n * cos_ang)) + 1  
        e = -n * cos_ang  
        b = np.zeros((c, d), dtype=a.dtype)  
  
        for i in range(c):  
            iii = c - i - 1  
            for j in range(d):  
                jjj = d - j - 1  
                ii = int(round(jjj * sin_ang + iii * cos_ang))  
                jj = int(round(jjj * cos_ang - iii * sin_ang))  
                if 0 <= ii < m and 0 <= jj < n:  
                    b[i, j] = a[ii, jj]  
  
    else:  
        raise ValueError("Ángulo fuera del rango esperado (0 < ang <=  $\pi$ )")  
  
    return b
```

```
# Ejemplo de uso  
img = plt.imread('imagen.jpg')  
if img.ndim == 3:  
    img = np.dot(img[..., :3], [0.299, 0.587, 0.114]) # convertir a escala de grises  
  
rotada = rotarImg(img, 45)  
  
plt.imshow(rotada, cmap='gray')  
plt.axis('off')  
plt.title("Imagen rotada")  
plt.show()
```

Reducción de la resolución de una imagen

- La reducción de la resolución es el proceso de **disminuir el número de píxeles** en una imagen, lo que implica **perder detalle** visual.

Supongamos una imagen original representada como una matriz $I \in \mathbb{R}^{M \times N}$

Submuestreo directo (sin interpolación):

La reducción de resolución por un factor $f \in \mathbb{N}$ se puede expresar como:

$$I_{\text{reducida}}(i, j) = I(i \cdot f, j \cdot f)$$

donde:

- f es el **factor de reducción** (por ejemplo, $f = 2$ reduce la resolución a la cuarta parte).
- Solo se conservan los píxeles ubicados cada f posiciones.

👉 Esto equivale a tomar **una muestra cada f píxeles**  en ambas direcciones (horizontal y vertical).

Ejemplo de reducción

Ejemplo visual:

- Si tienes una imagen de 6×6 píxeles:

[A	B	C	D	E	F]
[G	H	I	J	K	L]
[M	N	O	P	Q	R]
[S	T	U	V	W	X]
[Y	Z	a	b	c	d]
[e	f	g	h	i	j]

- Si hace una reducción por factor 2, obtienes:

[A	C	E]
[M	O	Q]
[Y	a	c]

Reducción de la resolución

Imagen Original



Imagen Reducción de resolución



```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  img1 =plt.imread('Imagen.jpg')/255
5
6  '''Toma 1 de cada zoom_factor píxeles tanto en filas como en columnas.
7  Es decir, si zoom_factor = 5, tomará las filas 0, 5, 10, 15...
8  y las columnas 0, 5, 10, 15... Esto se llama submuestreo o downsampling.
9  '''
10 zoom_factor = 5
11 zoomed = img1[::zoom_factor, ::zoom_factor]
12
13 print("Tamaño imagen original : ",img1.shape)
14 print("Tamaño con la reducción de resolución : ",zoomed.shape)
15
16 plt.figure("Reducción de resolución")
17
18 plt.subplot(2, 1, 1)
19 plt.title("Imagen Original")
20 plt.imshow(img1)
21 plt.axis('off')
22
23 plt.subplot(2, 1, 2)
24 plt.title("Imagen Reducción de resolución")
25 plt.imshow(zoomed)
26 plt.axis('off')
27 plt.show()
28
29
```

Ampliación de una región

Imagen Original



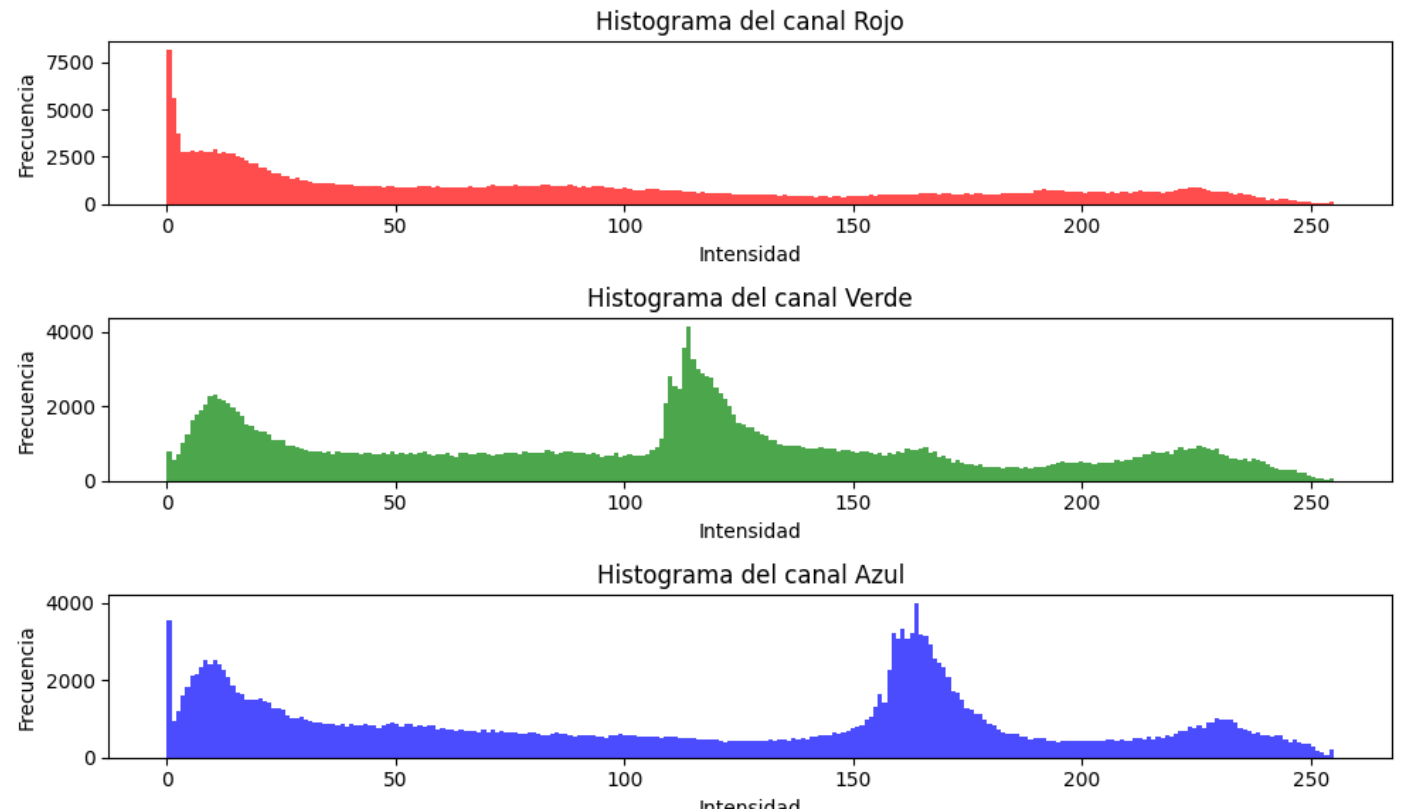
Zoom sobre región central



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Leer y normalizar la imagen
5 img1 = plt.imread('Imagen.jpg') / 255
6
7 # Coordenadas para recortar el centro (por ejemplo, un recorte de 100x100)
8 h, w = img1.shape[:2]
9 zoom_area = 100
10 start_row = h // 2 - zoom_area // 2
11 end_row = h // 2 + zoom_area // 2
12 start_col = w // 2 - zoom_area // 2
13 end_col = w // 2 + zoom_area // 2
14
15 # Recortar región central
16 recorte = img1[start_row:end_row, start_col:end_col]
17
18 # Aumentar tamaño del recorte (ampliación)
19 zoom_factor = 5
20 zoomed = np.kron(recorte, np.ones((zoom_factor, zoom_factor, 1)))
21
22 # Mostrar resultados
23 plt.figure("Zoom hacia adentro")
24
25 plt.subplot(1, 2, 1)
26 plt.title("Imagen Original")
27 plt.imshow(img1)
28 plt.axis('off')
29
30 plt.subplot(1, 2, 2)
31 plt.title("Zoom sobre región central")
32 plt.imshow(zoomed)
33 plt.axis('off')
34
35 plt.show()
```

Histograma de una imagen

- El **histograma de una imagen** es una representación gráfica que muestra cómo se distribuyen los niveles de intensidad (brillo) de los píxeles en una imagen.



Histograma de una imagen

Conceptualmente:

- En una imagen en escala de grises, cada píxel tiene un valor de intensidad entre 0 (negro) y 255 (blanco).
- El histograma **cuenta cuántos píxeles** tienen cada nivel de intensidad.
- En imágenes a color, se puede calcular un histograma **para cada canal** (rojo, verde, azul).

Histograma de una imagen~

¿Qué muestra el histograma?

- **Eje X:** niveles de intensidad (0 a 255).
- **Eje Y:** número de píxeles que tienen ese nivel.

¿Para qué sirve?




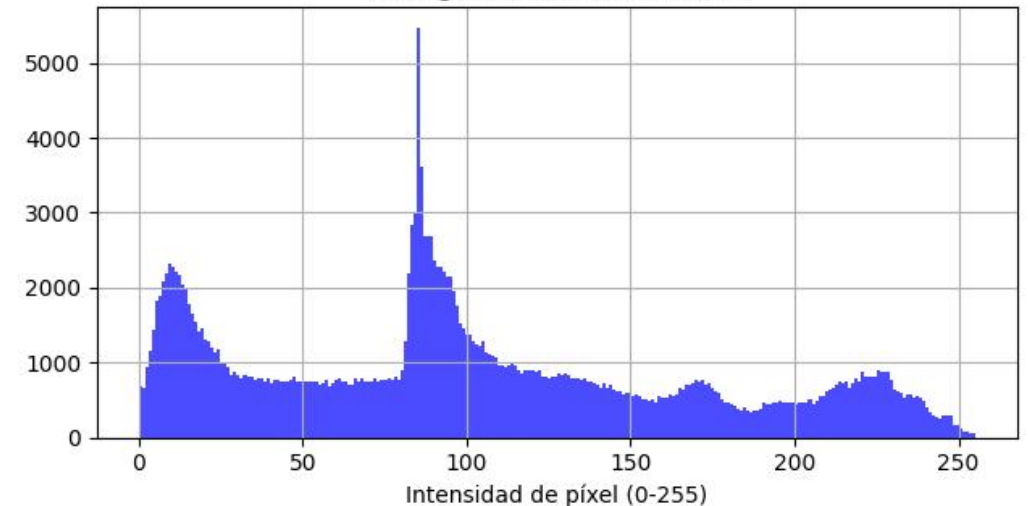
-  **Analizar el brillo:** si la mayoría de valores están a la izquierda, la imagen es oscura; si están a la derecha, es clara.
-  **Evaluar el contraste:** si los valores están concentrados en un rango estrecho, el contraste es bajo.
-  **Corregir imágenes:** se usa para ajustar el brillo, el contraste, la exposición y para hacer ecualización de histograma.

Imagen en escala de grises



Histograma de intensidades



Ejemplo de Histograma usando plt.hist()

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

# Cargar imagen
img = plt.imread('imagen.jpg')

# Asegurar que los valores estén entre 0-255
if img.max() <= 1.0:
    img = (img * 255).astype(np.uint8)

# Separar canales
R = img[..., 0]
G = img[..., 1]
B = img[..., 2]

# Crear figura para mostrar los histogramas por canal
plt.figure(figsize=(10, 6))

# Histograma canal rojo
plt.subplot(3, 1, 1)
plt.hist(R.ravel(), bins=256, color='red', alpha=0.7)
plt.title('Histograma del canal Rojo')
plt.xlabel('Intensidad')
plt.ylabel('Frecuencia')

# Histograma canal verde
plt.subplot(3, 1, 2)
plt.hist(G.ravel(), bins=256, color='green', alpha=0.7)
plt.title('Histograma del canal Verde')
plt.xlabel('Intensidad')
plt.ylabel('Frecuencia')

# Histograma canal azul
plt.subplot(3, 1, 3)
plt.hist(B.ravel(), bins=256, color='blue', alpha=0.7)
plt.title('Histograma del canal Azul')
plt.xlabel('Intensidad')
plt.ylabel('Frecuencia')

plt.tight_layout()
plt.show()
```


Creando nuestro propio Histograma

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def mhist(RGB, tipo='n', color='gray'):
5     """
6     Grafica el histograma de una imagen.
7
8     Parámetros:
9     - RGB: imagen en escala de grises o un canal (2D array)
10    - tipo: 'p' para porcentaje, 'n' para conteo absoluto
11    - color: color del histograma (ej. 'red', 'green', 'blue', 'gray')
12    """
13
14    formato = tipo.lower()
15    filas, columnas = RGB.shape[:2]
16
17    if formato == 'p':
18        factor = (filas * columnas) / 100 # para mostrar en porcentaje
19    else:
20        factor = 1 # para mostrar en conteo
21
22    histograma = np.zeros(256)
23
24    for nivel in range(256):
25        histograma[nivel] = np.sum(RGB == nivel) / factor
26
27    plt.bar(np.arange(256), histograma, color=color, width=1)
28    plt.title("Histograma")
29    plt.xlabel("Nivel de Intensidad (0-255)")
30    plt.ylabel("Porcentaje" if formato == 'p' else "Frecuencia")
31    plt.xlim([0, 255])
32    plt.tight_layout()
33    plt.show()
34
35 # Ejemplo de uso
36 img = plt.imread('imagen.jpg')
37
38 # Asegurar que esté entre 0 y 255
39 if img.max() <= 1.0:
40     img = (img * 255).astype(np.uint8)
41
42 # Convertir a escala de grises si es RGB
43 if img.ndim == 3:
44     gray = np.dot(img[..., :3], [0.299, 0.587, 0.114]).astype(np.uint8)
45 else:
46     gray = img
47
48 # Llamar a la función con opciones
49 mhist(gray, tipo='p', color='gray') # Mostrar en porcentaje
50 # mhist(gray, tipo='n', color='blue') # Mostrar en número de píxeles
```