Filtros, Convoluciones y Transformaciones Básicas

Antonio Falcó y Juan Pardo

Introducción a la Inteligencia Artificial

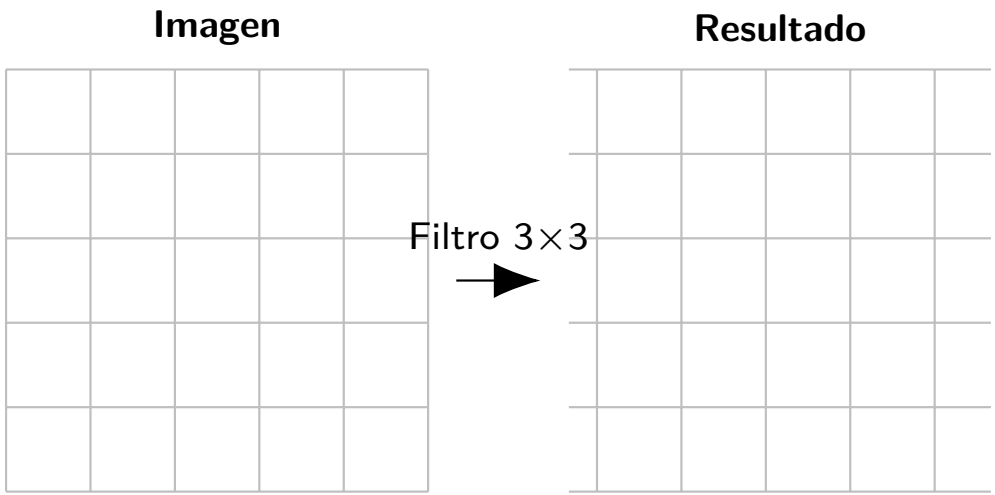
Lectura 2

Objetivos de la sesión

- Comprender qué es un filtro y cómo actúa sobre una imagen.
- Introducir la operación de convolución como base del procesado digital.
- Explorar ejemplos de filtros: media, gaussiano y de bordes (Sobel).
- Aplicar transformaciones geométricas: traslación, rotación, escalado.
- Conectar estas técnicas con el preprocesado para modelos de IA.

Que es un filtro?

- Un filtro transforma una imagen modificando los valores de sus píxeles.
- Se define mediante una pequeña matriz de pesos (**kernel**) que se aplica localmente.
- Los filtros permiten:
  - Suavizar (reducir ruido)
  - Resaltar bordes o texturas
  - Detectar formas



La operación de convolución

Definición discreta de convolución 2D

Sea $f(x, y)$ una imagen discreta y $k(i, j)$ un kernel (o máscara) de tamaño $(2a + 1) \times (2b + 1)$. La convolución discreta de k con f se define como:

$$(k * f)(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x - i,\, y - j)\, k(i, j)$$

y produce una nueva imagen $g(x, y) = (k * f)(x, y)$.

- Cada píxel se reemplaza por una combinación ponderada de sus vecinos.
- El resultado depende de la forma del kernel.

Ejemplo intuitivo

Un filtro de media 3×3 reemplaza cada píxel por la media de su vecindario inmediato.

Nota sobre correlación

En muchas librerías (p.ej., OpenCV, PyTorch) se implementa la **correlación cruzada**:

$$(k \star f)(x, y) \;=\; \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x+i,\, y+j)\, k(i,j),$$

que no invierte el kernel. La diferencia respecto a la convolución es una reflexión espacial del filtro.

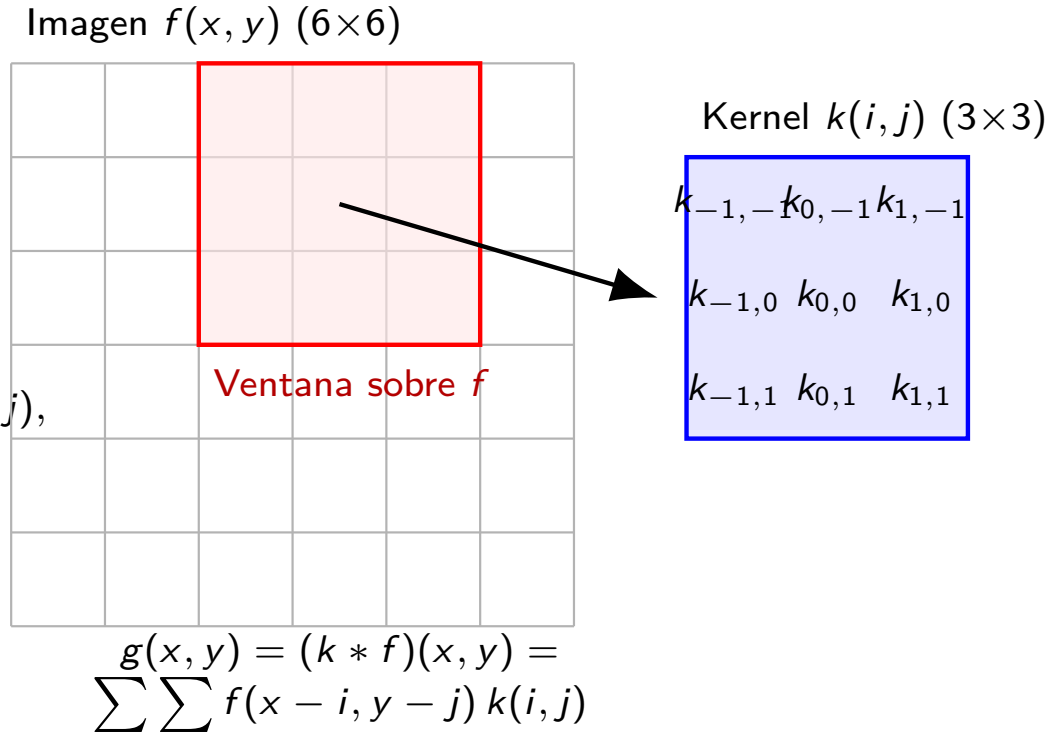
Convolución discreta 2D: definición y esquema

Definición (discreta, 2D)

Sea $f(x, y)$ una imagen discreta y $k(i, j)$ un kernel (máscara) de tamaño $(2a + 1) \times (2b + 1)$. La convolución de k con f se define como:

$$(k*f)(x, y) \;=\; \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x-i,\; y-j)\, k(i, j),$$

y produce una nueva imagen $g(x, y) = (k * f)(x, y)$.



Bordes (padding)

Común: *zero-padding*, *replicate*, *reflect*. El padding determina el tamaño de la salida y evita pérdida en los bordes.

Ejemplo numérico: convolución con filtro de media 3×3

Parche de imagen (3×3):

$$\mathbf{P} = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

Ventana 3×3

10	20	30
40	**50**	60
70	80	90

Kernel de media:

$$\mathbf{K} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Cálculo del píxel central:

$$(\mathbf{K} * \mathbf{P})_{\text{centro}} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} P_{(1+i,\,1+j)}\, K_{(i,j)}.$$

$$\sum P_{(u,v)} = 10+20+30+40+50+60+70+80+90 = 450.$$

$$(\mathbf{K} * \mathbf{P})_{\text{centro}} = \frac{450}{9} = \boxed{50}.$$

Interpretación del filtro de media y extensión general

Resultado del ejemplo

El valor de salida en el centro es:

$$(\mathbf{K} * \mathbf{P})_{\text{centro}} = 50$$

Es decir, el nuevo píxel toma el **promedio** de los nueve valores vecinos:

$$50 = \frac{1}{9}(10 + 20 + \cdots + 90)$$

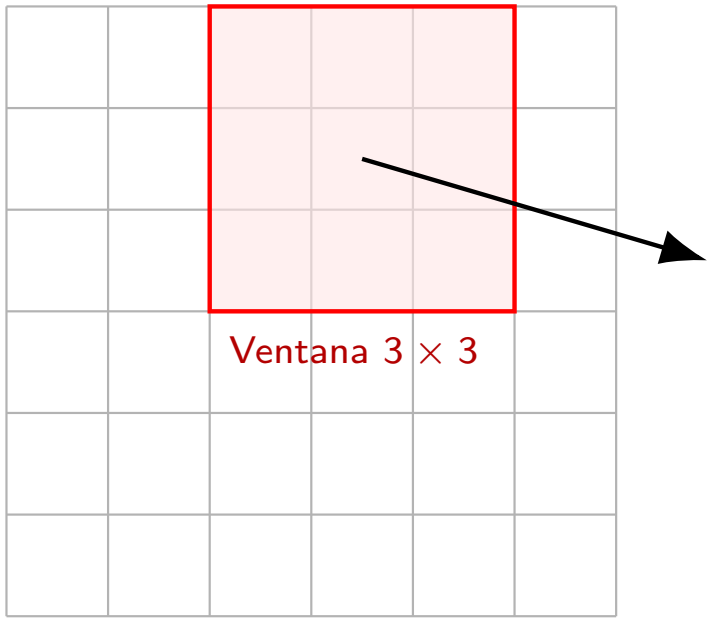
Efecto visual

- El filtro de media suaviza la imagen, reduciendo ruido.
- Sin embargo, también difumina bordes y detalles finos.
- Es un caso particular de filtro *de paso bajo*.

Interpretación del filtro de media y extensión general

En una imagen completa:

- La ventana 3×3 se desliza por toda la imagen.
- Cada posición genera un nuevo valor promedio.
- Los bordes requieren un tratamiento especial (*padding*).



Conclusión

Los filtros de convolución permiten modificar las imágenes según la forma del kernel: suavizado, detección de bordes o realce de contraste.

Promedio local (3×3) en un 6×6 con datos realistas

Imagen original f (parche realista): Salida $g = \text{media}_{3\times3}(f)$ (padding *reflect*):

$f(x, y)$

94	103	114	122	137	149
94	105	118	129	139	156
100	113	122	158	170	155
104	112	124	163	178	162
104	117	131	142	152	166
111	126	132	145	156	165

$$g(x, y) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(x - i, y - j)$$

97	**104**	**114**	**126**	**138**	**147**
100	**107**	**120**	**134**	**146**	**152**
103	**110**	**127**	**145**	**157**	**159**
106	**114**	**131**	**149**	**161**	**163**
110	**118**	**132**	**147**	**159**	**164**
113	**121**	**133**	**143**	**155**	**162**

Lectura del efecto

La media 3×3 **suaviza** valores locales y **reduce el ruido**. La zona brillante (col. 4–5, filas 3–4) se difunde a sus vecinos: el pico se atenúa y el entorno se eleva ligeramente.

Promedio local (3×3) con zero-padding: diferencias visibles

Imagen original f (impulso en el centro):

$f(x, y)$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	100	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Salida $g = \text{media}_{3\times3}(f)$ (zero-padding):

$$g(x, y) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(x - i, y - j) \text{ (media 3×3)}$$

0	0	0	0	0	0
0	0	**11.1**	**11.1**	**11.1**	0
0	0	**11.1**	**11.1**	**11.1**	0
0	0	**11.1**	**11.1**	**11.1**	0
0	0	0	0	0	0
0	0	0	0	0	0

Por qué aquí sí hay diferencias

Con **zero-padding**, los vecinos fuera de la imagen cuentan como 0. Un impulso aislado (100) se *difunde* en un parche 3×3 con valor medio $100/9 \approx 11{,}1$, mostrando claramente el efecto de **suavizado**.

Filtro gaussiano: fundamentos y parámetros

- **Definición (2D continuo)**:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\,\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- **Kernel discreto**: muestrea G en una rejilla de tamaño impar $k \times k$.
- **Regla práctica de tamaño**: $k \approx 2\lceil 3\sigma \rceil + 1$ (*cubre* $\pm 3\sigma$).
- **Normalización**: reescala para que $\sum K = 1$ (conserva brillo medio).
- **Efecto**: reduce altas frecuencias (ruido), preserva estructuras grandes;
  bordes se atenúan más cuanto mayor es σ.
- **Bordes (padding)**: `zero`, `edge` (replicar), `reflect` (recomendado para evitar halos).

Generación de kernel (idea)

$$K[i,j] \propto \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right), \quad i,j \in \{-r,\ldots,r\},\ r = \tfrac{k-1}{2}; \quad K \leftarrow \frac{K}{\sum K}$$

Filtro gaussiano: separabilidad e implementación

- **Separabilidad**: $G(x, y) = g(x)\, g(y)$. Convolucionar en 1D *horizontal* + *vertical*.
  Coste: de $\mathcal{O}(k^2)$ a $\mathcal{O}(2k)$ por píxel.
- **Variante LoG/DoG**:
  - LoG: $(\nabla^2 G) * f$ (bordes tras suavizado).
  - DoG: $G_{\sigma_1} * f - G_{\sigma_2} * f \approx$ LoG si $\sigma_2 > \sigma_1$ (más rápido).

Filtro gaussiano: separabilidad e implementación

```
import numpy as np

# 1) Kernel 1D (separable)
def gaussian1d(size=5, sigma=1.0):
    r = size//2
    ax = np.arange(-r, r+1, dtype=np.float32)
    k = np.exp(-(ax**2)/(2*sigma**2))
    k /= k.sum()
    return k  # shape: (size,)
```

Reglas rápidas

• Elige $k \approx 6\sigma + 1$; • usa `reflect` para bordes; • antes de Sobel, aplica gaussiano pequeño para estabilizar bordes.

Suavizado gaussiano (3×3) en un parche 6×6

Kernel gaussiano (normalizado):

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \Rightarrow \qquad (g)(x,y) = (K * f)(x,y)$$

Imagen original f (valores realistas):

$f(x,y)$

94	103	114	122	137	149
94	105	118	129	139	156
100	113	122	158	170	155
104	112	124	163	178	162
104	117	131	142	152	166
111	126	132	145	156	165

Salida $g = K * f$ (padding *reflect*, redondeo):

$$g(x,y) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} K(i,j)\, f(x-i, y-j)$$

99	**104**	**115**	**126**	**138**	**145**
101	**107**	**119**	**133**	**145**	**150**
105	**111**	**126**	**147**	**159**	**161**
108	**114**	**130**	**152**	**164**	**165**
112	**118**	**131**	**146**	**158**	**162**
114	**120**	**132**	**143**	**154**	**160**

Lectura

El gaussiano (paso bajo) **suaviza** preservando mejor estructuras que la media uniforme: picos se atenúan con menos artefactos y el ruido fino se reduce de forma más natural.

Cálculo paso a paso: píxel central con filtro gaussiano 3×3

Kernel gaussiano: $K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. **Vecindario (centrado en $(3,3)$):**

$$\begin{bmatrix} 105 & 118 & 129 \\ 113 & 122 & 158 \\ 112 & 124 & 163 \end{bmatrix}$$

Producto elemento a elemento y suma:

$$\begin{aligned} \text{Num} &= 1 \cdot 105 + 2 \cdot 118 + 1 \cdot 129 \\ &\quad + 2 \cdot 113 + 4 \cdot 122 + 2 \cdot 158 \\ &\quad + 1 \cdot 112 + 2 \cdot 124 + 1 \cdot 163 \\ &= 105 + 236 + 129 + 226 + 488 + 316 + 112 + 248 + 163 \\ &= 2023. \end{aligned}$$

$$g(3,3) = (K * f)(3,3) = \frac{\text{Num}}{16} = \frac{2023}{16} \approx \boxed{126{,}44}$$ (p.ej., 126 si redondeamos).

Interpretación

El gaussiano pondera más el centro y menos los vecinos. Resultado: **suaviza** preservando mejor las estructuras que la media uniforme.

Vecindario 3×3

105	118	129
113	**122**	158
112	124	163

Pesos K

$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$
$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$

Fundamentos de la detección de bordes (filtro de Sobel)

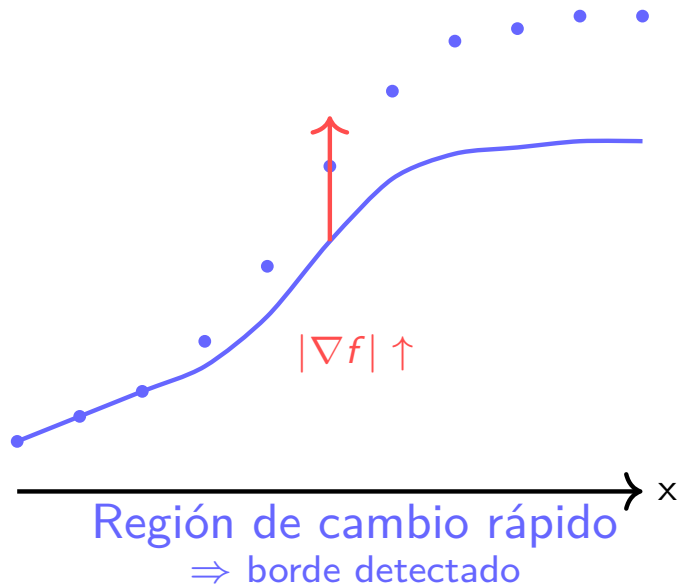
Idea básica: Los **bordes** son zonas donde la intensidad de la imagen cambia bruscamente. En términos matemáticos, corresponden a regiones con **gradiente alto** de la intensidad. El gradiente mide el cambio espacial de intensidad:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right), \qquad |\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}.$$

Aproximación discreta (Sobel):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

- G_x: detecta bordes verticales (cambios horizontales).
- G_y: detecta bordes horizontales (cambios verticales).
- Ambos se aplican por convolución: $f_x = G_x * f, \quad f_y = G_y * f.$



Magnitud del borde

Después de aplicar G_x y G_y, se obtiene:

$$|\nabla f| = \sqrt{(G_x * f)^2 + (G_y * f)^2}.$$

Los píxeles con valores altos de $|\nabla f|$ se marcan como **bordes**.

Sobel 3×3: ejemplo numérico paso a paso

Parche centrado (3×3): $\mathbf{P} = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 120 & 220 \\ 10 & 120 & 220 \end{bmatrix}$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Derivada horizontal: $f_x = (G_x * P)$

$$\begin{aligned} f_x &= (-1)\cdot 10 + 0\cdot 10 + 1\cdot 10 \\ &\quad + (-2)\cdot 10 + 0\cdot 120 + 2\cdot 220 \\ &\quad + (-1)\cdot 10 + 0\cdot 120 + 1\cdot 220 \\ &= (-10 + 0 + 10) + (-20 + 0 + 440) \\ &\quad + (-10 + 0 + 220) \\ &= 0 + 420 + 210 \;=\; \boxed{630}. \end{aligned}$$

Derivada vertical: $f_y = (G_y * P)$

$$\begin{aligned} f_y &= (-1)\cdot 10 + (-2)\cdot 10 + (-1)\cdot 10 \\ &\quad + 0\cdot 10 + 0\cdot 120 + 0\cdot 220 \\ &\quad + 1\cdot 10 + 2\cdot 120 + 1\cdot 220 \\ &= (-10 - 20 - 10) + 0 + (10 + 240 + 220) \\ &= -40 + 0 + 470 \;=\; \boxed{430}. \end{aligned}$$

Magnitud y orientación del borde

$$|\nabla f| \;=\; \sqrt{f_x^2 + f_y^2} \;\approx\; \sqrt{630^2 + 430^2} \;\approx\; \boxed{7{,}63 \times 10^2} \;(\approx 762\text{–}763),$$

$$\theta \;=\; \operatorname{atan2}(f_y, f_x) \;=\; \operatorname{atan2}(430, 630) \;\approx\; \boxed{34^\circ}.$$

Lectura: borde con dirección θ (normal al borde) e intensidad proporcional a $|\nabla f|$.

Fundamentos de las transformaciones geométricas

Idea básica: Una transformación geométrica modifica la posición de los píxeles de una imagen manteniendo sus valores de intensidad. Matemáticamente, consiste en un cambio de coordenadas:

$$(x', y') = T(x, y),$$

donde T puede ser una traslación, rotación, escalado o una combinación lineal (afin, proyectiva, etc.).

Tipos de transformaciones

- **Traslación:** desplaza la imagen en el plano.

$$T(x, y) = (x + t_x,\ y + t_y)$$

- **Escalado:** amplía o reduce el tamaño.

$$T(x, y) = (s_x x,\ s_y y)$$

- **Rotación (ángulo θ):**

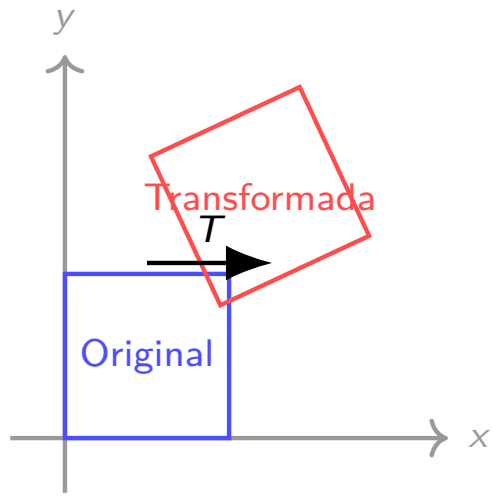
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- **Reflexión / Cizalladura:** deforman la geometría preservando líneas rectas.

Fundamentos de las transformaciones geométricas

Aspectos prácticos

- Cada transformación requiere una **interpolación**: ¿cómo asignar intensidades a las nuevas posiciones?
- El mapeo suele calcularse al *revés* (de destino a origen).
- Ejemplo: al rotar 30°, el píxel de salida (x', y') proviene de una posición no entera en la imagen original.



Intuición

Una transformación geométrica **reubica los píxeles** según una función T, mientras que las *convoluciones* modifican sus **valores**.

Transformaciones afines en imágenes

Idea: Las transformaciones afines combinan operaciones lineales (rotación, escala, cizalladura) y traslación en una única formulación matricial.

Expresión matricial

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{Matriz afín } A} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\Rightarrow \quad \begin{cases} x' = a_{11}x + a_{12}y + t_x, \\ y' = a_{21}x + a_{22}y + t_y. \end{cases}$$

- a_{11}, a_{22} controlan la **escala** y rotación.
- a_{12}, a_{21} controlan la **cizalladura** (shear).
- t_x, t_y definen la **traslación**.

Transformaciones afines en imágenes

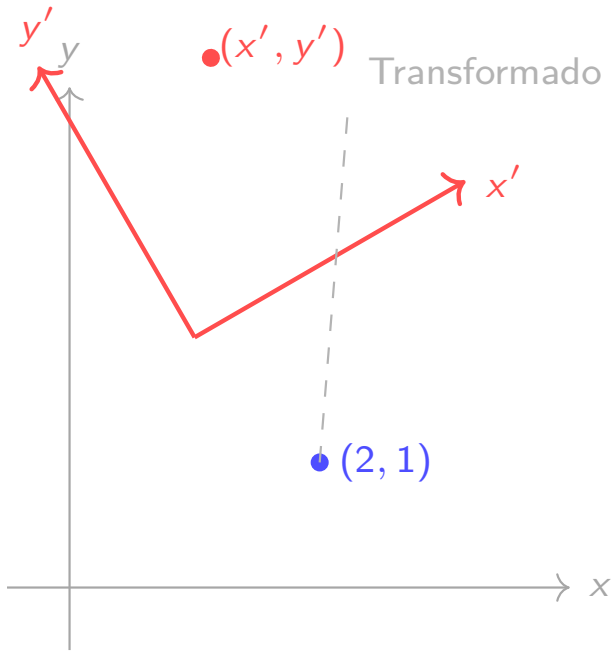
Ejemplo numérico:

$$A = \begin{pmatrix} \cos 30^\circ & -\sin 30^\circ & 1 \\ \sin 30^\circ & \cos 30^\circ & 2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0{,}866 & -0{,}5 & 1 \\ 0{,}5 & 0{,}866 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Aplicando a un punto $(x, y) = (2, 1)$:

$$x' = 0{,}866 \cdot 2 - 0{,}5 \cdot 1 + 1 = 2{,}23,$$
$$y' = 0{,}5 \cdot 2 + 0{,}866 \cdot 1 + 2 = 3{,}87.$$

$$\Rightarrow (x', y') = (2{,}23,\ 3{,}87)$$



Resumen

Una transformación afín puede representarse con una matriz 2×3.
Permite combinar rotación, escala, cizalladura y traslación en una sola operación lineal.

Composición de transformaciones geométricas

Idea clave: Varias transformaciones geométricas pueden combinarse mediante la **multiplicación de sus matrices homogéneas**. El orden de aplicación **sí importa**.

Formulación general

Si $T_1, T_2, \ldots, T_k$ son transformaciones afines:

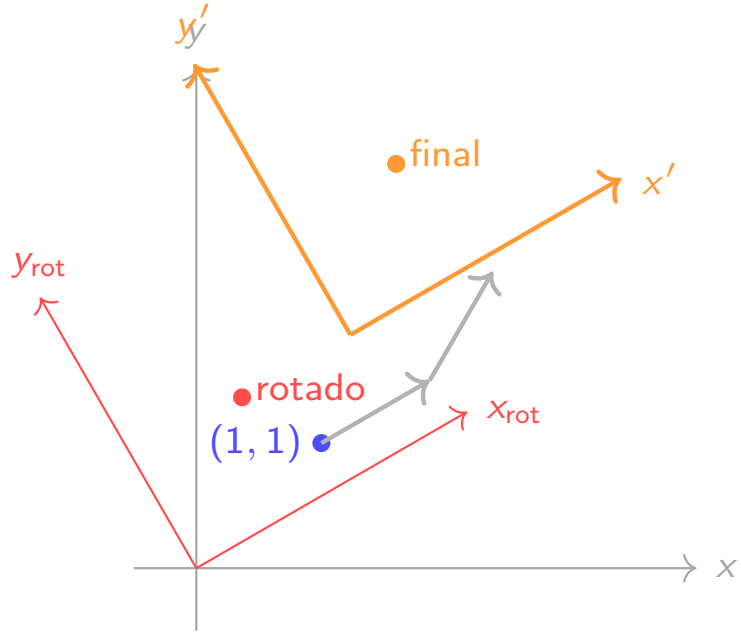
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = T_k \cdots T_2 \, T_1 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad T_i \in \mathbb{R}^{3\times 3}.$$

Ejemplo: $T_{\text{final}} = T_{\text{traslación}} \, T_{\text{rotación}}$.

Ejemplo numérico:

$$T_{\text{rot}} = \begin{pmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$T_{\text{tras}} = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$



$$T_{\text{final}} = T_{\text{tras}}\, T_{\text{rot}} = \begin{pmatrix} 0{,}866 & -0{,}5 & 2 \\ 0{,}5 & 0{,}866 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

Aplicando a $(x, y) = (1, 1)$:

$$x' = 0{,}866 - 0{,}5 + 2 = 2{,}37,$$
$$y' = 0{,}5 + 0{,}866 + 1 = 2{,}37.$$

$$\Rightarrow (x', y') = (2{,}37,\ 2{,}37)$$

Nota

El orden de multiplicación determina el resultado:

$$T_{\text{tras}}\, T_{\text{rot}} \neq T_{\text{rot}}\, T_{\text{tras}}.$$

En procesamiento de imágenes, se suele aplicar el **mapeo inverso** para evitar huecos en la imagen resultante.

Transformaciones proyectivas (homografías)

Idea: Una homografía describe una transformación de perspectiva entre dos planos:

$$\lambda \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad H \in \mathbb{R}^{3\times 3},\ H \sim \alpha H\ (\alpha \neq 0).$$

En coordenadas cartesianas:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}.$$

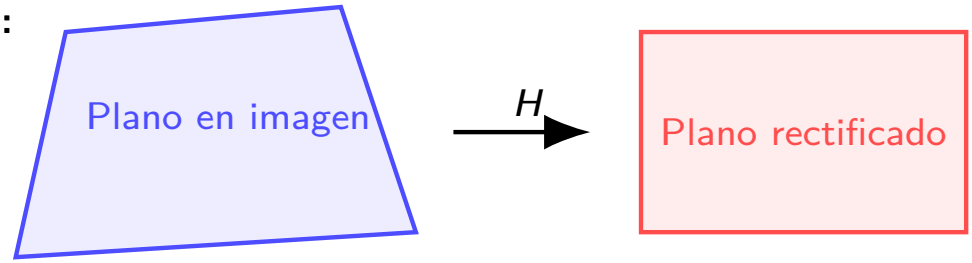
Fundamentos e intuición

- Generaliza las **transformaciones afines** añadiendo un término de *perspectiva* (denominador dependiente de x, y).
- Determinada (hasta escala) por **4 correspondencias** de puntos no colineales.
- Usos: *rectificación* de planos, mosaicos (stitching), cambio de punto de vista.

Transformaciones proyectivas (homografías)

Ejemplo numérico (perspectiva a lo largo de x):

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0{,}2 & 0 & 1 \end{pmatrix} \Rightarrow \begin{cases} x' = \dfrac{x}{0{,}2x+1}, \\ y' = \dfrac{y}{0{,}2x+1}. \end{cases}$$



$(x, y) = (2, 1) \Rightarrow$ den $= 1{+}0{,}2{\cdot}2 = 1{,}4 \Rightarrow (x', y') = \left(\frac{2}{1{,}4}, \frac{1}{1{,}4}\right) \approx (1{,}43,\ 0{,}71).$

Lectura: los puntos con x grande "se comprimen", simulando profundidad.

Práctica

- Estimar H con 4–8 correspondencias (DLT + normalización).
- Aplicar **mapeo inverso** (destino→origen) y **interpolación** (bilineal/bicúbica).

Homografía por DLT: pasos prácticos

Objetivo: hallar $H \in \mathbb{R}^{3\times 3}$ tal que $\lambda\,[x'\ y'\ 1]^T = H\,[x\ y\ 1]^T$ para $N \geq 4$ correspondencias $\{(x_i, y_i) \leftrightarrow (x_i', y_i')\}$.

1) Normalización (Hartley)

Para mejorar la condición numérica:

- Traslada los puntos para que su **centroide** esté en el origen.
- Escala para que la **distancia media** al origen sea $\sqrt{2}$.

Obtienes matrices de similitud T y T' y puntos normalizados $\tilde{\mathbf{x}} = T\mathbf{x}$, $\tilde{\mathbf{x}}' = T'\mathbf{x}'$.

2) Sistema lineal $A\,\mathbf{h} = 0$

Construcción del sistema lineal $A\,\mathbf{h} = 0$

Modelo geométrico en coordenadas homogéneas:

$$\lambda \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}.$$

En coordenadas cartesianas:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}.$$

1. **Multiplicamos cruzado** para eliminar el denominador:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13},$$
$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}.$$

2. **Reorganizamos todos los términos al mismo lado:**

$$x'h_{31}x + x'h_{32}y + x'h_{33} - h_{11}x - h_{12}y - h_{13} = 0,$$
$$y'h_{31}x + y'h_{32}y + y'h_{33} - h_{21}x - h_{22}y - h_{23} = 0.$$

3. **Vectorizamos las incógnitas:**

$$\mathbf{h} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}^T.$$

4. **Coeficientes de cada h_{ij} $\rightarrow$ una fila en A.**

Filas generadas por una correspondencia $(x, y) \leftrightarrow (x', y')$

$$\begin{bmatrix} -x - y & -1 & 0 & 0 & x\,x' & y\,x' & x' \\ 0 & 0 & -x & -y - 1 & x\,y' & y\,y' & y' \end{bmatrix}$$

5. Se apilan las $2N$ filas para N correspondencias:

$$A\,\mathbf{h} = 0, \qquad A \in \mathbb{R}^{2N \times 9}.$$

3) Solución y des-normalización

- Calcula SVD: $A = U\Sigma V^\top$. La solución es el **vector singular mínimo**: **h** = última columna de V.
- Re-da forma: $\hat{H}$ con **h** (3×3).
- Des-normaliza: $H = T'^{-1}\,\hat{H}\,T$ y escala H (p.ej. $h_{33} = 1$).

4) Consejos

- Usa **RANSAC** si hay outliers; refina con **optimización no lineal** (reproyección).
- (¿Cómo asignar valores de intensidad a los píxeles de salida, si las coordenadas transformadas no caen exactamente sobre píxeles originales?) Al evaluar, aplica **mapeo inverso** (destino→origen) + **interpolación** bilineal/bicúbica.

RANSAC: estimación robusta de modelos

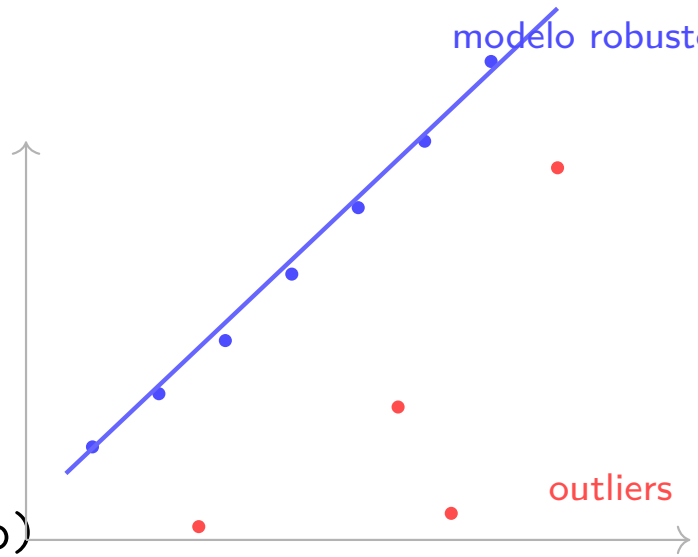
Idea general: RANSAC (*RANdom SAmple Consensus*) permite estimar un modelo (recta, plano, homografía...) **aun cuando muchos datos son erróneos (outliers)**.

Intuición

En lugar de ajustar el modelo a todos los datos (lo que los outliers distorsionan), RANSAC repite muchas veces:

1. Selecciona aleatoriamente un pequeño subconjunto (s puntos mínimos).
2. Calcula un modelo con ellos.
3. Evalúa qué puntos del conjunto completo se ajustan bien (inliers).
4. Guarda el modelo con más inliers.

Implementación práctica (pseudocódigo):




```
best_inliers = []
for i in range(N_iter):
    sample = random_subset(points, s)
    model = fit_model(sample)
    inliers = []
    for p in points:
        error = compute_error(model, p)
        if error < eps:
            inliers.append(p)
    if len(inliers) > len(best_inliers):
        best_model = model
        best_inliers = inliers
# Reajustar modelo con todos los inliers
final_model = fit_model(best_inliers)
```

En la práctica

- En OpenCV: `cv::findHomography(pts1, pts2, RANSAC, thresh)`.
- En scikit-image: `measure.ransac(data, model, min_samples, residual_threshold)`.

- Parámetros clave: nº de iteraciones, umbral de error (ε), mínimo de puntos s.

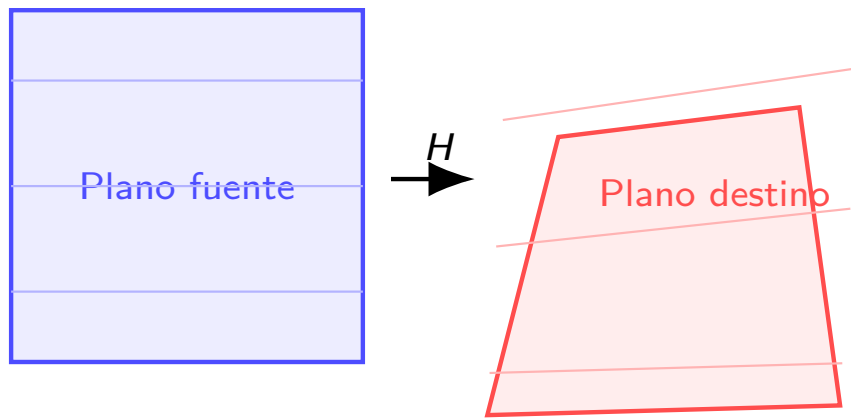
Interpretación geométrica de la homografía

Idea intuitiva: Una homografía es una transformación *proyectiva* que:

- Mapea un plano del espacio 3D sobre otro plano (por ejemplo, entre dos vistas de una cámara).
- **Preserva las líneas rectas**, pero no las distancias ni los ángulos.
- Transforma paralelas en líneas que pueden converger en un punto de fuga.
- Se representa por una matriz $H \in \mathbb{R}^{3\times 3}$ (definida hasta un factor escalar).

Efecto visual

- Rectángulos se transforman en trapecios (o cualquier cuadrilátero convexo).
- Se conserva la **incidencia**: puntos colineales siguen en la misma línea.
- Las proporciones cambian según la perspectiva (dependen de h_{31}, h_{32}).



Resumen conceptual

- H modela una proyección entre planos (por ejemplo, entre imágenes de una misma superficie).
- Las homografías son la base de la **rectificación**, el **stitching panorámico** y la **visión estereoscópica**.

Conclusiones: filtros y transformaciones geométricas

Qué hace cada familia

- **Filtros (convoluciones)**: operan sobre *intensidades* combinando píxeles de un vecindario (ej. media, gaussiano, Sobel).
- **Transformaciones geométricas**: operan sobre *coordenadas* reubicando píxeles (traslación, rotación, escala, afines, homografías).

Cuándo usar

- **Suavizado/ruido** $\Rightarrow$ gaussiano/mediana (paso bajo).
- **Detección de bordes** $\Rightarrow$ Sobel/Prewitt (gradiente) $\pm$ suavizado previo.
- **Alineación/rectificación** de vistas planas $\Rightarrow$ afines/homografías.
- **Preparación para IA** $\Rightarrow$ normalización/estandarización + filtros ligeros.

Aspectos numéricos

- **Padding** cambia la respuesta en bordes (zero/reflect/replicate).
- **Interpolación** en geometría: nearest (rápida), bilineal (equilibrio), bicúbica (suave).
- **Convolución vs. correlación**: en práctica muchas librerías usan $\star$ (sin invertir kernel).

Buenas prácticas

- Normaliza entradas (p.ej. $[0, 1]$ o z-score) para estabilidad del aprendizaje.
- Suaviza antes de derivar (Sobel) si hay ruido.
- En homografías: normaliza puntos, usa **RANSAC** y refina por reproyección.
- En pipeline: (geo) $\rightarrow$ (filtro) $\rightarrow$ (normaliza) $\rightarrow$ modelo según objetivo.

Mensaje final

Filtros $\neq$ transformaciones: unos **modulan valores**, otros **reubican posiciones**. Combinarlos correctamente mejora **calidad visual**, **robustez** y **rendimiento** en IA.

Avance: Redes Convolucionales (CNNs)

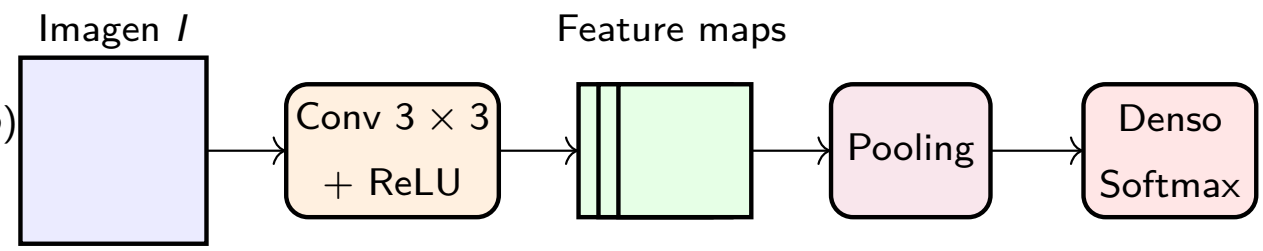
Idea clave: Las CNNs *aprenden filtros* (kernels) a partir de datos para extraer características visuales (bordes, texturas, formas) de manera jerárquica.

¿Por qué CNNs?

- **Parámetros compartidos**: un mismo kernel se aplica en toda la imagen.
- **Localidad / campos receptivos**: detectan patrones locales y los combinan.
- **Traslación-invariante** (aprox.) gracias a convolución + pooling.

Bloques fundamentales

- **Conv 2D**: feature maps $= f(W * I + b)$
- **No linealidad**: ReLU $f(x) = \max(0, x)$
- **Pooling**: reduce resolución (max/avg) y agrega robustez
- **Capas profundas**: de rasgos simples $\rightarrow$ rasgos semánticos



Conexión con hoy: de *filtros fijos* (media, Sobel, gaussiano) a *filtros aprendidos* por gradiente descendente.

Próxima sesión

Arquitectura básica (Conv–ReLU–Pool), conceptos de *padding*, *stride*, mapas de activación, *receptive field*, y entrenamiento (pérdida, backprop, regularización).