PROCESAMIENTO DE IMÁGENES

Programa de teoría

1. Adquisición y representación de imágenes.
2. **Procesamiento global de imágenes.**
3. Filtros y transformaciones locales.
4. Transformaciones geométricas.
5. Espacios de color y el dominio frecuencial.
6. Análisis de imágenes.

(c) Ginés García Mateos, http://dis.um.es/profesores/ginesgm
Dept. de Informática y Sistemas, Universidad de Murcia

Tema 2. Procesamiento global de imágenes.

2.1. Tipos de operaciones. Histogramas.
2.2. Operaciones elementales con píxeles.
2.3. Transformaciones del histograma.
2.4. Combinación de imágenes.
2.5. Transformaciones de color.
A.2. Procesamiento global en OpenCV.

2.1. Tipos de operaciones. Histogramas.

- **Pregunta:** ¿Cuál es la base teórica del procesamiento de imágenes? ¿Qué operaciones aplicar?
- **Recordatorio:** ¡una imagen digital no es más que una matriz, o array bidimensional, de números!

90	67	68	75	78
92	87	73	78	82
63	102	89	76	98
45	83	109	80	130
39	69	92	115	154

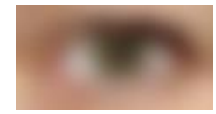
→ Podemos aplicar las mismas operaciones que sobre cualquier número: sumar, restar, multiplicar, dividir, aplicar and, or, máximo, mínimo, integrales, derivadas...

2.1. Tipos de operaciones. Histogramas.

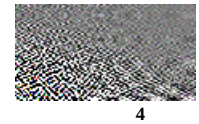
- **Principales tipos de procesamientos de imágenes:**
  - **Operaciones de procesamiento global:** cada píxel es tratado de forma independiente, ya sea con una o con varias imágenes.
  - **Filtros y convoluciones:** se considera la vecindad local de los píxeles.
  - **Transformaciones geométricas:** se modifica el tamaño y forma de las matrices.
  - **Transformaciones lineales:** Fourier, wavelets, etc.











2.1. Tipos de operaciones. Histogramas.

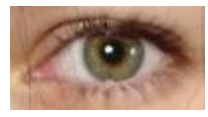
- **Operaciones de procesamiento global:**
  - **Aritméticas**: sumar, restar, multiplicar, máximo, etc.
    - Unarias: una sola imagen y un valor constante.
    - Binarias: con dos imágenes.
  - **Booleanas**: and, or, not, etc.
    - Unarias: una sola imagen y una constante.
    - Binarias: con dos imágenes.
  - **Otras** transformaciones generales:
    - Transformaciones de histograma.
    - Transformaciones de color.
    - Binarización, etc.
- Cada operación tendrá un **significado**, **utilidad** y **aplicaciones** específicos. +

2.1. Tipos de operaciones. Histogramas.

- Supongamos una imagen de **entrada A** y una imagen **resultado R**.
- Una operación global (píxel a píxel) se puede expresar como **una función**:

$R(x,y) := f(A(x,y))$ → El valor del píxel resultante es función de (y sólo de) el píxel correspondiente de entrada.

- **Ejemplo. Invertir.** $R(x,y) := 255 - A(x,y)$

Imagen A  → Imagen R 

2.1. Tipos de operaciones. Histogramas.

$$R(x,y) := f(A(x,y)), \quad \forall\ (x,y)$$

- **Comparar con:**
  - **Filtros y convoluciones:** el valor de un píxel depende de la vecindad local de ese píxel:

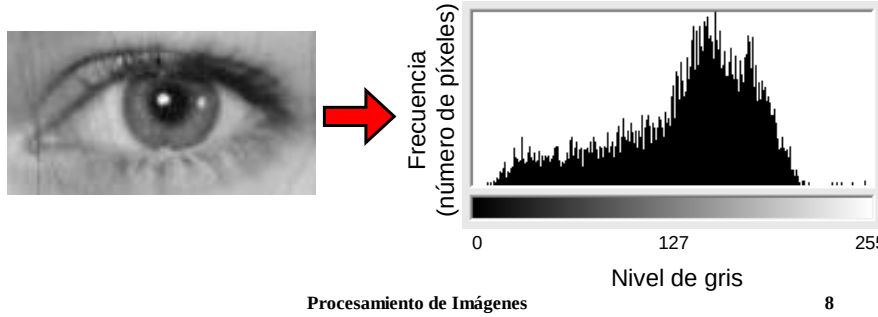
    $R(x,y) := f(A(x-k,y-k), ..., A(x,y), ..., A(x+k,y+k))$
  - **Transformaciones geométricas:** el valor de un píxel depende de píxeles situados en otras posiciones:

    $R(x,y) := A(f_1(x,y), f_2(x,y))$
  - **Transformaciones lineales:** el valor de un píxel puede depender de todos los píxeles de la imagen:

    $R(x,y) := f(A, x, y)$

2.1. Tipos de operaciones. Histogramas.

- Para comprender el significado de muchas transformaciones y saber cuál conviene aplicar se usan histogramas.
- ¿Qué es un histograma? → Repasar estadística...
- Un **histograma** representa gráficamente una distribución de frecuencias.
- **Histograma de una imagen:** representa las frecuencias de los diferentes valores de gris en la imagen.



2.1. Tipos de operaciones. Histogramas.

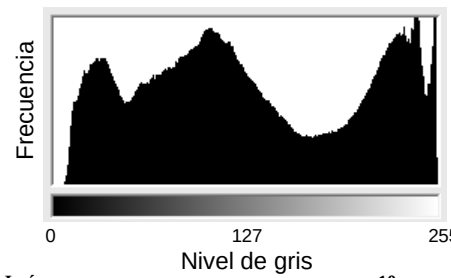
- **Algoritmo.** Cálculo de un histograma.

- **Entrada.** A: imagen de ancho x alto
- **Salida.** Histograma: array [0,...,255] de entero
- **Algoritmo:**

```
Histograma[]:= 0
para y:= 0, ..., alto-1 hacer
    para x:= 0, ..., ancho-1 hacer
        Histograma[A(x,y)]:= Histograma[A(x,y)]+1
```

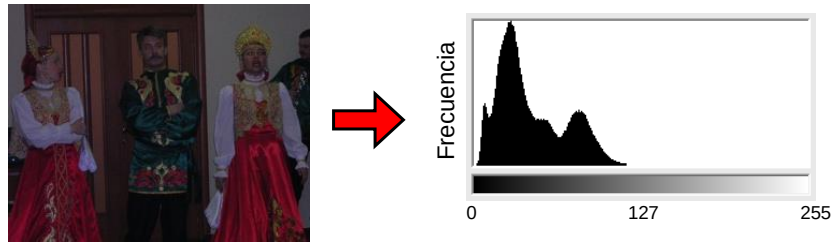

2.1. Tipos de operaciones. Histogramas.

- Los histogramas son una herramienta importante en **análisis de imágenes**: ¿es buena la calidad de una imagen?, ¿sobra luz?, ¿falta contraste?
- Ayudan a decidir cuál es el procesamiento más adecuado para **mejorar la calidad** de una imagen...
  - Tanto **cualitativamente** (qué operación aplicar),
  - Como **cuantitativamente** (en qué cantidad).
- En principio, una buena imagen debe producir un **histograma** más o menos **uniforme** y repartido en todo el rango de valores.

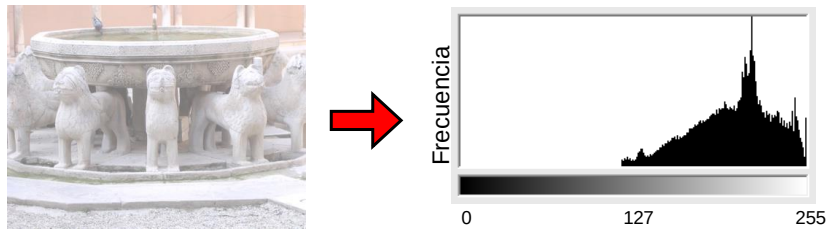


2.1. Tipos de operaciones. Histogramas.

- **Ejemplo 1.** La imagen es muy oscura. Falta luz.

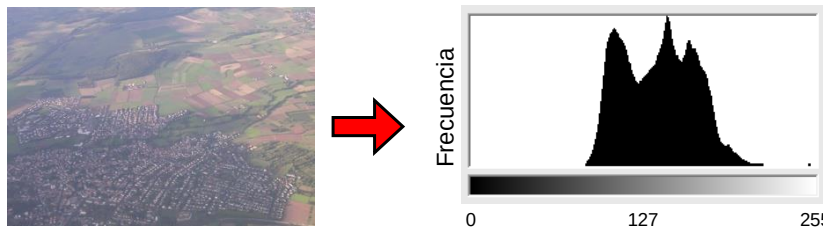


- **Ejemplo 2.** La imagen es muy clara. Sobra brillo.

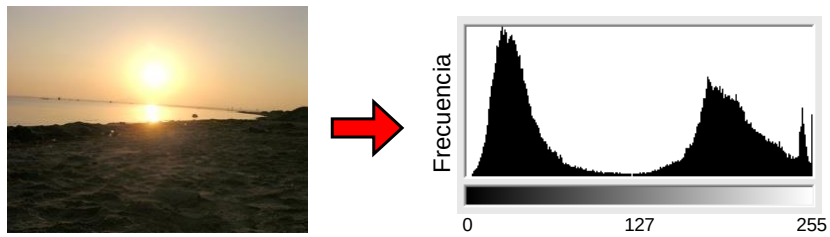


2.1. Tipos de operaciones. Histogramas.

- **Ejemplo 3.** La imagen tiene poco contraste.

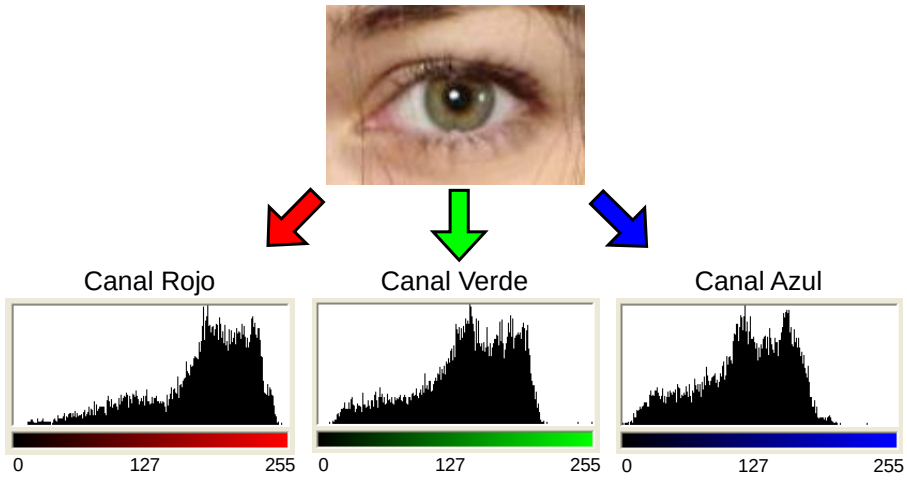


- **Ejemplo 4.** Hay mucho contraste, pocos medios tonos.



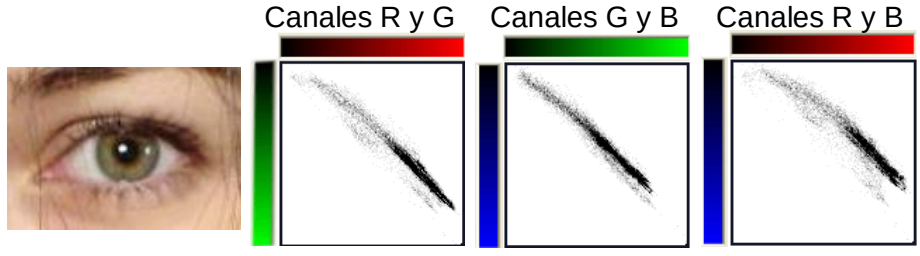
2.1. Tipos de operaciones. Histogramas.

- **Histogramas de color.** En imágenes multicanal podemos obtener un histograma de cada canal por separado.



2.1. Tipos de operaciones. Histogramas.

- O, también, podemos calcular histogramas conjuntos, en 2 ó 3 dimensiones.

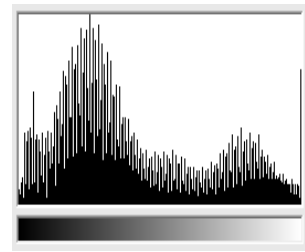


- Estos histogramas aportan información sobre los rangos de **colores más frecuentes** en la imagen.
- En teoría, el histograma es de 256x256 celdas (*bins*).
- Pero, para obtener buenos resultados, mejor usar un **número reducido de celdas**. Por ejemplo 64x64 ó 32x32.

2.1. Tipos de operaciones. Histogramas.

- Uso de histogramas para mejorar la calidad de las imágenes.
- **Ejemplo.** El histograma indica tonos muy oscuros.

- **Solución.** Aplicar un operador que "estire" el histograma.



2.2. Operaciones elementales con píxeles.

- **A**: imagen de entrada.
- **R**: imagen resultante (del mismo tamaño que A).

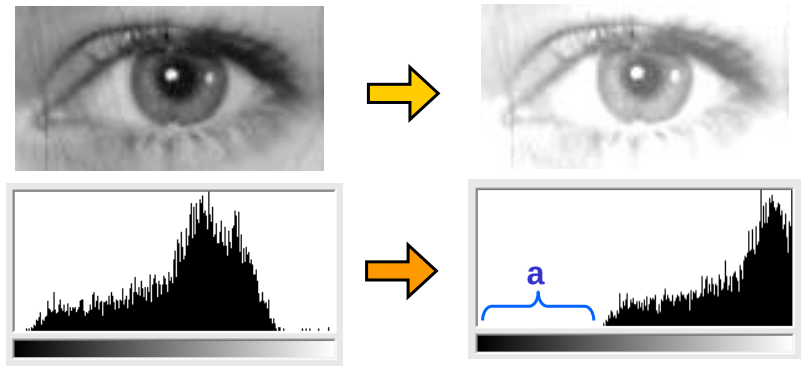
Operaciones unarias:

- Sumar una constante: $R(x, y) := A(x, y) + a$
- Restar una constante: $R(x, y) := A(x, y) - a$
- Multiplicar por una constante: $R(x, y) := b \cdot A(x, y)$
- Dividir por una constante: $R(x, y) := A(x, y)/b$
- Transformación lineal genérica: $R(x, y) := bA(x, y)+a$
- Transformación de gama: $R(x, y) := A(x, y)^c$
- Cualquier función $\mathbf{N} \rightarrow \mathbf{N}$: $R(x, y) := f(A(x,y))$

2.2. Operaciones elementales con píxeles.

Sumar una constante: R(x, y):= A(x, y) + a

- **Significado:** incrementar el brillo de la imagen en la cantidad indicada en **a**.
- El histograma se desplaza a la derecha en **a** píxeles.



2.2. Operaciones elementales con píxeles.

- **Ojo:** la suma puede ser mayor que 255...
- La operación debería comprobar el overflow:
  si A(x, y) + a > 255 **entonces** R(x, y):= 255
  sino R(x, y):= A(x, y) + a
- Esto se debe hacer también en las demás operaciones, comprobando si el valor es <0 ó >255.
- Coloquialmente, un píxel "por encima" de 255 o por debajo de 0 se dice que está **saturado**.
- La saturación supone una pérdida de información.

Ejemplo de imagen muy saturada

2.2. Operaciones elementales con píxeles.

- En imágenes en color, la suma se realiza sobre los tres canales (R, G y B) y con el **mismo valor**.

  R(x, y).R:= A(x, y).R + a R(x, y).G:= A(x, y).G + a

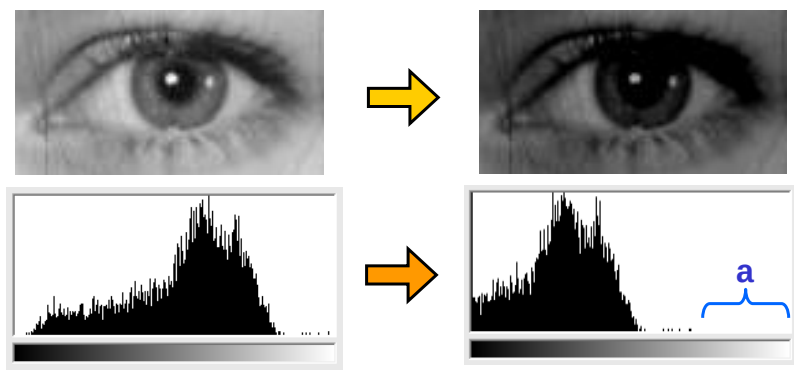
  R(x, y).B:= A(x, y).B + a


- ¿Qué ocurre si se suma un valor distinto a cada canal?

2.2. Operaciones elementales con píxeles.

Restar una constante: R(x, y):= A(x, y) - a

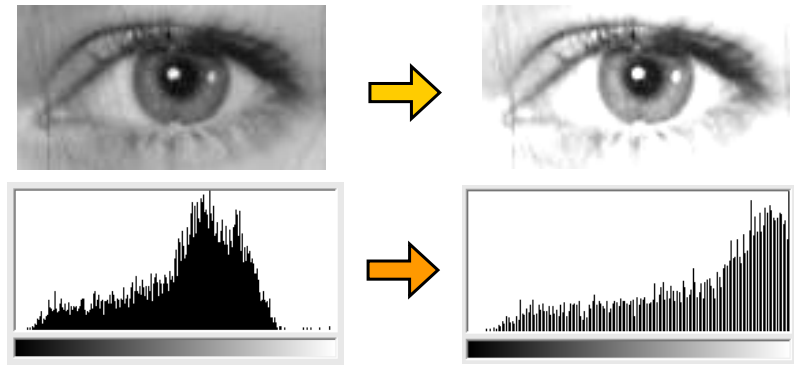
- **Significado:** decrementar el brillo de la imagen en la cantidad indicada en **a**.
- El histograma se desplaza a la izquierda en **a** píxeles.



2.2. Operaciones elementales con píxeles.

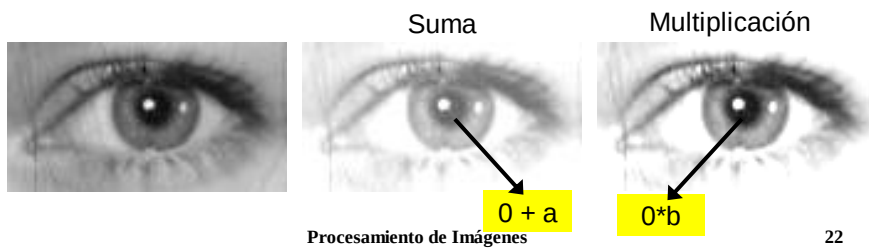
Multiplicar por una constante: R(x, y):= b·A(x, y)

- **Significado:** aumentar la intensidad de la imagen en **b**.
- El histograma se “estira” hacia la derecha.



2.2. Operaciones elementales con píxeles.

- Tanto en la suma como en la multiplicación, se aumenta el nivel de gris de los píxeles, pero de forma distinta.
  - **En la suma**, el parámetro **a** (entero) indica el número de niveles de gris a aumentar: de -255 a 255.
  - **En el producto**, el parámetro **b** (real) indica el factor a multiplicar.
    - b=1 → Ningún cambio
    - b=2 → Se duplica el valor de gris. Los píx. >127 se saturan.
    - b=0,5 → Se “encoge” a la mitad el histograma.

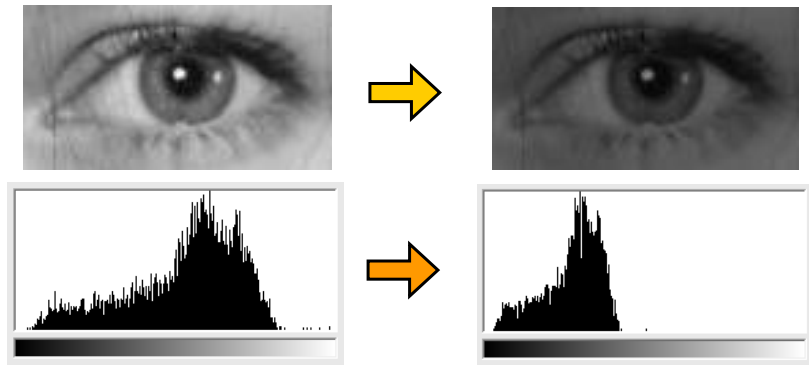


2.2. Operaciones elementales con píxeles.

Dividir por una constante: R(x, y):= A(x, y) / b

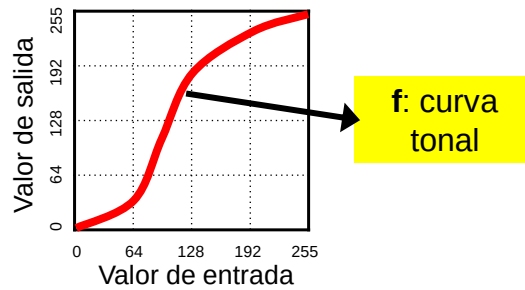
= Multiplicar por 1/b ... ¡obviamente!

- El histograma se “encoge”.



2.3. Transformaciones del histograma.

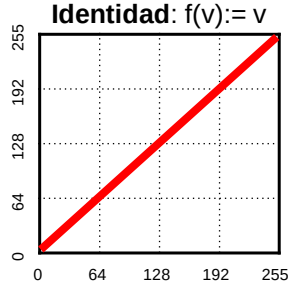
- Las transformaciones elementales se pueden ver como **funciones f: N → N**.
- **Interpretación:** para cada valor de gris de entrada hay un valor de salida.



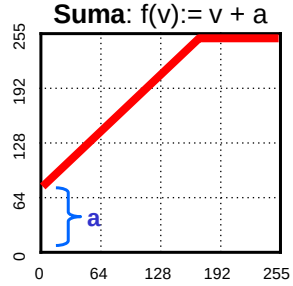
- Se puede usar cualquier función **f**.
- La transformación hace que se modifique el histograma.

2.3. Transformaciones del histograma.

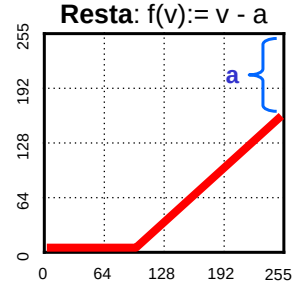
Identidad: f(v):= v



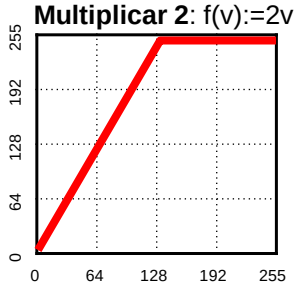
Suma: f(v):= v + a



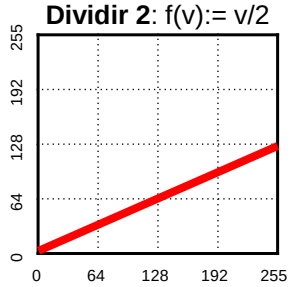
Resta: f(v):= v - a



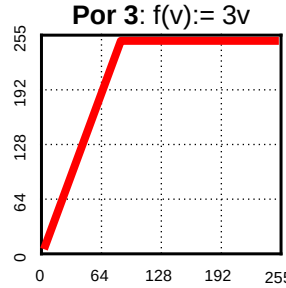
Multiplicar 2: f(v):=2v



Dividir 2: f(v):= v/2



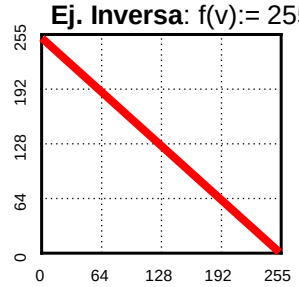
Por 3: f(v):= 3v



2.3. Transformaciones del histograma.

- En general, podemos definir una **transformación lineal genérica** de la forma:
  f(v):= b·v + a

Ej. Inversa: f(v):= 255 - v

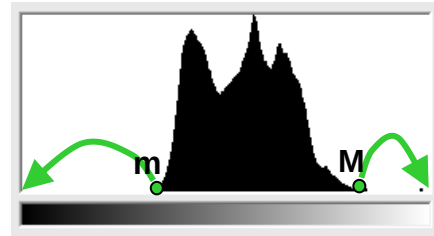


- Pero la transformación también puede ser **no lineal**: cuadrática, polinomial, exponencial, logarítmica, escalonada, etc.
- ¿Cómo decidir cuál es la transformación más adecuada? → Usar el histograma.

2.3. Transformaciones del histograma.

- Normalmente, interesa “estirar” el histograma, para conseguir que aparezca todo el rango de valores.
- **Idea:** definir una transformación lineal tal que el histograma resultante vaya de 0 a 255.
- **Ajuste lineal** o **estiramiento** ***(stretch)*** del histograma:
  - Buscar el valor mínimo del histograma: **m**
  - Buscar el valor máximo: **M**
  - f(v):= (v-m)*255/(M-m)

Nota: Esto es una simple regla de 3

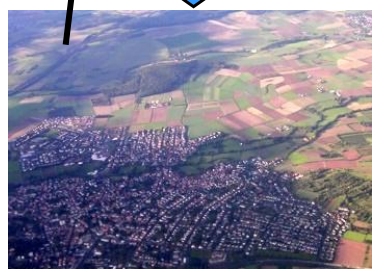


2.3. Transformaciones del histograma.

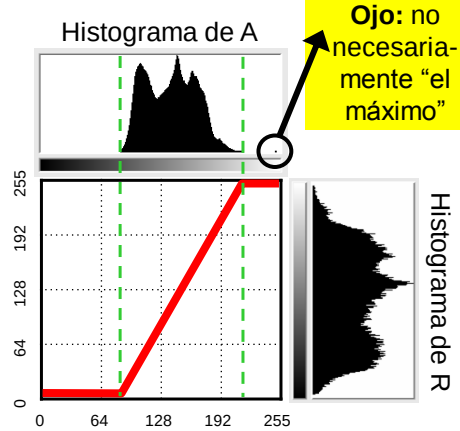
A

Para imágenes en color, se aplica la misma función a los tres canales (R,G,B)

R

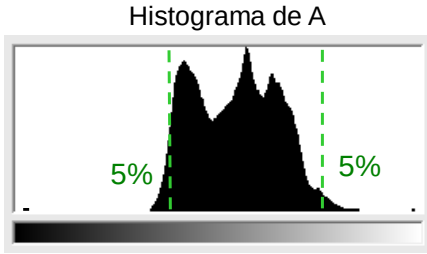


- **Ejemplo.** m= 86, M= 214
  R(x,y):= (A(x,y)-86)*1,99



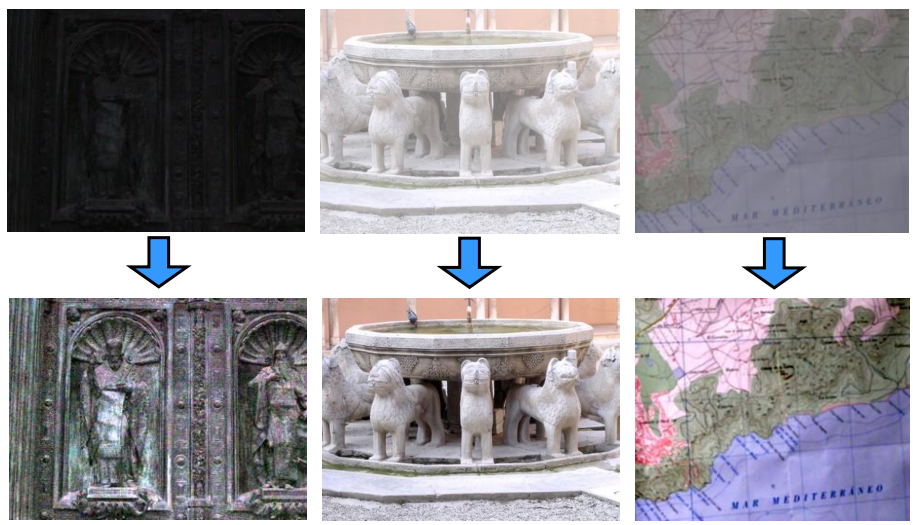
2.3. Transformaciones del histograma.

- **Cuidado:** un simple píxel con valor muy alto o muy bajo puede hacer que el ajuste del histograma sea muy malo.
- Por ejemplo, si hay un píxel con valor 0 y otro con 255, la transformación sería la identidad (la imagen no cambia).
- **Solución:** en lugar de mínimo y máximo, ajustar usando dos percentiles del histograma (p. ej. 10%-90%, ó 5%-95%).



2.3. Transformaciones del histograma.

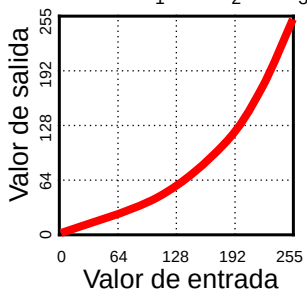
- Más ejemplos de estiramiento lineal del histograma.



2.3. Transformaciones del histograma.

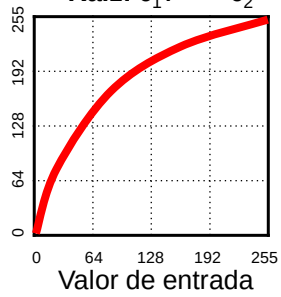
- La **transformación de histograma** puede tomar cualquier forma (no necesariamente lineal).
- **Ejemplos.**

Parábola: $c_1v^2 + c_2v + c_3$



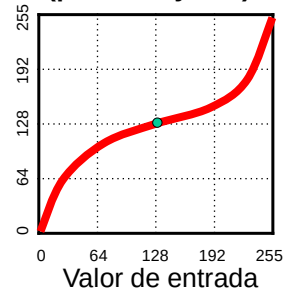
Resultado: oscurecer los medios tonos.

Raíz: $c_1v^{0.5} + c_2$



Resultado: aclarar los medios tonos.

Dos trozos de curva (parábola y raíz)

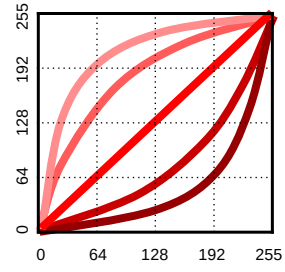


Resultado: aclarar tonos oscuros y oscurecer los claros.

2.3. Transformaciones del histograma.

- Elevar a 2, elevar a 1/2, ...
- Se define la **transformación de gama** como:

$f(v) := 255 \cdot (v/255)^{1/GAMA}$

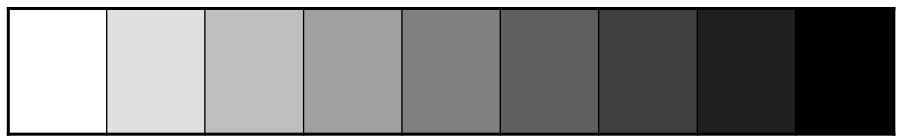


Gama 0,5 Gama 0,75 Gama 1 Gama 2 Gama 4

2.3. Transformaciones del histograma.

- La diferencia entre diferentes dispositivos (televisores, cámaras, escáneres) se modela con una **transformación de gama**.
- Si el comportamiento del dispositivo fuera perfectamente lineal, Gama = 1.

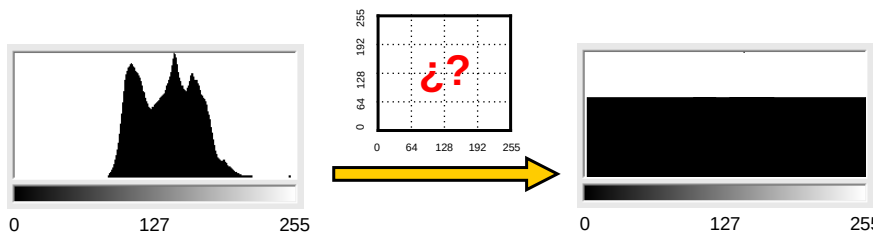
Blanco Negro



- ¿Dónde está el 50% de gris? ¿Es la escala lineal?
- ¿Dónde estaría si tomáramos una foto?

2.3. Transformaciones del histograma.

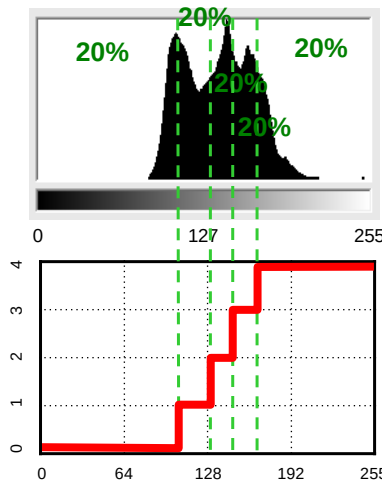
- Otra transformación habitual es la ecualización del histograma (del latín *aequalis* = igual).
- **Ecualización del histograma:** es una transformación definida de forma que el histograma resultante se reparte uniformemente en todo el rango de grises.



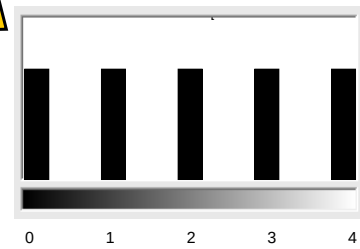
- En este caso se usa una función escalonada:
  f: array [0..255] de byte

2.3. Transformaciones del histograma.

- ¿Cómo definir **f** para conseguir la ecualización?
- **Idea:** suponer que a la salida hay 5 niveles de gris.



para todo píxel (x,y) de R **hacer**
R(x,y):= f[A(x,y)]



2.3. Transformaciones del histograma.

- **Algoritmo.** Cálculo de la función de ecualización del histograma.
- **Entrada.** Histograma: array [0,...,255] de entero
  np: entero (número total de píxeles = m_x*m_y)
- **Salida.** f: array [0,...,255] de byte

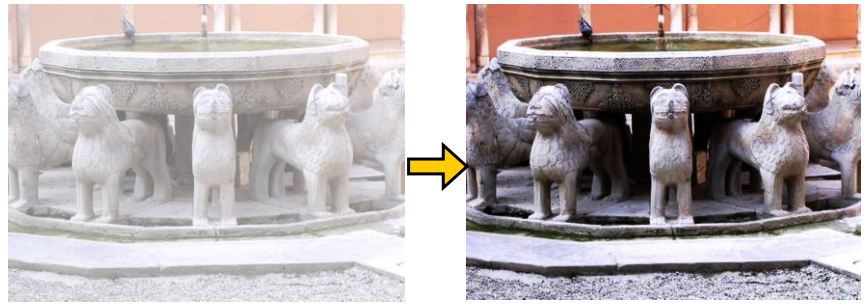
- **Algoritmo:**

```
f[0]:= 0
acumulado:= Histograma[0]
para i:= 1, ..., 254 hacer
    f[i]:= acumulado*255/np
    acumulado:= acumulado + Histograma[i]
finpara
f[255]:= 255
```

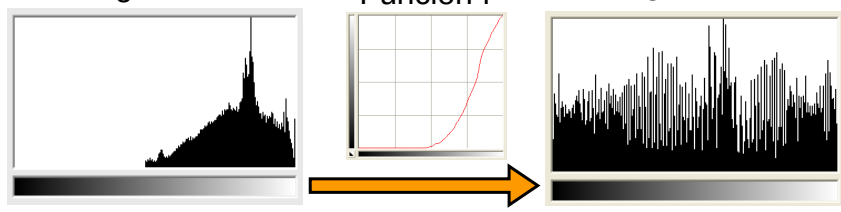
La función de ecualización es la **integral del histograma**, escalada por el factor 255/**np**.

2.3. Transformaciones del histograma.

Imagen de entrada (A) **Imagen ecualizada (R)**

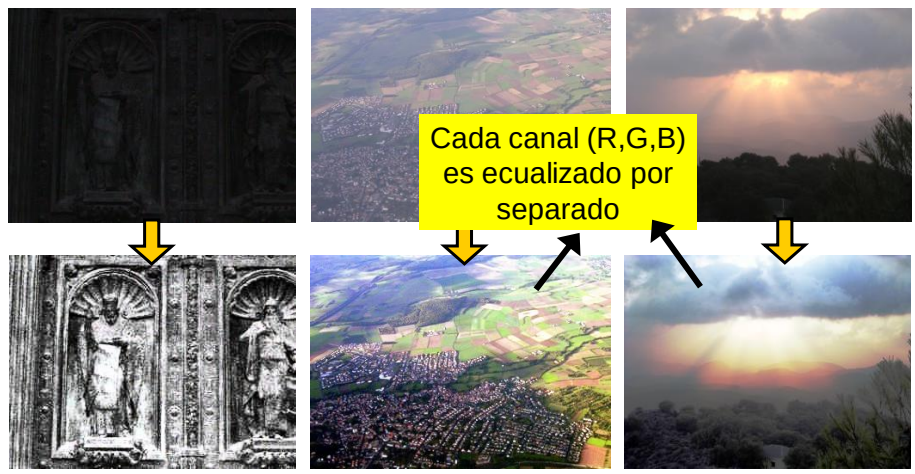


Histograma de A Función f Histograma de R



2.3. Transformaciones del histograma.

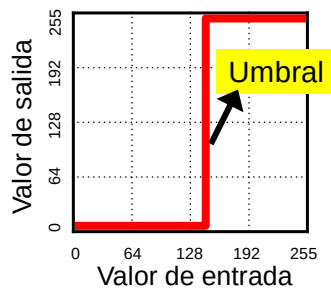
- **Ejemplos.** Ecualización del histograma.



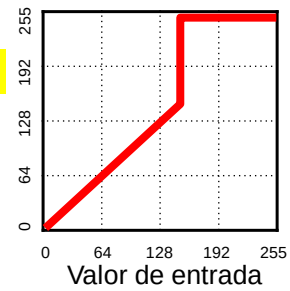
- Cuidado, en algunos casos los resultados pueden ser *artificiosos*.

2.3. Transformaciones del histograma.

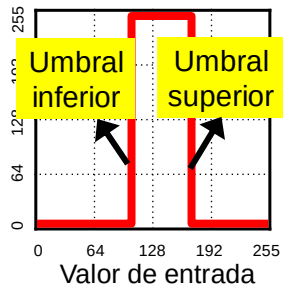
- **Umbralización de imágenes.** En algunas aplicaciones puede ser interesante convertir la imagen en binaria, o recortar cierto rango de valores.
- Las funciones tienen las siguientes formas:



Umbralizar la imagen con valor cte.



Cortar un rango y mantener el resto



Seleccionar un rango

2.3. Transformaciones del histograma.

- Las funciones serán del estilo:

  f(v):= **si** v > umbral1 **entonces** g(v)
  sino h(v)

- **Transformación de binarización** (saturar a 0 ó 255).
  f(v):= **si** v < umbral **entonces** 0 **sino** 255
- **Ejemplo 1**. La binarización se suele aplicar en OCR.

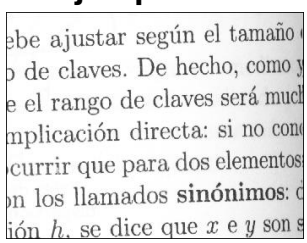
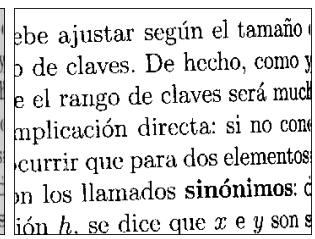
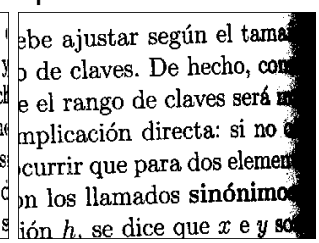


Imagen de entrada (256 grises)



Umbral = 160



Umbral = 215

2.3. Transformaciones del histograma.

- **Ejemplo 2**. Segmentación de objetos.



Imagen de entrada Umbralizar, u = 42 Umbralizar, u = 180

- La separación del objeto del fondo se llama **segmentación**.
- La umbralización se puede usar para segmentar...
- ... aunque por sí sola no suele funcionar muy bien.

Cortar rango (192, 255)

2.3. Transformaciones del histograma.

Conclusiones:

- Una **transformación elemental** se puede ver desde distintas perspectivas:
  - Como una **función** unidimensional: **f: N → N**
  - Como una **curva tonal**.
  - Como una **modificación** del **histograma**.
- La característica fundamental es que cada píxel se trata **independientemente** de los demás.
- Los **histogramas** son útiles para encontrar la transformación adecuada.
- En imágenes RGB, aplicamos **la misma operación** a los 3 canales para que se mantenga el color.

2.4. Combinación de imágenes.

- **Combinación de imágenes:** utilizar dos o más imágenes de entrada para producir una imagen de salida.
- **Entrada:** imágenes A y B.
- **Salida:** imagen R.

R(x, y):= f(A(x,y), B(x,y))

El valor del píxel resultante es función de los píxeles de A y B en la misma posición

En principio, todas las imágenes deben ser del mismo tamaño

- Posibles operaciones de combinación:
  - **Booleanas**: and, or, xor, not
  - **Aritméticas**: suma, resta, producto/división, media
  - **Relacionales**: máximo, mínimo

2.4. Combinación de imágenes.

- **Operadores booleanos:**
  - R(x, y):= A(x,y) AND B(x,y)
  - R(x, y):= A(x,y) OR B(x,y)
  - R(x, y):= A(x,y) XOR B(x,y)
  - R(x, y):= NOT A(x,y) AND B(x,y)
  - R(x, y):= A(x,y) OR NOT B(x,y)
  - ...
- Estos operadores tienen sentido cuando al menos una de las imágenes es binaria.
  - Negro (0) = FALSE
  - Blanco (1 ó 255) = TRUE

2.4. Combinación de imágenes.

- **Ejemplos**. Operadores booleanos.

Imagen de entrada **A**

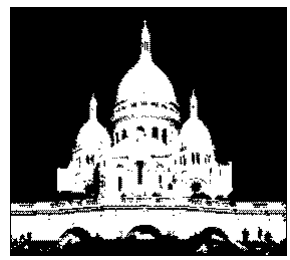
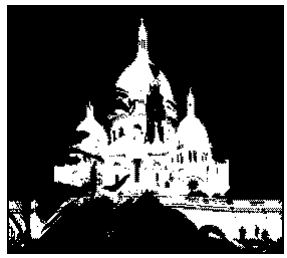
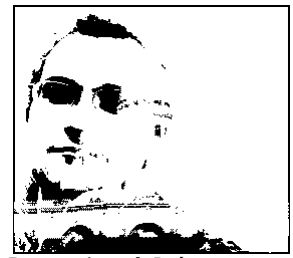


Imagen de entrada **B**

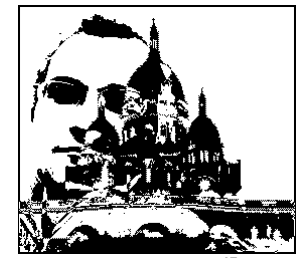
A AND **B**



A OR **B**



A XOR **B**



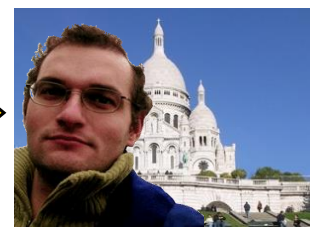
2.4. Combinación de imágenes.

- En imágenes no binarias no tienen mucho sentido... ¿Cómo se interpretan?

A_{Gris} AND B_{Gris}

A_{RGB} AND B_{RGB}

- Las operaciones binarias aparecen en análisis de imágenes, y también para trabajar con **máscaras y recortes** de objetos.



2.4. Combinación de imágenes.

- **Imágenes de entrada.**

A **B** **C**



- ¿Cómo conseguir el montaje de la página anterior?
- R:= (B AND NOT C) OR (A AND C)

2.4. Combinación de imágenes.

1. T1:= B AND NOT C

B  **NOT C**   **T1**

2. T2:= A AND C

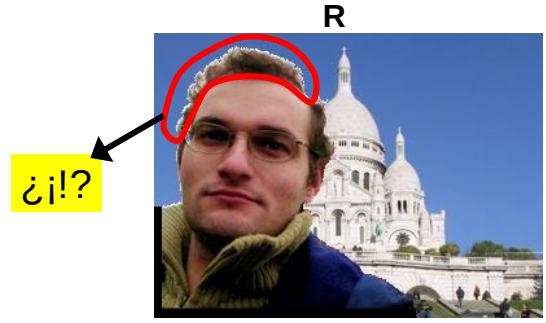
A  **C**   **T2**

3. R:= T1 OR T2

T1  **T2**   **R**

2.4. Combinación de imágenes.

- La imagen binaria (**C**) se suele denominar **máscara**.
- La máscara permite segmentar el objeto de interés.



Cuestiones:

- ¿Cómo crear la máscara de forma automática?
- La zona del pelo no se mezcla bien con el fondo. ¿Cómo evitar este problema?

2.4. Combinación de imágenes.

- **Operaciones aritméticas:**
  - R(x, y):= A(x, y) + B(x, y)
  - R(x, y):= A(x, y) - B(x, y)
  - R(x, y):= (A(x, y) + B(x, y))/2
  - R(x, y):= a·A(x, y) + (1-a)·B(x, y)
  - R(x, y):= A(x, y)·B(x, y)·c

- Se usan en generación y análisis de imágenes.
- Cuidado con los problemas de **saturación**.
- En imágenes binarias son equivalentes (en su mayoría) a los operadores booleanos.

2.4. Combinación de imágenes.

Sumar dos imágenes: R(x, y):= A(x, y) + B(x, y)

- **Significado:** mezclar las dos imágenes.

A

B

R

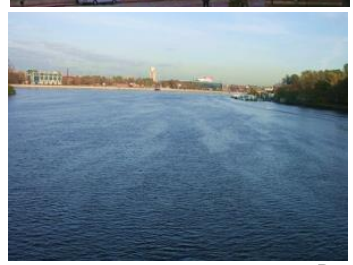
- **Ojo:** [0..255] + [0..255] = [0..510]

2.4. Combinación de imágenes.

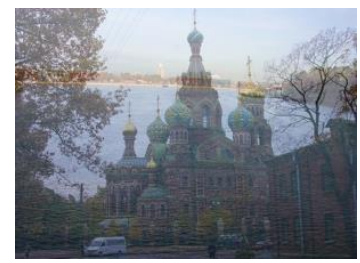
- Para evitar la saturación se puede usar la media.

Media de 2 imágenes: R(x, y):= (A(x,y)+B(x,y))/2

A



B



R

- **Significado:** las imágenes son semitransparentes (al 50%).

2.4. Combinación de imágenes.

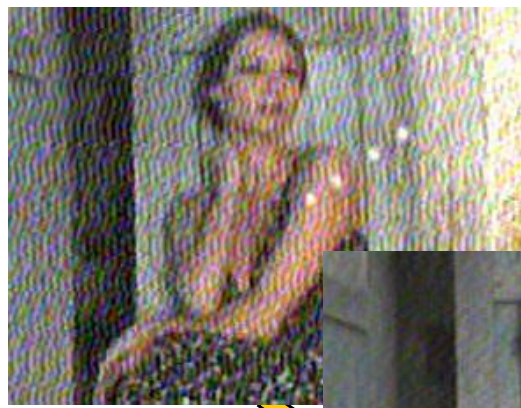
- De forma similar, se puede definir la **media ponderada**.

Media ponderada: R(x,y):= a·A(x,y) + (1-a)·B(x,y)

a = 0,25 a = 0,5 a = 0,75

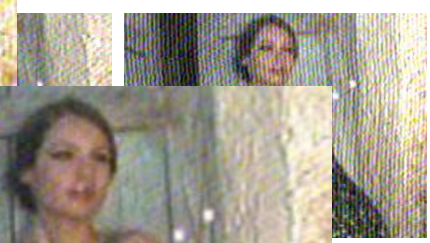
- La media ponderada se puede usar para crear una **transición suave** entre imágenes (o vídeos).

2.4. Combinación de imágenes.



usar para **acumular**

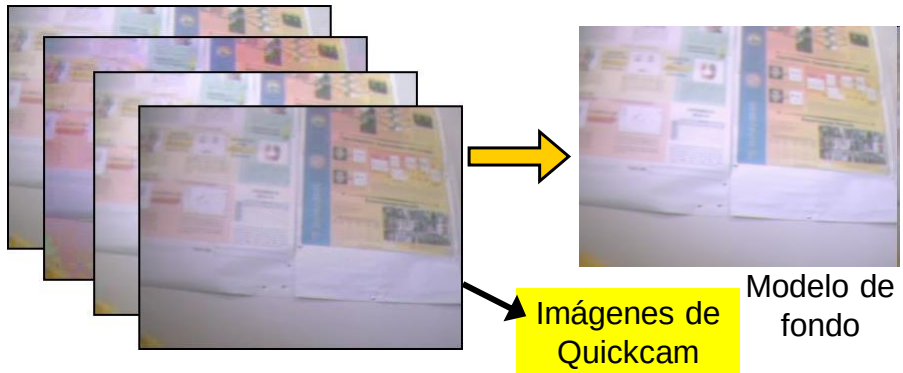
con mucho ruido de una
a con menos ruido.



Imágenes capturadas de TV

2.4. Combinación de imágenes.

- **Ejemplo 2.** Crear un modelo de fondo de una escena, acumulando varias imágenes.



Imágenes de Quickcam

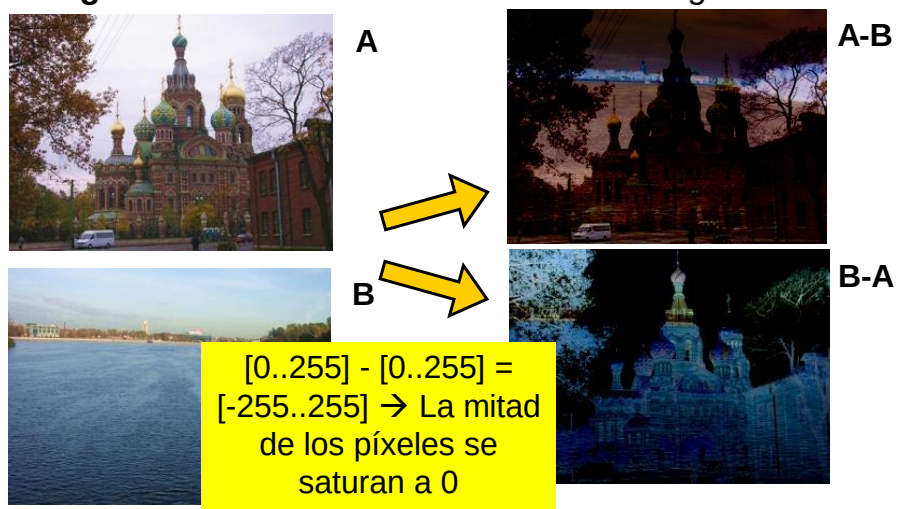
Modelo de fondo

- **Idea:** si además de la media en cada píxel calculamos también la varianza, podríamos tener un modelo gaussiano del fondo (N(μ,σ)).

2.4. Combinación de imágenes.

Restar dos imágenes: R(x, y):= A(x, y) - B(x, y)

- **Significado:** obtener diferencia entre imágenes.



[0..255] - [0..255] = [-255..255] → La mitad de los píxeles se saturan a 0

2.4. Combinación de imágenes.

Restar dos imágenes, manteniendo el rango de salida: R(x, y):= (A(x, y) - B(x, y))/2 + 128

A

(A-B)*

B

(B-A)*

2.4. Combinación de imágenes.

- Muchas veces lo que interesa es conocer la diferencia entre las imágenes. → **Solución:** tomar valor absoluto de la resta.

Diferencia: R(x, y):= abs(A(x, y) - B(x, y))

A

B

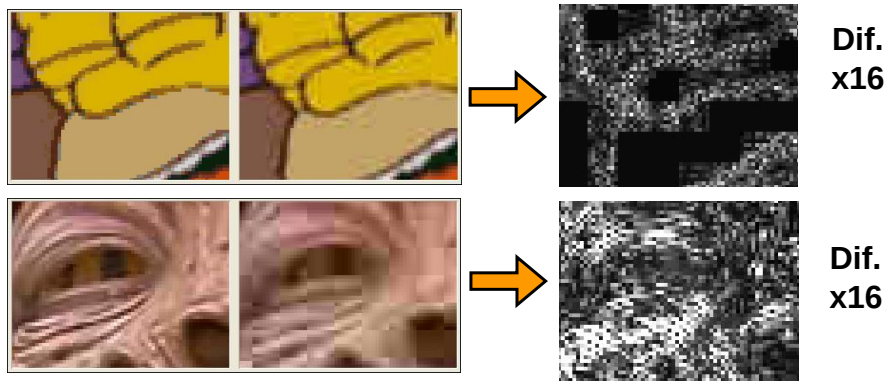
R

¿? Son muy distintas...

- **Píxel negro:** las dos imágenes son iguales en ese píxel.
- Cuando más clara es una zona, más se diferencian las imágenes.

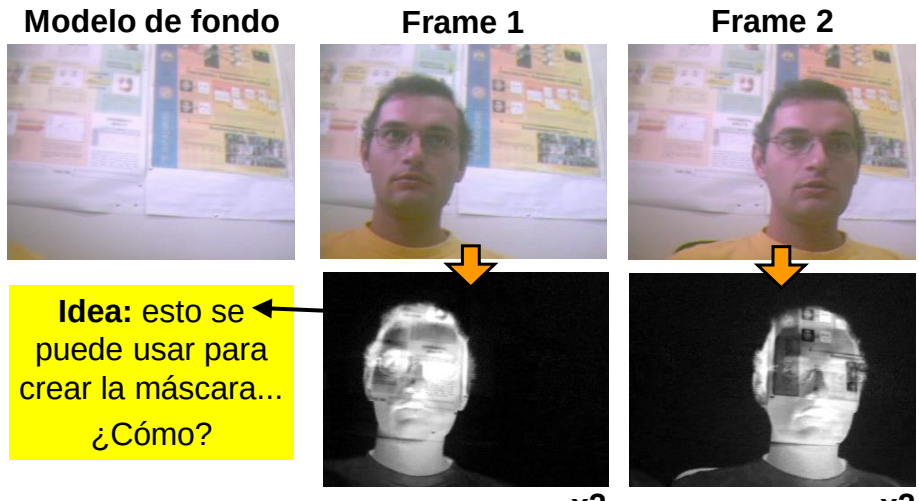
2.4. Combinación de imágenes.

- **Aplicaciones de la diferencia:** encontrar variaciones entre imágenes que, en principio, deberían ser parecidas.
- **Ejemplo 1.** Analizar la pérdida de información al comprimir una imagen. Por ejemplo, con JPEG.



2.4. Combinación de imágenes.

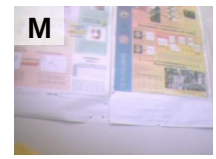
- **Ejemplo 2.** Segmentación del fondo de una escena.
- Tenemos un fondo (imagen media) y una nueva imagen.

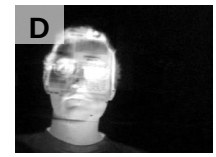


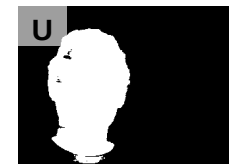
2.4. Combinación de imágenes.

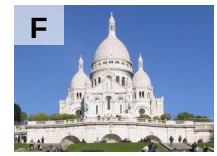
- **Proceso.**

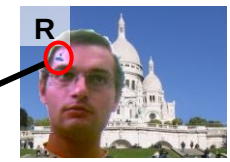
1. Obtener el modelo de fondo **M**.
2. Para cada imagen **A** del vídeo.
3. Calcular la diferencia: **D** = abs(M-A).
4. Umbralizar la imagen con un valor adecuado. **U** = umbralizar(D, x).
5. Sea **F** el nuevo fondo.
6. **R**:= (F AND NOT U) OR (A AND U)











¿Cómo arreglar eso?

2.4. Combinación de imágenes.

- **Ejemplo 3.** Detección de movimiento en vídeo.
- Dada una secuencia de vídeo, queremos saber si se ha producido alguna modificación, y en qué zonas de la imagen (“encuentra las 7 diferencias”).

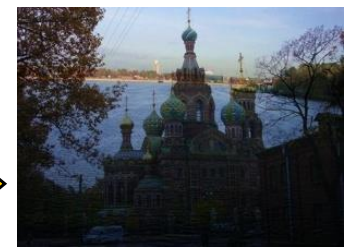
Frame 1 **Frame 2** **Diferencia x2**

- ¿Qué objetos se han movido y en qué dirección?

2.4. Combinación de imágenes.

Producto imágenes: R(x, y):= A(x, y)·B(x, y)/255

A



A·B

B

- Necesario escalar el resultado (dividir por 255).
- Efecto de **mezcla**, similar a la suma, pero conceptualmente más próximo a un AND...

2.4. Combinación de imágenes.

División imágenes: R(x, y):= 255·A(x, y)/B(x, y)

A

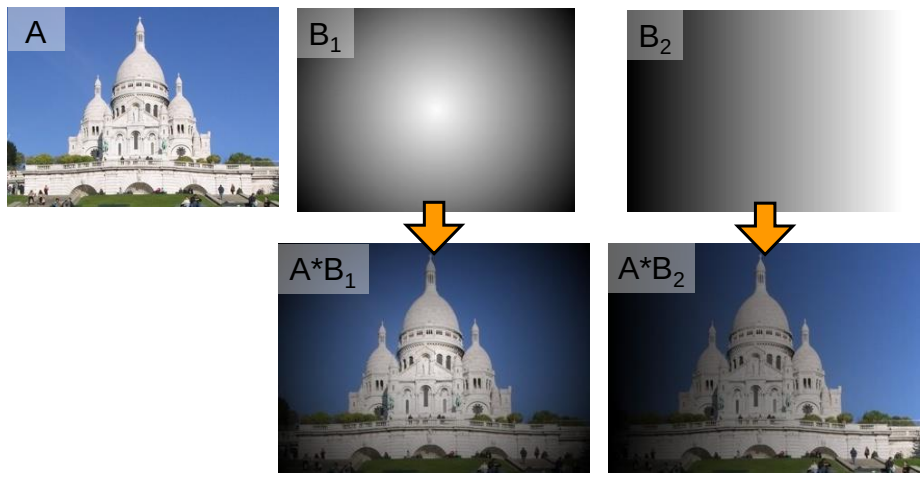
A/B

B

- También es necesario escalar el resultado (multiplicar por 255).
- ¿Cuál es interpretación del resultado?

2.4. Combinación de imágenes.

- **Ejemplo 1.** Realizar una transformación de intensidad distinta para cada píxel.



2.4. Combinación de imágenes.

- Estos mismos tipos de imágenes se pueden usar para hacer sumas, restas, divisiones, etc.
- **Ejemplo.** $R(x, y) := A(x, y) \cdot B(x, y)/128$
  - Si $B(x, y) = 128$ el píxel de A no cambia.
  - Si $B(x, y) < 128$ el píxel se oscurece.
  - Si $B(x, y) > 128$ el píxel se aclara.

- El producto es también la base en la idea de **máscara** o **selección difusa**.
- **Idea:** una imagen se compone de distintos elementos o capas, que tienen definido cierto nivel de transparencia.

2.4. Combinación de imágenes.

- **Ejemplo 2.** Mezcla y combinación de imágenes. Queremos combinar dos imágenes, por ejemplo, para poner una etiqueta descriptiva en una foto. Una imagen binaria sirve de **máscara**: 0 = fondo, 1 = etiqueta.

- **Resultado:**

  R:= (A AND NOT M) OR (B AND M)

No me convence... mejor un reborde suave (difuminado)

2.4. Combinación de imágenes.

- **Solución.** Usar una máscara "suave", una imagen en gris: 0 = transparente, 255 = opaco. Combinar: sumas y productos.

- **Resultado:**

  R:= A·(255-N)/255 + B·N/255

Producto de imágenes

2.4. Combinación de imágenes.

- **Indicaciones sobre el ejemplo 2.**
  - La “mascara suave” es la idea del **canal alfa**.
  - RGB → RGBA, donde el canal A indica el **grado de opacidad** de un píxel (0= transparente, 255= opaco).
  - **Uso:** definimos imágenes, con sus canales alfa, y las componemos poniendo unas sobre otras.
  - La **composición de imágenes** con canal alfa es básicamente una media ponderada como hemos visto.
  - En el modo binario, muchas herramientas incorporan las ideas de **máscara**, **selección**, **región de interés** (cuando es rectangular) o **canal de interés** (en multicanal).
  - No necesitamos trabajar con **operaciones booleanas**, aunque implícitamente es lo que hay subyacente.

2.4. Combinación de imágenes.

Otras operaciones no lineales

- **Mínimo de 2 imágenes.** R(x, y):= min(A(x, y), B(x, y))

- **Máximo de 2 imágenes.** R(x, y):= max(A(x, y), B(x, y))

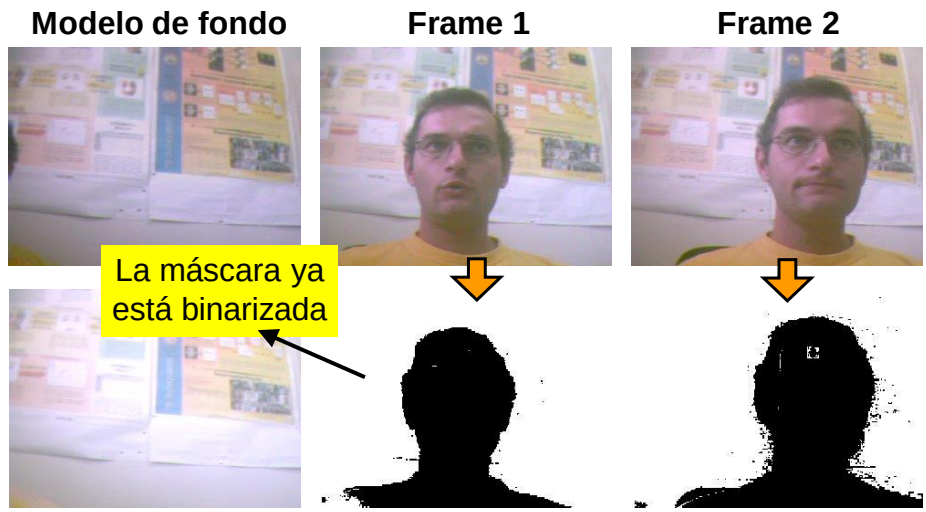
2.4. Combinación de imágenes.

- **Ejemplo.** Una alternativa para crear modelos de fondo es usar máximos y mínimos. En lugar de tener media y varianza, tenemos **máximo** y **mínimo** del **fondo** en cada píxel.

- Dada una imagen nueva, para cada píxel, comprobar si su valor está **entre el máximo y el mínimo**. Si lo está: fondo; si no lo está: objeto.

2.4. Combinación de imágenes.

- Con esto tenemos otra forma de hacer la segmentación de los objetos.



2.4. Combinación de imágenes.

Conclusiones:

- Operaciones de **combinación**: a partir de dos o más imágenes obtener una nueva imagen.
- La operación a aplicar depende de lo que queramos conseguir.
- Operaciones **booleanas**: útiles para trabajar con máscaras de objetos.
- Operaciones **aritméticas**: útiles en vídeo, modelos acumulados, detección de movimiento, transparencias difusas, etc.
- En general, cualquier tipo de operación es posible, ya sean lineales o no lineales.

2.5. Transformaciones de color.

- En los puntos anteriores la transformación era la misma para todos los canales (R, G y B).
- Si es **distinta**, hablamos de **transformación de color**:

  $R(x, y).R := f_1(A(x,y).R, A(x,y).G, A(x,y).B)$

  $R(x, y).G := f_2(A(x,y).R, A(x,y).G, A(x,y).B)$

  $R(x, y).B := f_3(A(x,y).R, A(x,y).G, A(x,y).B)$
- **Posibilidades:**
  - Aplicar **las mismas** transformaciones que antes (suma, producto, ajuste de histograma, etc.), pero con distintos parámetros para cada canal.
  - Transformaciones basadas en **modelos de color**. Cambiar el modelo de color (RGB, HSV, HLS, XYZ, YUV, etc.) y aplicar la función en ese modelo.

2.5. Transformaciones de color.

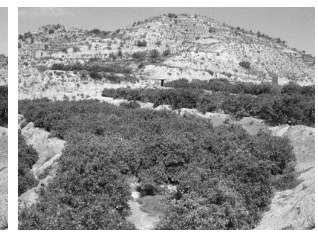
Conversión color → escala de grises

- **Conversión sencilla:**
  R(x, y):= (A(x,y).R + A(x,y).G + A(x,y).B)/3
- **Conversión precisa:**
  R(x, y):= 0.21A(x,y).R + 0.72A(x,y).G + 0.07A(x,y).B

Pero, ¿de dónde salen esos pesos?


Imagen de entrada

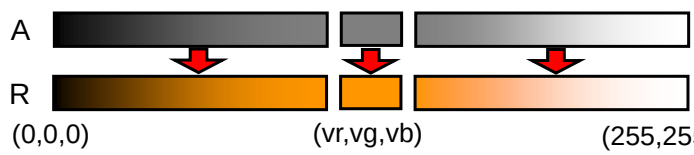

Grises (media)


Grises (precisa)

2.5. Transformaciones de color.

Transformación escala de grises → escala de color

- **Idea:** dada un imagen en gris, producir una imagen en escala de cierto color dado.
- Sea **A** una imagen en grises y un color objetivo **(vr, vg, vb)**. La escala se puede descomponer en dos partes:



- **Transformación** (obviamos (x,y)):
  si A<128 **entonces**
  R.R:= vr·A/128; R.G:= vg·A/128; R.B:= vb·A/128
  sino
  R.R:= vr+(255-vr)(A-128)/128; R.G:= vg+(255-vg)(A-128)/128
  R.B:= vb+(255-vb)(A-128)/128
  finsi

2.5. Transformaciones de color.

- **Ejemplo.** Transformación a sepia.

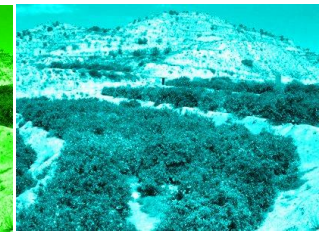

Imagen de entrada


Escala de grises


Escala de sepias

- ¿Cómo conseguir que el punto intermedio sea un valor cualquiera (distinto de 128)?

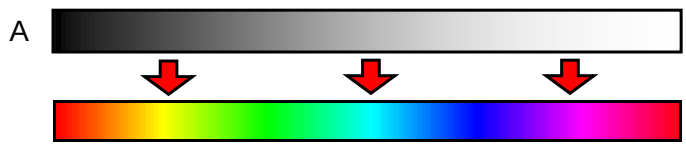

Escala de (30,255,0)


Escala de (0,255,255)

2.5. Transformaciones de color.

Transformación de color falso

- Es una transformación de la misma familia, cuyo objetivo es hacer más visibles las **pequeñas variaciones** del nivel de gris.
- Se define una paleta de salida adecuada y una transformación de cada valor de gris en la paleta.



R=	255	↓	0	0	↑	255
G=	↑	255	255	↓	0	0
B=	0	0	↑	255	255	↓

2.5. Transformaciones de color.

- **Ejemplo.** Transformación de color falso.
  Las transformaciones de este tipo son comunes en imágenes **médicas** y de **satélite**.

  En estas aplicaciones, la profundidad del canal puede ser fácilmente mayor que 1 byte. Al usar sólo 256 grises se pierde información.

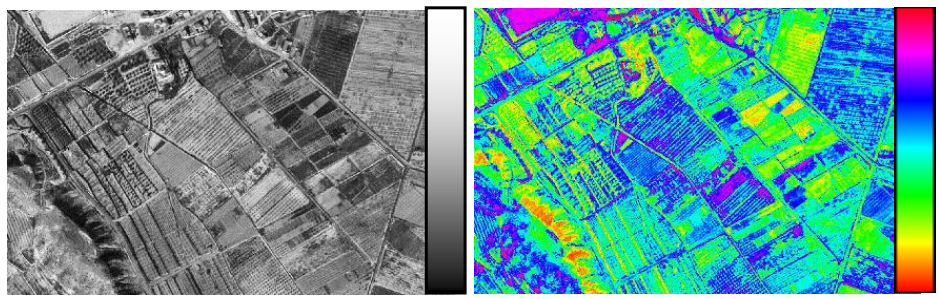


Imagen de entrada Imagen con color falso

2.5. Transformaciones de color.

Transformaciones de agregar color (colorear)

- **Idea:** usar las operaciones de suma, resta y producto, pero con una constante distinta por cada canal.

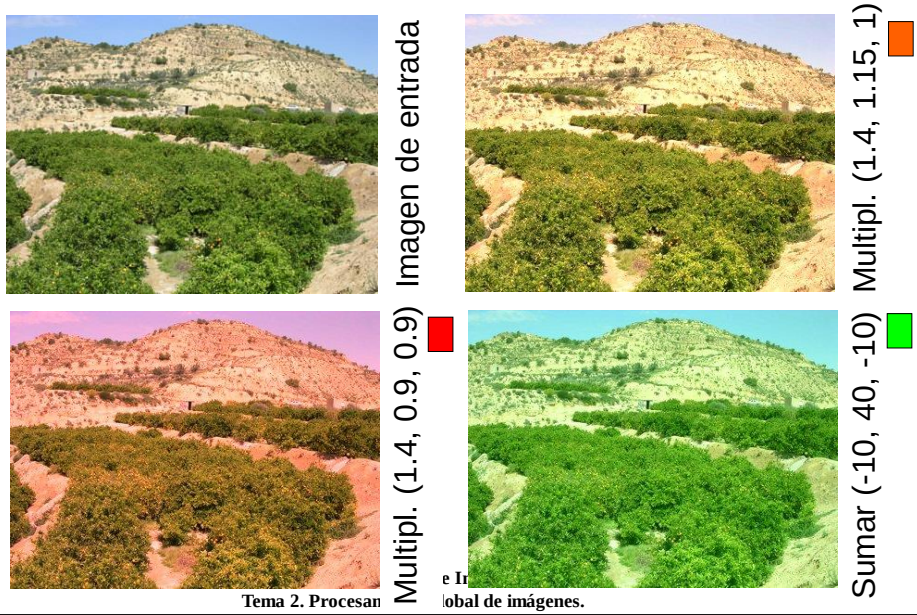
  R.R:= vr+A.R; R.G:= vg+A.G; R.B:= vb+A.B

  R.R:= fr·A.R; R.G:= fg·A.G; R.B:= fb·A.B

- **(vr, vg, vb)** y **(fr, fg, fb)** indican el tono de color que se da a la imagen.

2.5. Transformaciones de color.

Transformaciones de agregar color (colorear)



2.5. Transformaciones de color.

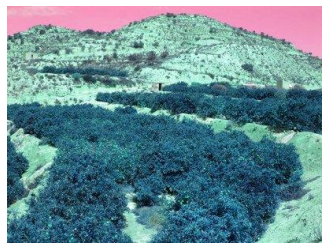
- Estas transformaciones están relacionadas con el **balance de blancos**.
- Las salidas de los fotodetectores de cada canal (R,G,B) deberían ser acordes a la apreciación subjetiva del color por parte del humano.
- Esto implica multiplicar cada canal por un factor adecuado.
- **Cuestión:** ¿qué imagen tiene los colores más realistas?
- **Ejemplos:**
  - Priorizar rojos (medio): fr= 1.2, fg= 0.9, fb= 0.9
  - Priorizar verdes (mucho): fr= 0.8, fg= 1.6, fb= 0.8
  - Priorizar amarillos (poco): fr= 1.1, fg= 1.1, fb= 0.8

2.5. Transformaciones de color.

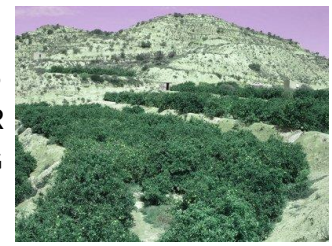
- También es posible **mezclar** y **cambiar** los canales, con transformaciones como las siguientes.

Imagen de entrada

R.R= A.G
R.G= A.B
R.B= A.R



R.R= A.B
R.G= A.R
R.B= A.G



R.R= (A.R+A.B)/2
R.G= (A.G+A.R)/2
R.B= (A.G+A.B)/2

2.5. Transformaciones de color.

- Finalmente, recordar que las operaciones de ajuste y ecualización del histograma se pueden aplicar **conjuntamente** (usando el histograma de gris) o por **separado** (usando el histograma de cada canal).
- La diferencia es que mientras el primero **mantiene** los colores (cambia la intensidad) el segundo no los mantiene.

Imagen de entrada

Ajuste conjunto

Ajuste separado

2.5. Transformaciones de color.

Conclusiones:

- Las transformaciones globales se pueden realizar **igual** en todos los canales o con valores **distintos**.
- En el primer caso, habrá un cambio en la **intensidad**. En el segundo, puede haber también un **cambio de color**.
- **Balance de blancos:** compensar los canales para obtener los colores más realistas posibles.
- Veremos más cuestiones relacionadas con el color cuando estudiemos **espacios de color**.

2. Procesamiento global de imágenes.

Conclusiones:

- **Procesamiento global:** el valor de un píxel de salida depende del píxel (o píxeles) correspondientes de la imagen de entrada.
- Operaciones aritméticas, lógicas, etc.
- **Distintas aplicaciones:** mejora del histograma, reducción de ruido, composición de imágenes, ajuste del color, etc.
- Normalmente no aparecen solas, sino combinadas con otros tipos de operaciones.

Anexo A.2.
Procesamiento global en OpenCV.

- Operaciones unarias
- Operaciones binarias
- Operaciones con histogramas
- Ejemplos

A.2. Procesamiento global en OpenCV.

Operaciones de procesamiento global:

- **Tipos**: unarias, binarias, histogramas.
- Las funciones reciben una (unarias) o dos (binarias) imágenes de entrada, y producen una de salida. Las de entrada **deben estar creadas** y normalmente deben tener el mismo tipo y tamaño.
- La mayoría de las funciones admite **modo inplace**: una imagen de entrada se puede usar también para guardar el resultado.
- Muchas operaciones trabajan con **ROI** y **mask** → No operan sobre toda la imagen sino solo sobre una **parte** concreta.
- Un **ROI** es una imagen que se crea como un fragmento (rectangular) de otra imagen.

A.2. Procesamiento global en OpenCV.

- Es importante observar las **restricciones** impuestas sobre las imágenes admitidas en cada función.
- En otro caso ocurrirá un error (se produce una **excepción**).
- **Normalmente**:
  - Todas las imágenes deben ser del **mismo tipo**: profundidad y número de canales.
  - Todas las imágenes deben tener el **mismo tamaño**. Si se utilizan imágenes ROI, el tamaño debe ser el mismo en todas.
  - Algunas funciones requieren imágenes con 1 solo canal.
  - Opcionalmente, algunas admiten el uso de una **máscara** (**mask**), que será otra imagen, de 8 bits y con un 1 canal. Un valor 0 significa que el píxel no se procesa, y ≠ 0 sí se procesa.
  - Ver la **documentación** para cada función.

A.2. Procesamiento global en OpenCV.

- Operaciones globales:
  - Asignación, copia y constantes: =, clone, copyTo, zeros, ones
  - Aritméticas: +, -, *, /, scaleAdd, add, subtract, addWeighted, multiply, divide
  - Valor absoluto: abs, absdiff
  - Conversión: convertTo, split, merge
  - Booleanas: bitwise_and, bitwise_or, bitwise_xor, bitwise_not
  - Comparación: ==, >, >=, <, <=
  - Otras: threshold, adaptiveThreshold, pow, log, exp
  - Comparación: min, max, compare, inRange
- Operaciones con **histogramas**:
  - calcHist, at, minMaxLoc, compareHist, normalize, equalizeHist, LUT

A.2. Procesamiento global en OpenCV.

- **Inicializar una imagen con un valor constante**: Se realiza directamente asignando (**=**) a la imagen un valor de tipo Scalar.
  - **Ejemplo**. Inicializar a verde:

    Mat imagen(480, 640, CV_8UC3);

    imagen= CV_RGB(0, 255, 0);
- **Asignación entre imágenes**: La asignación (**=**) de una imagen a otra, no implica una copia, sino que ambas comparten la misma memoria.
- **Copia de imágenes**: Para copiar una imagen, debe usarse **clone()**.
  - **Ejemplo**. Copia de una imagen:

    Mat imagen(480, 640, CV_8UC3);

    Mat copia= imagen.clone();
- **Imagen como ROI de otra**: una imagen puede ser una región de interés (ROI) de otra imagen. Esto es útil, por ejemplo, si queremos operar solo sobre un fragmento rectangular de otra imagen. Dos formas de crearla:

  Mat imagen(480, 640, CV_8UC3);

  Mat roi1(imagen, Rect(x, y, ancho, alto));

  Mat roi2= imagen(Rect(x, y, ancho, alto));

A.2. Procesamiento global en OpenCV.

- **Crear una imagen con un valor inicial**: En la definición de la variable:

  Mat imagen1(ancho, alto, tipo, CV_RGB(rojo, verde, azul));
- **Imágenes con valor inicial 0 o 1**: **Mat::zeros** y **Mat::ones**:

  Mat img1= Mat::zeros(ancho, alto, tipo);

  img1= Mat::ones(ancho, alto, tipo);
- **Copiar una imagen en otra**: copia el contenido (el valor de los píxeles):

  void imagen.**copyTo** (Mat destino);

  → Si la imagen de destino no está creada o su tamaño o tipo no es correcto, se crea automáticamente. En otro caso, se usa la imagen existente.
  - **Ejemplo**. Copiar un trozo de la imagen img en otra posición:

    Mat imagen= imread("imagen.jpg");

    Mat roi1= imagen(Rect(0, 0, 100, 100));

    Mat roi2= imagen(Rect(200, 200, 100, 100));

    roi1.copyTo(roi2);

  void imagen.**copyTo** (Mat destino, Mat máscara);

  → Copia en la imagen de destino solo aquellos píxeles que sean distintos de cero en máscara (debe ser de un canal).

A.2. Procesamiento global en OpenCV.

- **Sumar a una imagen un valor constante**: Suponiendo que tenemos: Mat img, img1, img2; Scalar escalar;

  img1 = img2 + escalar;

  img += escalar;

  → Suma con saturación píxel a píxel: img(x,y):= img(x,y) + escalar

  - **Ejemplo**. Sumar un poco de azul: img + Scalar(40,0,0);
  - Aumentar el brillo (inplace): img+= Scalar::all(50);
- **Restar a una imagen un valor constante**: Igual que antes pero con -:

  img1 = img2 - escalar; img1= escalar - img2;

  img -= escalar;

  - **Ejemplo**. Disminuir el brillo: img-= Scalar::all (50);
  - **Ejemplo**. Invertir el color de una imagen: img= Scalar::all(255)-img;
- **Sumar y restar dos imágenes**: Se puede hacer también con + y -:

  img = img1 + img2; img1+= img2;

  img = img1 - img2; img1-= img2;

A.2. Procesamiento global en OpenCV.

- **Multiplicar una imagen por una constante**: Siendo valor un número:

  img= img1*valor; img*= valor; img= img1/valor; img/= valor;

  - **Ejemplo**. Aumentar el contraste: img*= 1.5;
- **Media ponderada de dos imágenes**:

  img= img1*0.5 + img2*0.5;
- Las operaciones de suma y resta también se pueden hacer con notación funcional, con las operaciones **add** y **substract**. Estas funciones: (1) permiten usar máscaras; (2) permiten definir el tipo de datos de salida; y (3) img1 o img2 pueden ser un Scalar (de manera que el resultado sería sumar o restar un escalar a todos los píxeles de una imagen).
- **Suma de dos imágenes**:

  void **add** (Mat img1, Mat img2, Mat dest, [Mat máscara=NULO, int tipo=-1]);

  →si máscara(x,y)≠0 entonces dest(x,y):= img1(x,y) + img2(x,y)

  Tipo de la imagen dest
- **Resta de dos imágenes**:

  void **subtract** (Mat img1, Mat img2, Mat dest, Mat máscara, int tipo);

  →si máscara(x,y)≠0 entonces dest(x,y):= img1(x,y) - img2(x,y)

A.2. Procesamiento global en OpenCV.

- **Suma escalada de una imagen con otra**:

  void **scaleAdd** (Mat img1, double valor, Mat img2, Mat dest);

  → dest(x, y) := valor*img1(x,y) + img2(x,y)

- **Suma ponderada de dos imágenes**:

  void **addWeighted**(Mat A, double a, Mat B, double b, double c, Mat dest, [int tipo]);

  → dest:= a*A + b*B + c

  - Se puede hacer con la suma y el producto, pero de esta manera se evita la posible saturación en pasos intermedios. A o B pueden ser escalares.
  - **Ejemplo**: crear una transición entre dos imágenes A y B, que deben tener el mismo tamaño y número de canales.

```
Mat A= imread("a.jpg", 1);
Mat B= imread("b.jpg", 1);
Mat res;
for (int i= 0; i<100; i++) {
    addWeighted(A, i/100.0, B, 1-i/100.0, 0, res);
    imshow("Transicion", res);
    waitKey(10);
}
```


A.2. Procesamiento global en OpenCV.

- **Valor absoluto de diferencia entre dos imágenes**:

  void **absdiff** (Mat src1, Mat src2, Mat dest);

  → dest(x,y):= | src1(x,y) – src2(x,y) |

  - **Ejemplo**. "Buscar las 7 diferencias": absdiff(img1, img2, resultado);

- **Valor absoluto de una imagen:**

  Mat **abs** (Mat imagen);

  → resultado(x,y):= | imagen(x,y) |

  - Tiene sentido cuando la profundidad son números con signo.

- **Convertir la profundidad de una imagen**:

  void Mat::**convertTo**(Mat dest, int tipo, [double prod=1, double suma=0]);

  → dest(x,y):= M(x,y) * prod + suma (igual para todos los canales)

  - Se puede hacer cualquier transformación lineal (suma/producto).
  - Permite convertir a cualquier profundidad, con igual número de canales.

- **Producto, suma y conversión del tipo de una imagen**:

  void **convertScaleAbs**(Mat src, Mat dest, [double prod=1, double suma=0]);

  → dest(x,y):= | src(x,y) * prod + suma | (igual para todos los canales)

  - Se puede hacer cualquier transformación lineal.
  - La función permite cualquier profundidad de entrada y convierte a 8U.

A.2. Procesamiento global en OpenCV.

- **Y lógico entre una imagen y otra (o una constante) a nivel de bits**:
  void **bitwise_and** (Mat src1, Mat src2, Mat dest, [Mat máscara = NULO]);
  → si máscara(x,y)≠0 entonces dest(x,y):= src1(x,y) AND src2(x,y)
- **O lógico entre una imagen y otra (o una constante) a nivel de bits**:
  void **bitwise_or** (Mat src1, Mat src2, Mat dest, [Mat máscara = NULO]);
  → si máscara(x,y)≠0 entonces dest(x,y):= src1(x,y) OR src2(x,y)
- **O exclusivo entre una imagen y otra (o una constante) a nivel de bit**:
  void **bitwise_xor** (Mat src1, Mat src2, Mat dest, [Mat máscara = NULO]);
  → si máscara(x,y)≠0 entonces dest(x,y):= src1(x,y) XOR src2(x,y)
- **Negación lógica de una imagen, a nivel de bits**:
  void **bitwise_not** (Mat src, Mat dest, [Mat máscara = NULO]);
  → si máscara(x,y)≠0 entonces dest(x,y):= NOT src(x,y)
  - **Ejemplo**. Vale para invertir una imagen: bitwise_not(img, img);

A.2. Procesamiento global en OpenCV.

- **Comparación entre una imagen y un valor constante**:
  void **inRange** (Mat src, Mat lowerb, Mat upperb, Mat dst);
  → dest(x,y):= (lowerb(x,y) <= src(x,y) <= upperb(x,y))
  - Comprueba, para cada píxel de la imagen, si su valor está entre el límite inferior (lowerb) y el superior (upperb), que pueden ser imágenes o constates (esclares).
  - El resultado es una imagen binaria, 8U, con 0 ó 255 (todos los bits a 1).
  - Si src es multicanal, comprueba todos los canales están en el rango.
- **Umbralización/binarización de una imagen**:
  double **threshold** (Mat src, Mat dst, double thresh, double maxval, int tipo);
  - Umbraliza la imagen según el método dado en **tipo** y el umbral es **thresh**.
  - P.ej., THRESH_BINARY para binarizar con umbral 128:
    threshold(A, C, 128, 255, THRESH_BINARY);
    → C(x,y):= si A(x,y) > 128 entonces 255 si no 0
  - La umbralización se hace con un valor constante. Para un método más avanzado ver **adaptiveThreshold**. El umbral se calcula para cada píxel, usando una vecindad local (adaptativo).

A.2. Procesamiento global en OpenCV.

- **Máximo entre dos imágenes o entre imagen y un valor constante**:

  void **max** (Mat src1, Mat src2, Mat dest);

  →dest(x,y):= max {src1(x,y), src(x,y)}

  - En imágenes multicanal, calcula el máximo para cada uno de los canales.
- **Mínimo entre dos imágenes o entre imagen y un valor constante**:

  void **min** (Mat src1, Mat src2, Mat dest);

  →dest(x,y):= min {src1(x,y), src(x,y)}
- **Potencia, exponencial y logaritmo de los píxeles de una imagen**:

  void **pow** (Mat A, double p, Mat C); → $C(x,y):= A(x,y)^p$

  void **exp** (Mat A, Mat C); → $C(x,y):= e^{A(x,y)}$

  void **log** (Mat A, Mat C); → $C(x,y):= \log_e |A(x,y)|$

  - Para evitar saturación y pérdida de información, es conveniente transformar las profundidad a reales de 32 o 64 bits.
  - **Ejemplo**. Transformación de gamma de una imagen img:

```
Mat im32F;
img.convertTo(im32F, CV_32F, 1.0/255.0);
pow(im32F, ui->doubleSpinBox->value(), im32F);
im32F.convertTo(img, CV_8U, 255);
```


A.2. Procesamiento global en OpenCV.

- **Multiplicar dos imágenes**:

  void **multiply** (Mat src1, Mat src2, Mat dest, [double scale=1, int tipo=-1]);

  → dest(x,y):= src1(x,y)*src2(x,y)*scale

  - El valor **scale** permite evitar problemas de saturación.
  - El valor **tipo** permite cambiar el tipo de datos de salida.
  - **src1** y **src2** pueden ser escalares. Y se puede usar modo in-place.
  - **Ejemplo**: multiplicar dos imágenes de 8 bits:
    multiply(im1, im2, resultado, 1./255);
- **Dividir una imagen por otra**:

  void **divide** (Mat src1, Mat src2, Mat dest, [double scale=1, int tipo=-1]);

  → dest(x,y):= scale*src1(x,y)/src2(x,y)

  - Las mismas opciones y parámetros que en la anterior función.
  - Tener cuidado con los problemas de saturación. Puede ser adecuado usar enteros con signo.
  - **Ejemplo**:
    divide(im1, im2, resultado, 255.0);

A.2. Procesamiento global en OpenCV.

- Observar el **estilo de programación** usando estas funciones.
- Por ejemplo, queremos implementar la operación:
  R(x,y):= A(x,y)·(255-N(x,y))/255 + B(x,y)·N(x,y)/255
  void **Combina** (Mat A, Mat B, Mat N, Mat &R);
- **Implementación 1**.

```
multiply(B, N, B, 1./255);
bitwise_not(N, N);
multiply(A, N, A, 1./255);
R= A+B;
```

Esto es más **sencillo** y **eficiente**, porque las operaciones están **optimizadas**

Aunque no del todo correcto, porque modifica las imágenes de entrada A, B y N...

- **Implementación 2**.

```
Vec3b pA, pB, pN, pR;
R.create(A.size(), A.type());
for (int y=0; y<A.size().height; y++)
  for (int x= 0; x<A.size().width; x++) {
    pA= A.at<Vec3b>(y, x);
    pB= B.at<Vec3b>(y, x);
    pN= N.at<Vec3b>(y, x);
    for (int c= 0; c<3; c++)
      pR[c]= (pA[c]*(255-pN[c])+pB[c]*pN[c])/255.0;
    R.at<Vec3b>(y, x)= pR;
  }
```

Esto es **menos eficiente** (~2 veces más lento) y menos recomendable

A.2. Procesamiento global en OpenCV.

Operaciones con histogramas

- En OpenCV los histogramas son de tipo **Mat** siendo la profundidad **float** y de 1 canal. Por lo tanto, podemos usar las operaciones lineales (suma, resta, producto, etc.) y el acceso y modificación con **at**.
- El número de dimensiones del histograma depende de los que queramos calcular (hist. de 1 canal, de 2 canales, etc.).
- El tamaño del histograma (**bins**) también puede cambiar, según la resolución que queramos (desde 2 hasta 256).
- Tenemos también una operación de ecualización del histograma: **equalizeHist**.
- Otra cuestión relacionada son las **tablas de transformación** (*look-up table*, o *LUT*), para realizar una transformación de curva tonal arbitraria.

A.2. Procesamiento global en OpenCV.

- **Propiedades** de un histograma:
  - **Número de dimensiones (dims)**. Normalmente tendremos 1 dimensión (escala de grises), 2 dimensiones (histogramas conjuntos de dos canales) o como mucho 3 (de triplas R,B,G).
  - Para cada dimensión, **número de celdas** (**histSize**). Normalmente será una potencia de 2, como 256, 64, 32...
  - **Rango de valores** correspondientes a cada celda (**ranges**), (normalmente el hist. es uniforme, y los valores son informes).
- **Ejemplos.**

Histograma de 1 dimensión y 4 celdas

Bin 0 (0-63)	Bin 1 (64-127)	Bin 2 (128-191)	Bin 3 (192-255)

Histograma de 2 dimensiones, con tamaños 3 y 2

	Bin 0 (0-127)	Bin 1 (128-255)
Bin 0 (0-85)		
Bin 1 (86-170)		
Bin 2 (171-255)		

A.2. Procesamiento global en OpenCV.

- **Calcular el histograma de una imagen**:

void **calcHist** (Mat* images, int nimages, const int* canales, Mat mask, Mat hist, int dims, int* histSize, float** ranges, [bool uniform= true, bool acumular= false]);

  - **images**: array de imágenes sobre las que se calcula el histograma. Normalmente será una sola imagen de 1 o 3 canales. Si hay varias, todas ellas deben tener el mismo tamaño.
  - **nimages**: número de imágenes que hay en el anterior array.
  - **canales**: array con los números de los canales sobre los que se quiere calcular el histograma. Los canales están numerados 0, 1, …
  - **mask**: máscara opcional. Si vale noArray(), no se usa. En otro caso, debe ser una imagen de 1 canal y profundidad 8U.
  - **hist**: histograma resultante, de profundidad float.
  - **dims**: número de dimensiones del histograma. Debe coincidir con el tamaño del array canales (por ejemplo, si **dims**=3, deben indicarse 3 canales).
  - **histSize**: tamaño (número de bins) en cada una de las dimensiones.
  - **ranges**: rango para cada dimensión. Normalmente (con histograma uniforme) será un array de arrays con valores {0, 256}.
  - **uniform**: indica si las celdas se distribuyen el rango uniformemt. (es lo normal).
  - **acumular**: permite acumular los valores entre distintas llamadas a la función.

A.2. Procesamiento global en OpenCV.

- **Ejemplo 1.** Calcular el histograma unidimensional del nivel de gris de una imagen “a.jpg” en color. El resultado se escribe en salida debug:

```
Mat img= imread("a.jpg", 1);
Mat gris;
Mat hist;
cvtColor(img, gris, CV_BGR2GRAY);  // Conversión a gris
int canales[1]= {0};
int bins[1]= {256};
float rango[2]= {0, 256};
const float *rangos[]= {rango};
calcHist(&gris, 1, canales, noArray(), hist, 1, bins, rangos);
for (int i= 0; i<256; i++)
    qDebug("Celda %d: %g", i, hist.at<float>(i));
```

- Observar el acceso a las celdas del histograma con:

  hist.at<float>(posición)

A.2. Procesamiento global en OpenCV.

- **Ejemplo 2.** Calcular el histograma bidimensional de los canales (R,G) de una imagen “a.jpg” en color, con 64x64 celdas. El resultado se pinta en una imagen:

```
Mat img= imread("a.jpg", 1);
Mat hist;
int canales[2]= {2, 1};  // El 2 es el canal R, y el 1 es el canal G
int bins[2]= {64, 64};
float rango[2]= {0, 256};
const float *rangos[]= {rango, rango};
calcHist(&img, 1, canales, noArray(), hist, 2, bins, rangos);

// Operaciones para pintar el histograma
Mat pinta(64, 64, CV_8UC1);
double minval, maxval;
minMaxLoc(hist, &minval, &maxval); // Para escalar el color entre blanco y negro
for (int r= 0; r<64; r++)
   for (int g= 0; g<64; g++)
      pinta.at<uchar>(r, g)= 255-255*hist.at<float>(r, g)/maxval;
namedWindow("Histograma R-G", 0);
imshow("Histograma R-G", pinta);
```

A.2. Procesamiento global en OpenCV.

- **Normalizar un histograma**: conseguir que la suma de todas las celdas sea un valor dado (por ejemplo, que sumen 100).
  - Si se conoce el número de píxeles de la imagen, se puede hacer con una simple ponderación: hist*= 100.0/**img.size().area()**;
  - Si no se conoce, se puede calcular la suma total de las celdas: hist*= 100.0/**norm**(hist, NORM_L1);
- **Obtener máximo y mínimo de un histograma**:

  void **minMaxLoc** (Mat h, double* minVal, double* maxVal=0, Point* minLoc=0, Point* maxLoc=0, Mat mask=noArray());
  - Dado el histograma, calcula el mínimo (**minVal**), el máximo (**maxVal**), el índice de la celda mínima (**minLoc**) y máxima (**maxLoc**).
  - Ver un uso en el ejemplo anterior.
  - También puede aplicarse esta función sobre imágenes, para buscar el píxel más claro y/o más oscuro.
- **Ecualizar el histograma de una imagen**:

  void **equalizeHist** (Mat src, Mat dst);
  - La imagen debe ser de 1 solo canal y 8U.
- **Otras funciones interesantes**: **compareHist** (comparar histogramas), **normalize** (normalizar los niveles de gris de una imagen, usando diferentes criterios).

A.2. Procesamiento global en OpenCV.

- Las **tablas de transformación** (*look-up table*, *LUT*) son tablas que definen funciones discretas de la forma:

  f: [0...255] → R
- Esto nos permite construir cualquier **curva tonal** arbitraria.
- En OpenCV, una LUT es una **matriz** de tipo **Mat**:

  Mat lut(1, 256, CV_8UC1);
  - 1 fila y 256 columnas. El número de canales puede ser 1 o 3.
  - La profundidad puede cambiar (8U, 8S, 16U, 32F, ...)
- **Aplicar una transformación de tabla LUT**:

  void **LUT** (Mat src, Mat lut, Mat dest);
  - La profundidad de entrada, **src**, debe ser 8 bits (con o sin signo), y la de salida (en **dest** y en **lut**) puede ser cualquiera.
  - Si la imagen **src** tiene 1 canal, **lut** debe ser de 1 canal. Si **src** tiene 3 canales, **lut** puede ser de 1 canal (aplicar la misma transformación a todos los canales) o de 3 canales (como si fueran 3 tablas LUT, una para cada canal).

A.2. Procesamiento global en OpenCV.

- **Ejemplo 1**. Aplicar una ecualización conjunta del histograma a una imagen “a.jpg” en color, usando **calcHist** y **LUT**:

```
Mat img= imread(“a.jpg", 1);
imshow("Entrada", img);
Mat gris, hist;
cvtColor(img, gris, CV_BGR2GRAY);
int canales[1]= {0}, bins[1]= {256};
float rango[2]= {0, 256};
const float *rangos[]= {rango};
calcHist(&gris, 1, canales, noArray(), hist, 1, bins, rangos);
hist*= 255.0/norm(hist, NORM_L1);
Mat lut(1, 256, CV_8UC1);
float acum= 0.0;
for (int i= 0; i<256; i++) {
   lut.at<uchar>(0, i)= acum;
   acum+= hist.at<float>(i);
}
Mat res;
LUT(img, lut, res);
imshow("Ecualizada", res);
```