Facultad de INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA
CACIC2024

Aprendizaje Automático Profundo

Redes Convolucionales

César Estrebou

Temas

01 Introducción a Imágenes

Visión humana vs. vision artificial. Imagen Digital. Representaciones en tonos de grises y RGB. Operaciones comunes con imágenes. Extracción de Características. Filtros de convolución. Ejemplos

02 Redes Convolucionales

Redes densas vs. convolucionales. Componentes y características de una red Convolucional. Capa convolucional y pooling. Ejemplos

03 Modelos Preentrenados y Fine-tunning

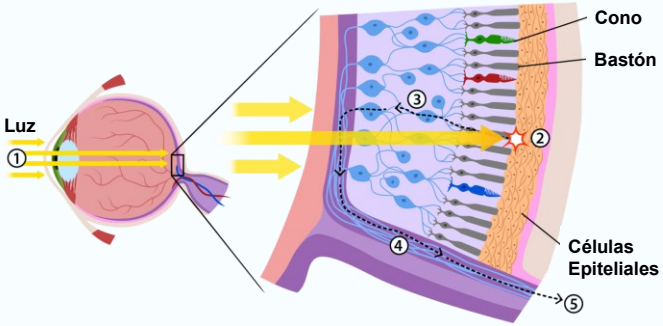
Modelos pre-entrenados. Modelo VGG16. Arquitectura. Desarrollo de ejemplo. Fine tuning. Ejemplo. Ventajas

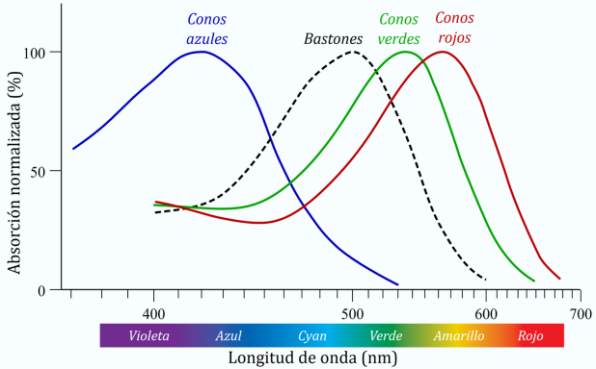
Introducción a Imágenes Digitales

Visión Humana

Sistema Visual Humano

- La luz llega a la retina donde se encuentran los fotorreceptores:
  - **Bastones**: sensibles a la luz baja, responsables de la visión en condiciones de poca iluminación
  - **Conos**: sensibles a la luz brillante y colores
- Las señales eléctricas de los fotorreceptores llegan a la corteza visual a través del nervio óptico donde de forma la imagen
- El cerebro procesa la imagen para entenderla combinando información de luz, color, profundidad y movimiento



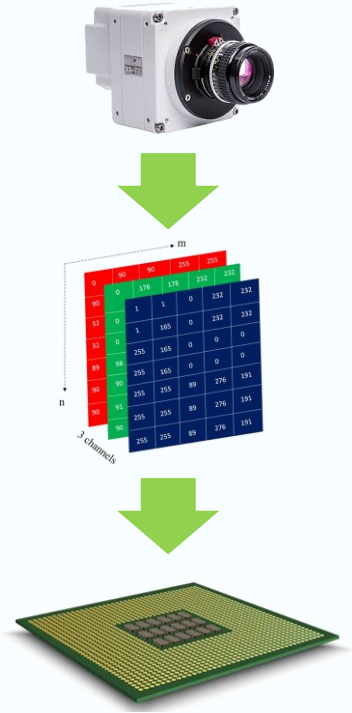


Visión Artificial

Sistema Visión Artificial

La estructura de una imagen de una computadora se basa en el sistema de visión humano

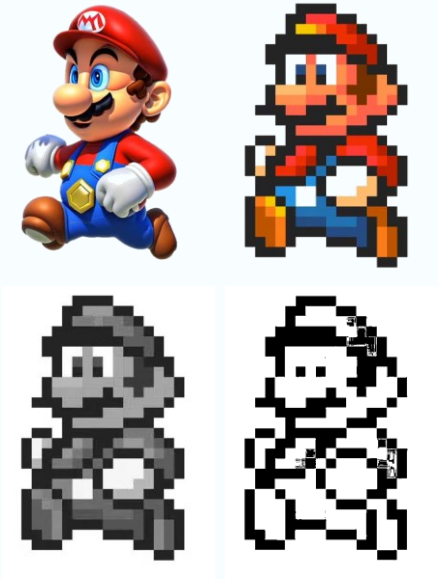
- Captura una imagen real y la convierte en datos digitalizados, procesables por la computadora
- Extrae información elemental de la imagen (bordes, formas, colores, patrones, etc.)
- Mediante algoritmos y modelos matemáticos, analiza y “entiende” el contenido de la imagen (visión por computadora, aprendizaje automático, redes neuronales, etc.)



Representación de una Imagen

Imagen Digital

- Es una representación bidimensional discreta de una imagen del mundo real
- Está compuesta por una cuadrícula de píxeles, donde cada píxel tiene un valor numérico que indica su color y posición en la imagen.
- Según la cantidad de información del píxel se pueden clasificar en monocromáticas, en escala o tonos de grises, a color.



Representación de una Imagen

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1 \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

f(x,y) tiene información de 1 pixel:

- 1 bit ➔ Imagen Binaria o Monocromática (blanco o negro)
- 8/16 bits ➔ Imagen en escala de grises (256/65K niveles de gris)
- 24bits ➔ Imágenes a color (256 niveles de intensidad en 3 canales)
- 32 bits ➔ Imágenes a color con transparencias

Imagen en Tonos de Grises

Imágenes de 8 bit/ 16 bits (Escala de grises)

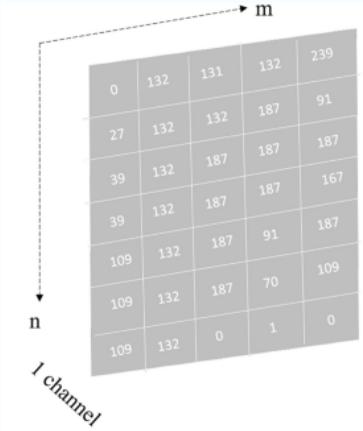


Imagen en Tonos de Grises

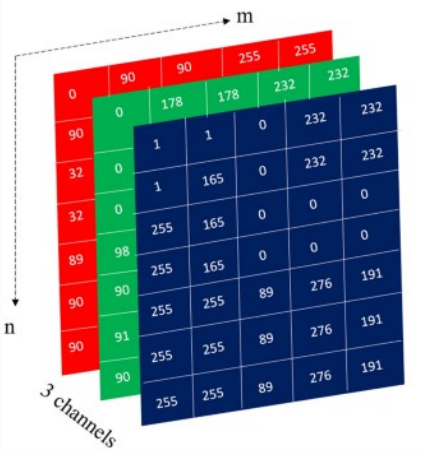
Imágenes de 8 bit / 16 bits (Escala de grises)

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Representación de una Imagen

Representación típica de una imagen RGB



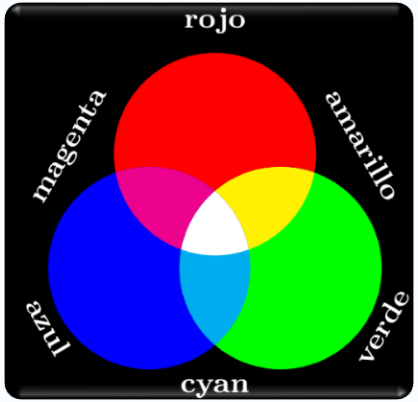
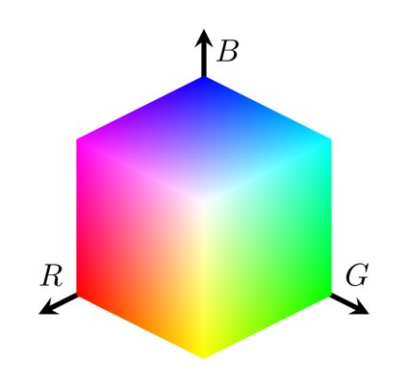


Imagen Color - RGB



Canal R (Red)
Intensidad de Rojo

Canal G (Green)
Intensidad de Verde

Canal B (Blue)
Intensidad de Azul

Imagen Color - RGB

Canal R (Red)
Intensidad de Rojo:

Canal G (Green)
Intensidad de Verde

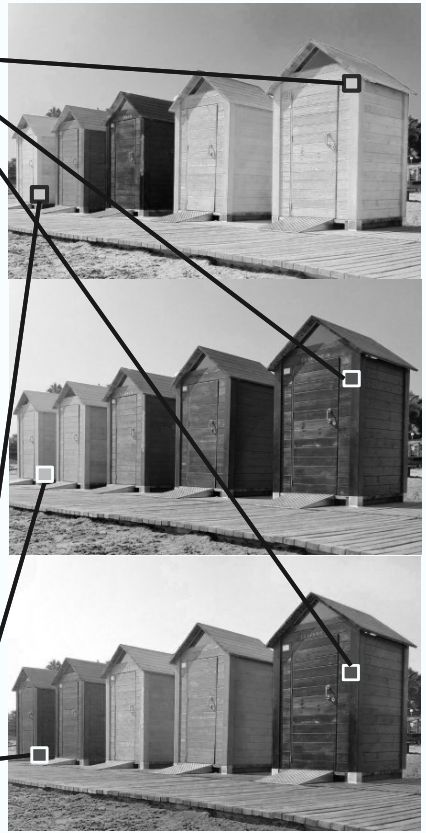
Canal B (Blue)
Intensidad de Azul

- Al igual que las imágenes en tonos de grises, cada canal RGB tiene valores que van de 0 a 255
- Normalmente, las imágenes se convierten a punto flotante y se escalan para que queden con valores entre 0 y 1

Imagen Color - RGB

R = 200
G = 82
B = 83

R = 207
G = 186
B = 96



Conversión RGB a Tonos de Grises

- La conversión a escala de grises combinar la información de los tres canales de color (rojo, verde y azul) en un solo valor de intensidad luminosa

- La fórmula más común es:

Intensidad Gris = 0.299×Rojo + 0.587×Verde + 0.114×Azul

- Los coeficientes (0.299, 0.587 y 0.114) son ponderaciones que representan la contribución relativa de cada canal al brillo percibido por el ojo humano por los conos

Trabajando con Imágenes

```python
from skimage import io
import matplotlib.pyplot as plt

DATOS_DIR = 'Datos/' # ruta a las imagenes
imgColor = io.imread(DATOS_DIR+'Casitas.jpg')

fig, axs = plt.subplots(2, 2, figsize=(10, 7))
axs = axs.flatten()

axs[0].imshow(imgColor)
axs[0].set_title('Imagen Original')
axs[0].axis('off')  # Ocultar ejes

# Mostrar las imágenes
C = ['Reds_r', 'Greens_r', 'Blues_r']
for p in range(3):
    axs[p + 1].imshow(imgColor[:, :, p], cmap=C[p])
    axs[p + 1].set_title(C[p][:-2]) # Titulos
    axs[p + 1].axis('off')          # Ocultar ejes
plt.show() # Muestra figura
```

Ejemplo de carga de imagen y visualización de canales R, G y B con skimage

Ver notebook Imagenes_con_skimage.ipynb

Trabajando con Imágenes

```python
from skimage import io, color
import matplotlib.pyplot as plt

DATOS_DIR = 'Datos/’ # ruta a las imagenes
imgColor = io.imread(DATOS_DIR + 'Casitas.jpg')

imgGray = color.rgb2gray(imgColor) # grises

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Mostrar la imagen original en la primera posición
axs[0].imshow(imgColor)
axs[0].set_title('Imagen Original')
axs[0].axis('off')  # Ocultar ejes

axs[1].imshow(imgGray, cmap='gray')
axs[1].set_title('Escala de Grises')
axs[1].axis('off')  # Ocultar ejes

plt.show() # Mostrar la figura
```

Ejemplo de conversión a tonos de grises con skimage

Ver notebook Imagenes_con_skimage.ipynb

Trabajando con Imágenes

Otras operaciones con skimage

```
from skimage import transform, exposure

# Rotar una imagen una cantidad de grados
imgRotated = transform.rotate(imgColor, 45, resize=False)

# Escalar (aumentar o disminuir) una imagen usando un factor
imgScaled = transform.rescale(imgColor, 0.5, channel_axis=-1)

# Trasladar la imagen de forma horizontal o vertical
tform = transform.SimilarityTransform(translation=(10, 10)) # en x e y
imgTranslated = transform.warp(imgColor, tform, preserve_range=True)

# Ajusta contraste de la imagen con un factor
imgContrast = exposure.adjust_gamma(imgColor, gamma=1/factor)

# Guardar una imagen usando la extensión del archivo
io.imsave(DATOS_DIR + 'Casitas.png', imgColor)
```


Ver notebook Imagenes_con_skimage.ipynb

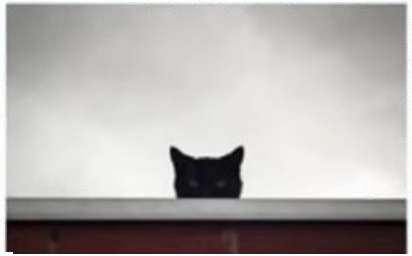
Dificultades en el procesamiento de imágenes

Ángulo de visión

Escala

Deformación

Oclusión



Condiciones de iluminación

Confusión de fondo

Variación intra-clase

Extracción de Características

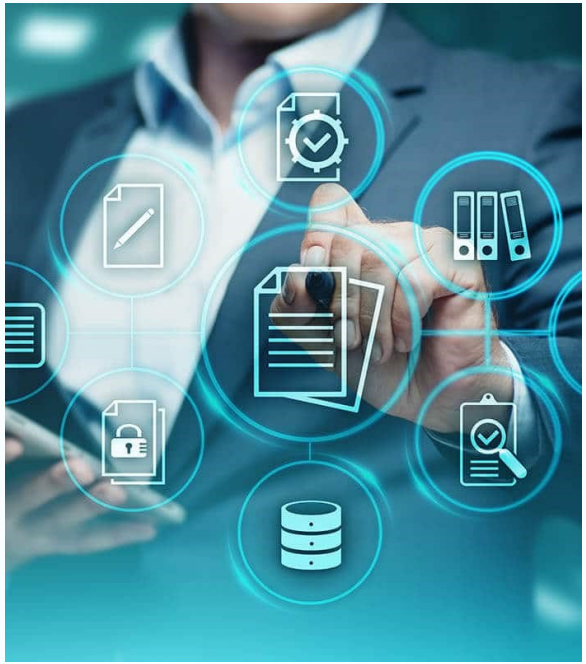
Extracción de Características de Bajo Nivel

- Identificar y capturar detalles y patrones fundamentales de la información visual en la imagen
- Estas características suelen ser básicas, como bordes, texturas y colores
- Se utilizan como bloques de construcción para análisis más avanzados en tareas como reconocimiento de objetos o clasificación de imágenes

Extracción de Características

Extracción de Características de Alto Nivel

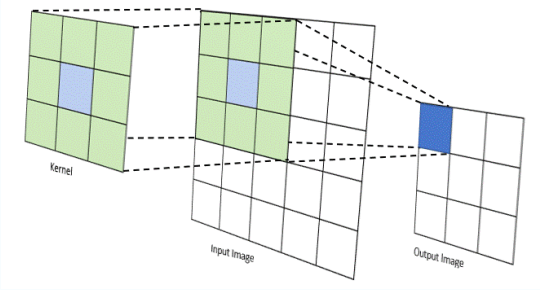
- Identificar y captura características abstractas y semánticamente significativas
- Estas características pueden incluir objetos, patrones complejos, o conceptos abstractos como la presencia de ciertos objetos o escenas
- Es un proceso que implica la combinación y comprensión de múltiples características de bajo nivel para formar representaciones más avanzadas
- Útiles para tareas como reconocimiento de objetos o clasificación de imágenes



Filtros de Convolución

La convolución discreta de dos funciones f y g se define como:

$$(f * g)[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2].g[x - n_1, y - n_2]$$



Un filtro de convolución se utiliza para resaltar características de una imagen: bordes, esquinas, texturas

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

	22			

```
1 * -1 + 2 * 0 + 8 * 1 +
1 * -1 + 4 * 0 + 9 * 1 +
1 * -1 + 2 * 0 + 8 * 1 = 22
```

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

$$2 * -1 + 8 * 0 + 2 * 1 +$$
$$4 * -1 + 9 * 0 + 3 * 1 +$$
$$2 * -1 + 9 * 0 + 2 * 1 = -1$$

Imagen de Salida

	22	-1		

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

```
8 * -1 + 2 * 0 + 1 * 1 +
9 * -1 + 3 * 0 + 1 * 1 +
8 * -1 + 2 * 0 + 1 * 1 = 22
```

Imagen de Salida

	22	-1	22	

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

	22	-1	-22	
	21			

```
1 * -1 + 4 * 0 + 9 * 1 +
1 * -1 + 2 * 0 + 8 * 1 +
1 * -1 + 3 * 0 + 7 * 1 = 21
```

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

$$4 * -1 + 9 * 0 + 3 * 1 +$$
$$2 * -1 + 8 * 0 + 2 * 1 +$$
$$4 * -1 + 7 * 0 + 1 * 1 = -3$$

Imagen de Salida

	22	-1	-22	
	21	-3		

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

	22	-1	-22	
	21	-3	-20	

```
9 * -1 + 3 * 0 + 1 * 1 +
8 * -1 + 2 * 0 + 1 * 1 +
7 * -1 + 1 * 0 + 2 * 1 = -20
```

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

$$1 * -1 + 2 * 0 + 8 * 1 +$$
$$1 * -1 + 3 * 0 + 7 * 1 +$$
$$1 * -1 + 2 * 0 + 8 * 1 = 20$$

Imagen de Salida

	22	-1	-22	
	21	-3	-20	
	20			

Filtros de Convolución

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

$$2 * -1 + 8 * 0 + 2 * 1 +$$
$$3 * -1 + 7 * 0 + 1 * 1 +$$
$$2 * -1 + 8 * 0 + 2 * 1 = -2$$

Imagen de Salida

	22	-1	-22	
	21	-3	-20	
	20	-2		

Filtros de Convolución

Observar que el tamaño de la salida es menor que la entrada

Aplicación de filtro paso a paso:

Imagen de Entrada

1	2	8	2	1
1	4	9	3	1
1	2	8	2	1
1	3	7	1	2
1	2	8	2	2

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

22	-1	-22
21	-3	-20
20	-2	-19

8 * −1 + 2 * 0 + 1 * 1 +
7 * −1 + 1 * 0 + 2 * 1 +
8 * −1 + 2 * 0 + 2 * 1 = -21

Filtros de Convolución

- Para mantener el tamaño de la imagen de salida se suelen agregar filas y columnas adicionales, según el tamaño del kernel

- A estos valores adicionales se los conoce como **padding**

- Dependiendo de la aplicación estos valores adicionales se pueden completar de varias maneras:
  - **Extender**: Replicar el valor del borde más cercano.
  - **Envolver**: Utilizar los valores del borde opuesto, como si la imagen se envolviera.
  - **Espejar**: Reflejar la secuencia de valores del borde más cercano.
  - **Constante**: Asignar un valor fijo a los píxeles adicionales.

Filtros de Convolución

Aplicación de filtro paso a paso con padding constante:

Se agregan ceros en los bordes de la imagen original y la imagen de salida queda del mismo tamaño y se hace el mismo proceso

Imagen de Entrada

0	0	0	0	0	0	0
0	1	2	8	2	1	0
0	1	4	9	3	1	0
0	1	2	8	2	1	0
0	1	3	7	1	2	0
0	1	2	8	2	2	0
0	0	0	0	0	0	0

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

6				

```
0 * -1  +  0 * 0 +  0 * 1  +
0 * -1  +  1 * 0 +  2 * 1  +
0 * -1  +  1 * 0 +  4 * 1  = 6
```

Filtros de Convolución

La cantidad de filas y columnas dependen del tamaño del kernel

Aplicación de filtro paso a paso con padding constante:

Resultado final

Imagen de Entrada

0	0	0	0	0	0	0
0	1	2	8	2	1	0
0	1	4	9	3	1	0
0	1	2	8	2	1	0
0	1	3	7	1	2	0
0	1	2	8	2	2	0
0	0	0	0	0	0	0

Kernel

-1	0	1
-1	0	1
-1	0	1

Imagen de Salida

6	**15**	**-1**	**-15**	**-5**
8	**22**	**-1**	**-22**	**-7**
9	**21**	**-3**	**-20**	**-6**
7	**20**	**-2**	**-19**	**-5**
5	**13**	**-2**	**-12**	**-3**

Filtros de Convolución - Ejemplos

Filtros de Suavizado

Producen imágenes borrosas eliminado ruido o detalles

Filtro Gaussiano (σ=1)

Filtro Promedio en RGB

1/1003 x

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	53	97	53	13	1
2	22	97	159	97	22	2
1	13	53	97	53	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

$$\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Kernel (aproximado por enteros)

1/25 x

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Ver notebook Filtros_con_skimage.ipynb

Filtros de Convolución - Ejemplos

Filtros de Realce

Bordes de Roberts (Escala de Grises)

1	0
0	-1

+

0	1
-1	0

Bordes de Sobel (Escala de Grises)

-1	0	1
-2	0	2
-1	0	1

+

-1	-2	-1
0	0	0
1	2	1

Resaltan detalles en las imágenes como bordes

Sharpening (Afilado)

0	-1	0
--1	5	-1
0	-1	0

Ver notebook Filtros_con_skimage.ipynb

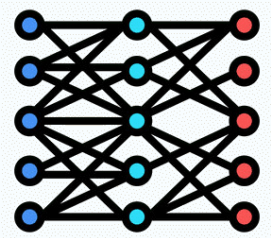
Redes Neuronales Convolucionales

Red Neuronal Convolucional

Problemas de las Redes Densas con Imágenes

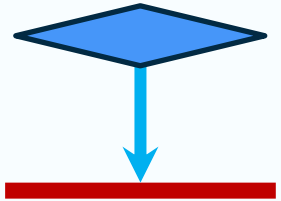
- Tienen un tamaño muy grande:

  Una imagen de 100 x 100 píxeles de entrada con 100 neuronas en la primera capa, requiere de 1.000.000 de pesos sin contar sesgos (bias)



- “Aplanamiento" (flattening) de la imagen:

  Requiere convertir la matriz en vector para poder introducirla a la red, lo que pierde la estructura espacial de los píxeles

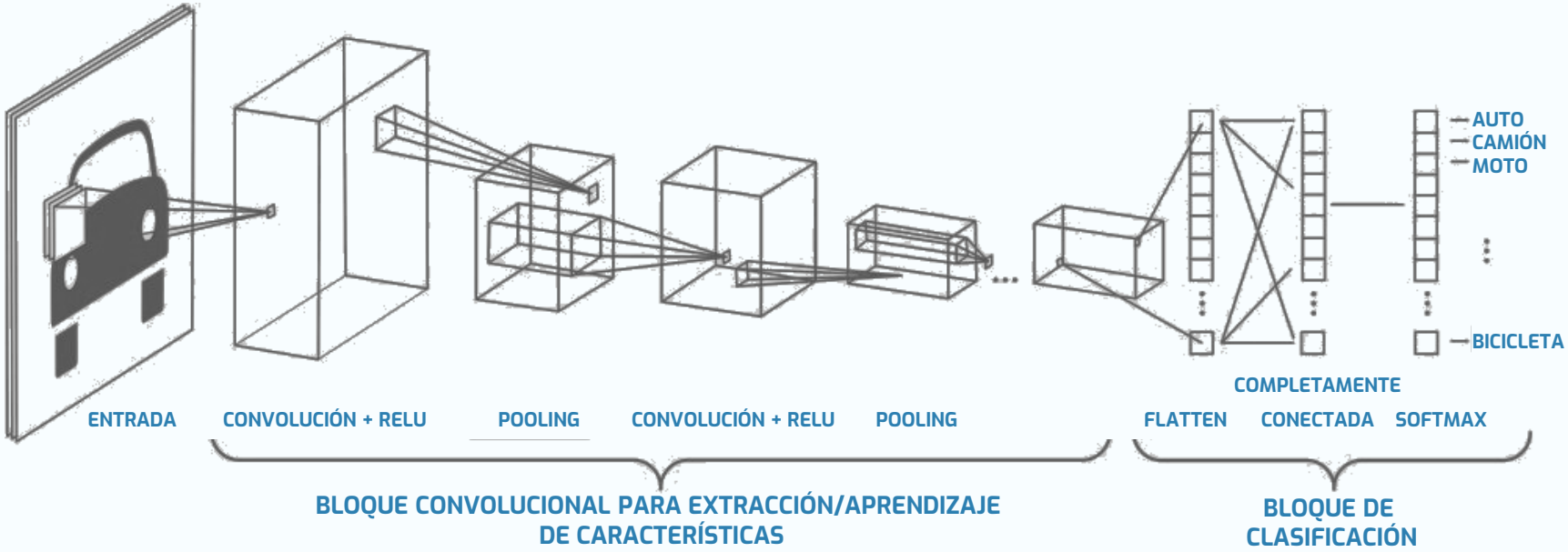


Red Neuronal Convolucional

Componentes de una Redes Convolucionales

- **Capa convolucional:**
  - Aprende y resalta características relevantes en los datos de entrada
- **Función de Activación (ReLU/Leaky ReLU):**
  - Introduce no linealidades para mejorar la capacidad de la red de aprender patrones complejos
  - Mejora la eficiencia computacional
  - Ayuda a mitigar el problema del desvanecimiento del gradiente
- **Capa de Pooling:**
  - Rduce la dimensionalidad de la salida de la capa convolucional, conservando las características más importantes.

Redes Neuronales Convolucionales



Capa Convolucional

Filtros/Kernels:

- Aplican filtros/kernels especializados que actúan como detectores de características
- Cada filtro aprende a detectar patrones específicos (bordes, texturas, formas)
- Procesan automáticamente características jerárquicas de la imagen

Estructura:

- Utilizan filtros/kernels para cada canal de la imagen
- Incluyen un término de sesgo (bias) que ajusta la intensidad de la activación
- Comparten pesos en toda la imagen (weight sharing), reduciendo parámetros

Capa Convolucional

Proceso de Aprendizaje:

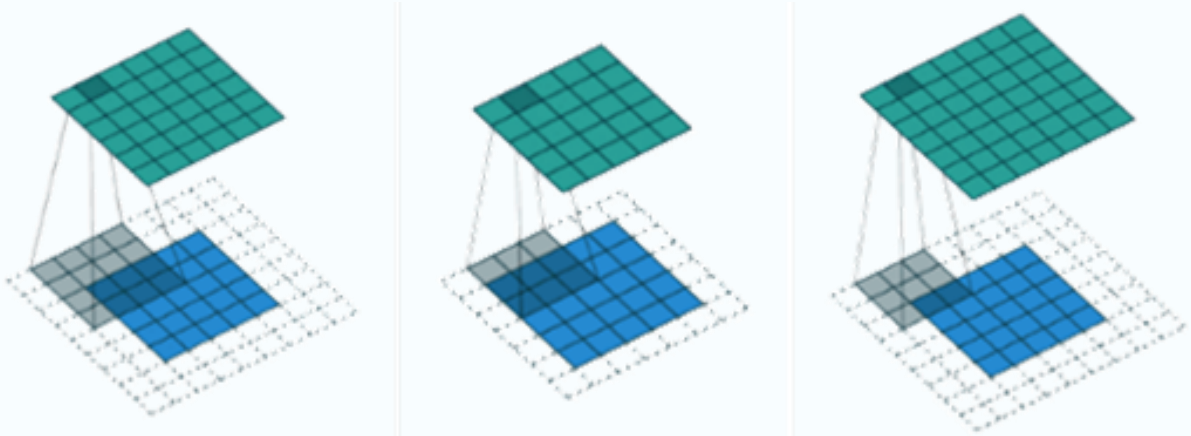
- Emplean **backpropagation** para optimizar automáticamente los valores de los filtros (pesos)
- Ajustan gradualmente los pesos para minimizar el error de predicción
- Aprenden representaciones cada vez más abstractas en capas más profundas

Ventajas Clave:

- Reutilización eficiente de pesos mediante el weight sharing
- Reducción significativa de parámetros comparado con redes densas
- Preservación de la estructura espacial de la imagen
- Invarianza a la traslación de objetos en la imagen

Convolución 2D - Padding

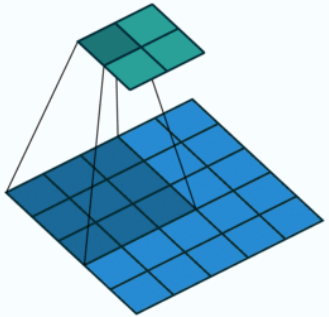
- El padding es una técnica que consiste en añadir píxeles alrededor de una imagen o mapa de características para mantener el tamaño original de la entrada al aplicar un filtro de convolución



Convolución 2D - Stride

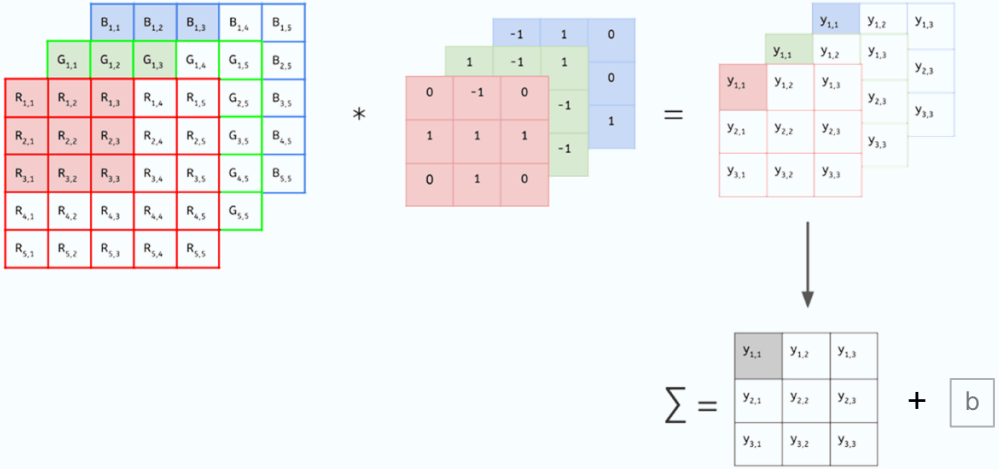
- Adicional padding en la convolución se puede aplicar el stride
- El stride indica la cantidad de pixeles que debe desplazarse (horizontal y vertical) el filtro al aplicar la convolución
- Para kernels mayores que 3 tiene sentido incrementar el desplazamiento para mejorar la superposición de los mismos

Stride=1

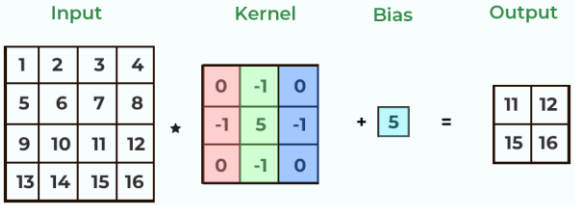


Stride=2

Convolución 2D



Convolución 2D con 3 canales

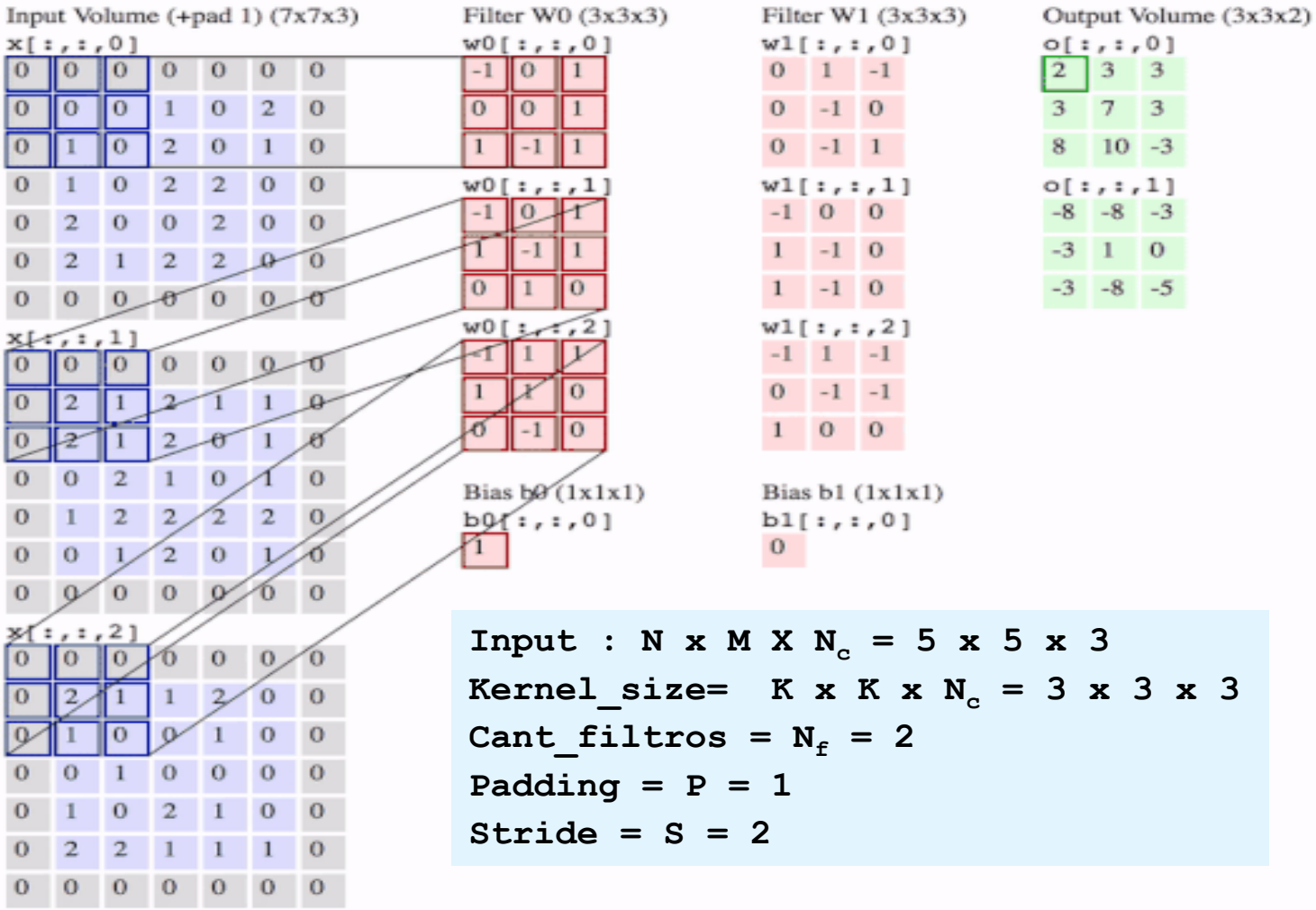


Convolución 2D con 1 canal

Capa Convolucional

```
tensorflow.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    activation=None,
    use_bias=True
)
```

- **filters:** cantidad de canales o mapas de características tendrá la salida de la capa convolucional. Filters = 32 genera 32 “imágenes” de salida
- **kernel_size:** tamaño del kernel o ventana de convolución
- **strides:** paso o desplazamiento del kernel strides = 1 significa que se mueve pixel por pixel. Valores mayores reduce el tamaño de la salida
- **padding:** Indica si se usa padding o no. “valid” no añade padding, la salida será más pequeña. "same": añade padding la salida. Influye en el tamaño de la salida
- **activation:** función aplicada después de la convolución. Normalmente se us “relu” o “leaky relu” para imágenes
- **use_bias:** indica si se incluye el término de sesgo (bias) en el cálculo del filtro



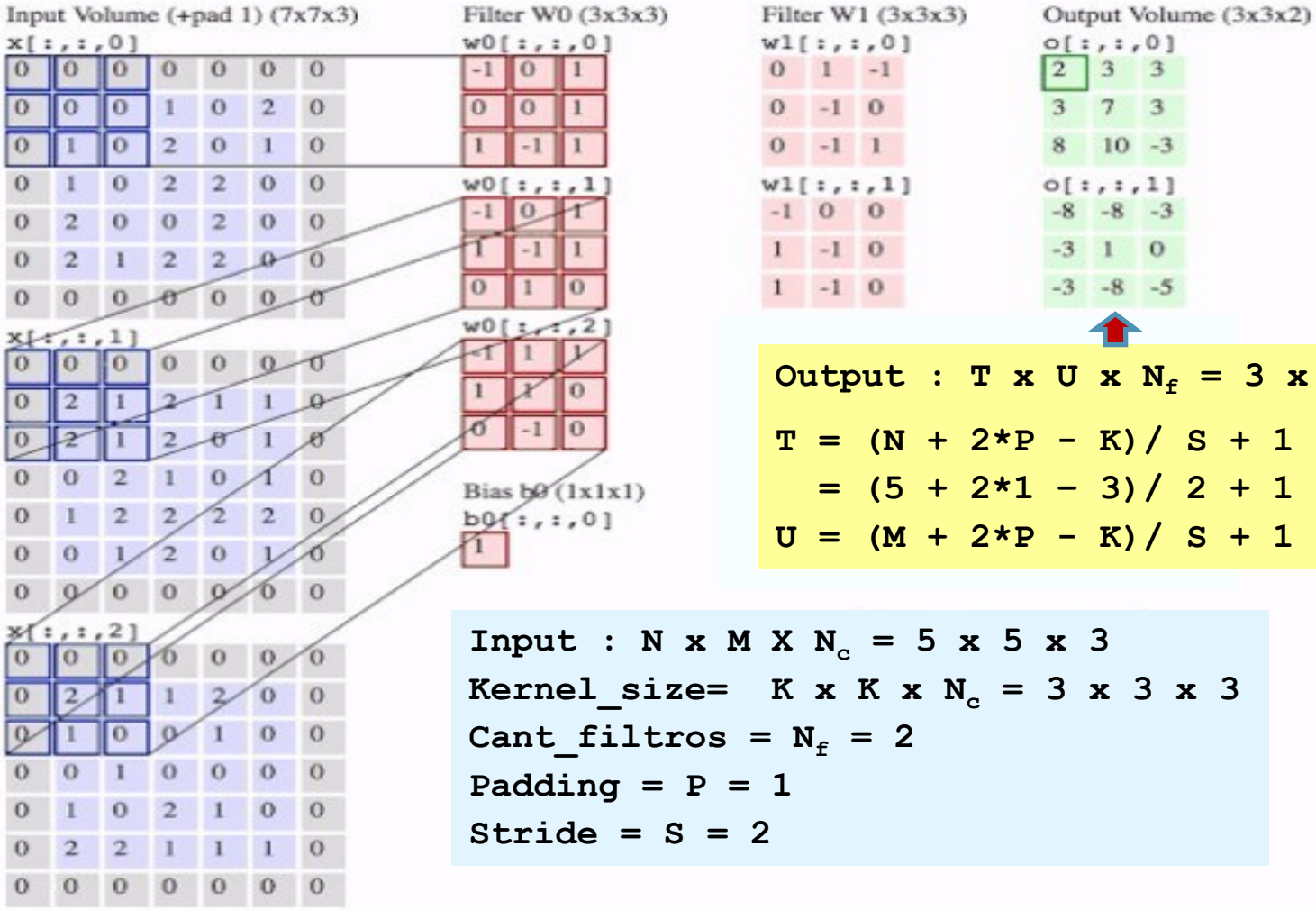
Input : N x M X N_c = 5 x 5 x 3

Kernel_size= K x K x N_c = 3 x 3 x 3

Cant_filtros = N_f = 2

Padding = P = 1

Stride = S = 2



Output : T x U x N_f = 3 x 3 x 2

T = (N + 2*P - K)/ S + 1

= (5 + 2*1 - 3)/ 2 + 1 = 3

U = (M + 2*P - K)/ S + 1 = 3

Input : N x M X N_c = 5 x 5 x 3

Kernel_size= K x K x N_c = 3 x 3 x 3

Cant_filtros = N_f = 2

Padding = P = 1

Stride = S = 2

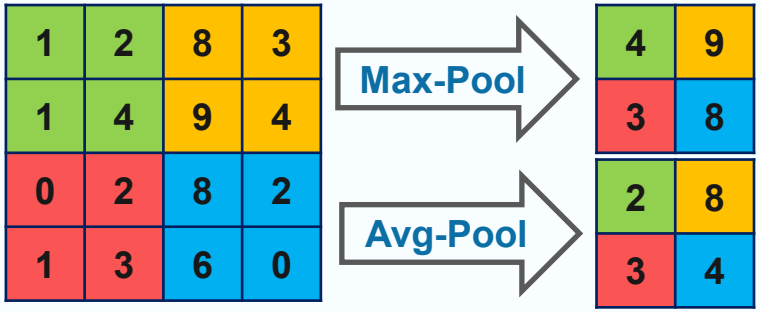
Capa de Pooling

- La capa de pooling reduce el tamaño de la salida de la capa convolucional
- Se aplica haciendo el recorrido sobre el canal de entrada similar al de la convolución con un stride igual al tamaño del kernel
- En el espacio del kernel se aplica una función que retorna un único calor sobre todos los pixels
- Hay 2 tipos de pooling: Max-Pooling y Average-Polling

Capa de Pooling

Tipos de Pooling

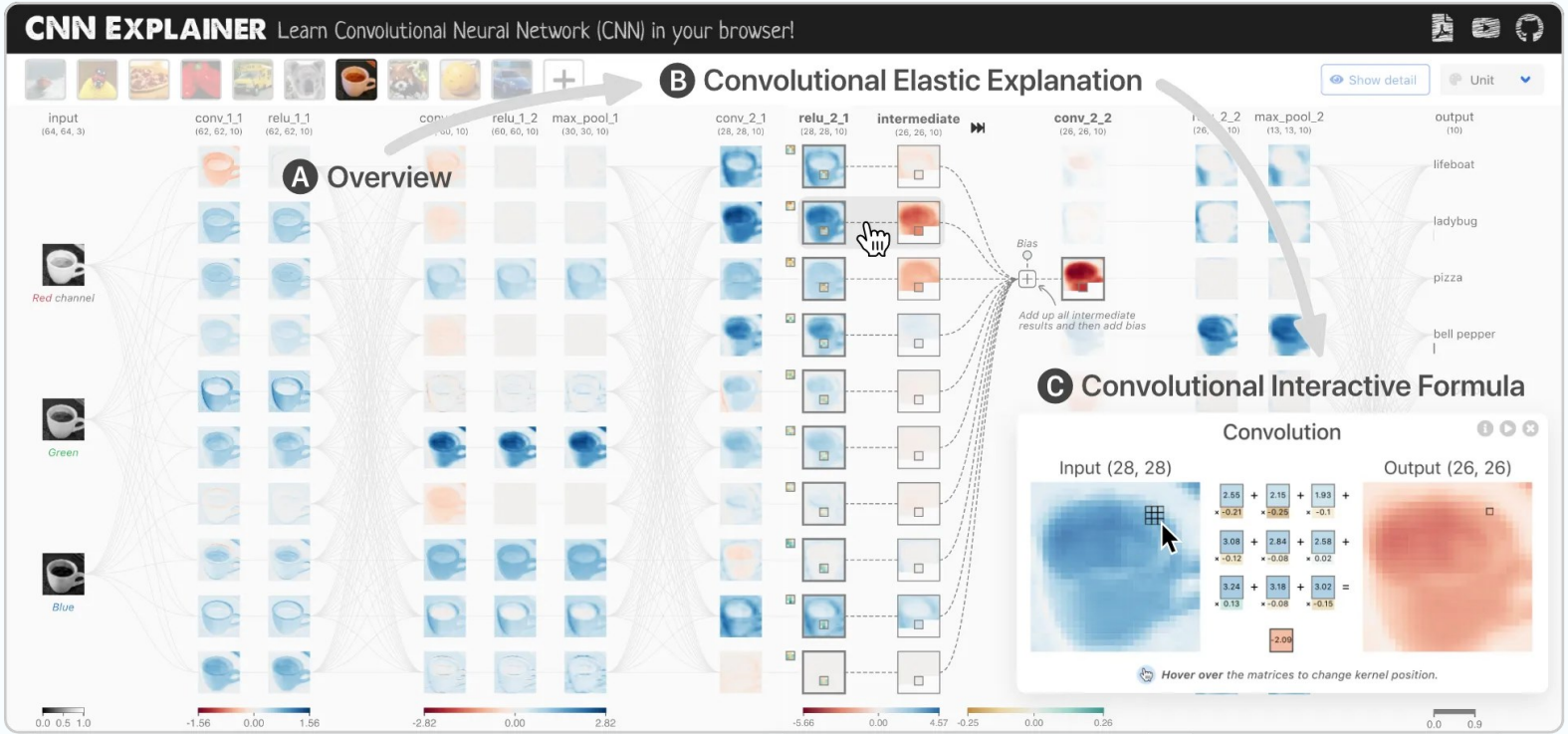
- **Max-Pooling:** obtiene el máximo valor del espacio del kernel
- **Average-Polling:** obtiene valor el promedio dentro del espacio del kernel



Capa de Pooling - Ventajas

- Reducción de dimensionalidad:
  - Reducen el tamaño de los mapas de características
  - La red es más eficiente computacionalmente y requiera menos memoria
- Invarianza a pequeñas traslaciones:
  - La red se vuelve más robusta a pequeños cambios en la posición de las características. Si un objeto está ligeramente desplazado en la imagen, la salida del pooling será similar
- Control del sobreajuste:
  - Al reducir la cantidad de parámetros, ayuda a prevenir el sobreajuste (overfitting). Actúa como una forma de regularización al generalizar las características espaciales
- Jerarquía de características:
  - Las capas más profundas detectan características más abstractas y de mayor nivel
- No tienen parámetros entrenables, lo que simplifica el entrenamiento

Red Convolucional – Ejemplo Interactivo



https://poloclub.github.io/cnn-explainer

Red Convolucional con MNIST

La base de datos MNIST contiene imágenes de 28×28, en escala de grises, de números escritos a mano.

Implementar un modelo de red neuronal convolucional para reconocer los dígitos de MNIST.

Dividir los datos en entrenamiento y prueba y medir el accuracy en ambos grupos.

Red Convolucional con MNIST

```
# Construccion del modelo
PADDING='same'
ACTIV=relu'

model = Sequential()

model.add(Input( (*IMG_SHAPE, 1) ))
model.add(Conv2D(16, kernel_size=(3,3), strides=(1,1), activation=ACTIV, padding=PADDING ))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, kernel_size=(3,3), strides=(1,1), activation=ACTIV, padding=PADDING ))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), strides=(1,1), activation=ACTIV, padding=PADDING ))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64, activation=ACTIV))
model.add(Dense(TARGET_CNT, activation='softmax'))
```


Ver notebook Convolucional_Digitos_MNIST.ipynb

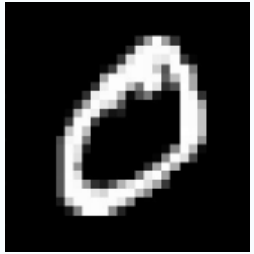
Red Convolucional con MNIST

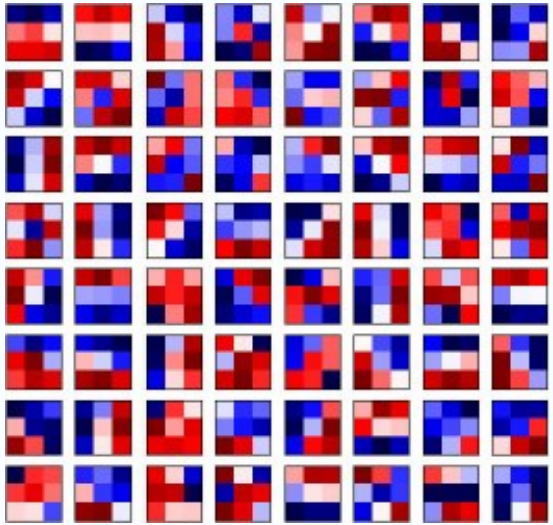
Model: "sequential"

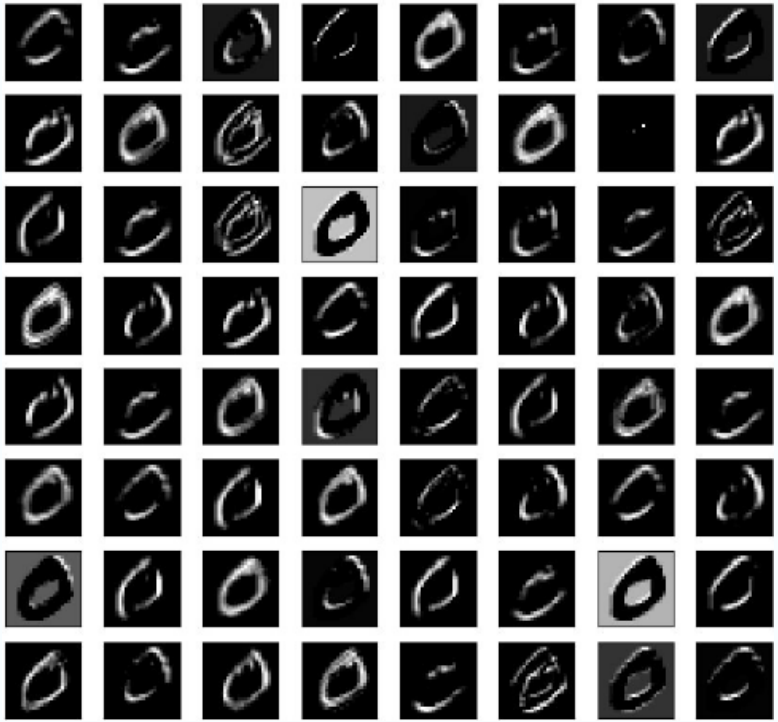
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 10)	650

Total params: 60,874 (237.79 KB)
Trainable params: 60,874 (237.79 KB)
Non-trainable params: 0 (0.00 B)

Red Convolucional con MNIST







Resultado de la aplicación
de los 64 filtros

Modelos Preentrenados

Y Fine-Tunning

Imagenet

- BBDD utilizada para reconocimiento de objetos en imágenes.
- Contiene 14 millones de imágenes etiquetadas con nombres de objetos de más de 20.000 categorías

ILSVRC

- ImageNet Large Scale Visual Recognition Challenge
- ~1.2 millones de imágenes de entrenamiento
- 50.000 imágenes de validación 100.000 imágenes de prueba

Modelos Pre-entrenados

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base
- Se utilizan estas redes y se ajusta un clasificador (ANN):
  - VGG16
  - Inception V3
  - Xception
  - ResNet50
  - MobilNet

Red neuronal convolucional con 16 capas propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford

El modelo se presentó al Desafío de Reconocimiento Visual a Gran Escala de ImageNet (ILSVRC) en 2014.

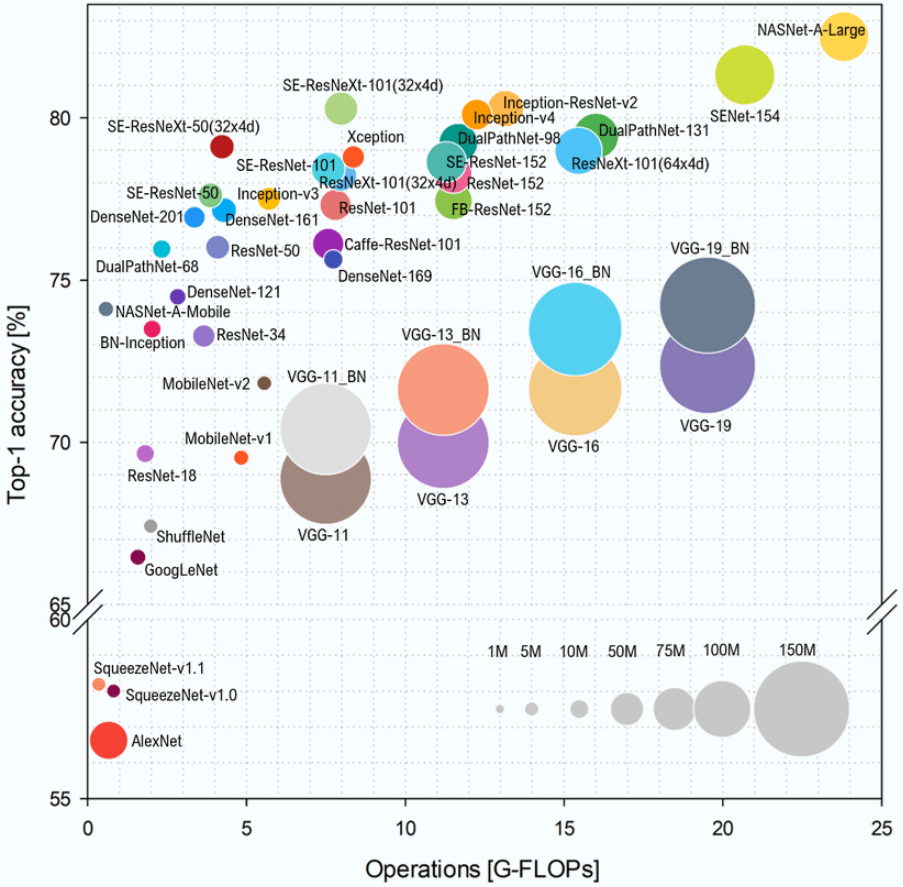
Modelos Pre-entrenados

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base
- Se utilizan estas redes y se ajusta un clasificador (ANN):
  - VGG16
  - Inception V3
  - Xception
  - ResNet50
  - MobilNet

Red neuronal convolucional entrenada con el conjunto de datos de ImageNet que tiene 50 capas y que ganó el primer puesto en el ILSVRC en 2015

Modelos Pre-entrenados

Evolución de Modelos Convolucionales



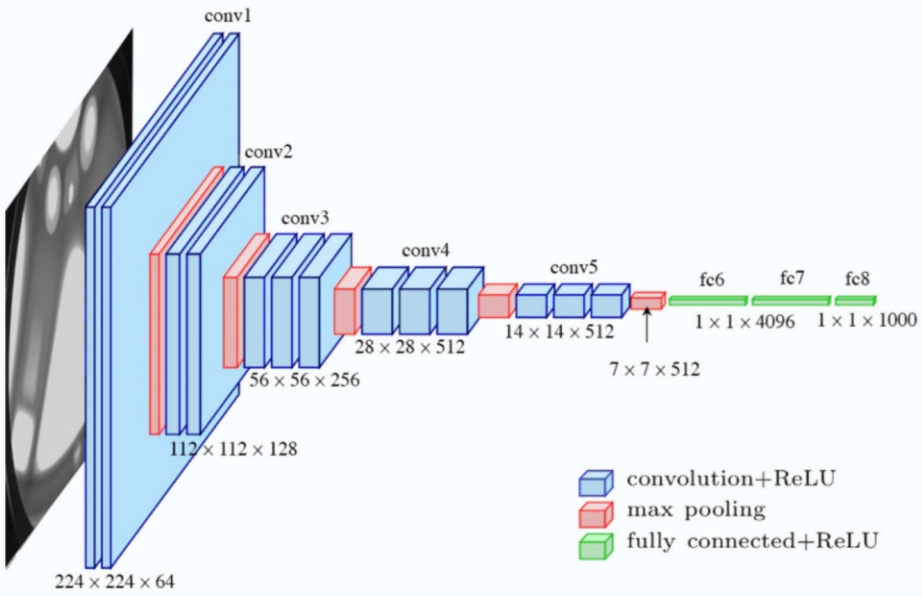
Modelo VGG16 - Creación

```
# Importa paquetes con el modelo VGG16 y funciones adicionales necesarias:
#  - preprocesamiento de la imagen antes de hacer la predicción
#  - decodificación de los resultados de la predicción: ID ➔ Nombre
#  - carga de imágenes
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.vgg16 import decode_predictions
from tensorflow.keras.preprocessing import image


# Cargar el modelo VGG16 preentrenado con pesos de ImageNet
model = VGG16(weights='imagenet')
```

Modelo VGG16 - Arquitectura

```python
# Arquitectura e información del modelo
print(model.summary())
```



Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_9 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)
Trainable params: 138,357,544 (527.79 MB)
Non-trainable params: 0 (0.00 B)

Modelo VGG16 – Preparación de Imagen

```
# Cargar y preprocesar la imagen
imgColor = image.load_img(Caballo.jpg', target_size=(224, 224))

# convierte a arreglo numpy
img_array = image.img_to_array(imgColor)

# agrega dimension requerida por modelo: (1, 224, 224, 3)
img_array = np.expand_dims(img_array, axis=0)

# preprocesa imagen para dar formato correcto para VGG16
img_array = preprocess_input(img_array)
```

Ver notebook Ejemplo_VGG16.ipynb

Modelo VGG16 – Predicción

```python
# Hacer la predicción
predictions = model.predict(img_array)

# Devuelve puntaje y nombre de las 5 mejores coincidencias
decoded_predictions = decode_predictions(predictions, top=5)[0]

# Recorre las mejores e imprime posición, etiqueta y puntaje de prediccion
print('Predicciones:')
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f" {i + 1}: {label} ({score:.3f})")
```

Salida:

```
Predicciones:
 1: cheetah (0.989)
 2: jaguar (0.006)
 3: leopard (0.004)
 4: snow_leopard (0.000)
 5: lynx (0.000)
```

Ver notebook Ejemplo_VGG16.ipynb

Ajuste Fino de Modelos Preentrenados

- Se busca reusar la base convolucional y mejorar la respuesta del clasificador en una tarea específica

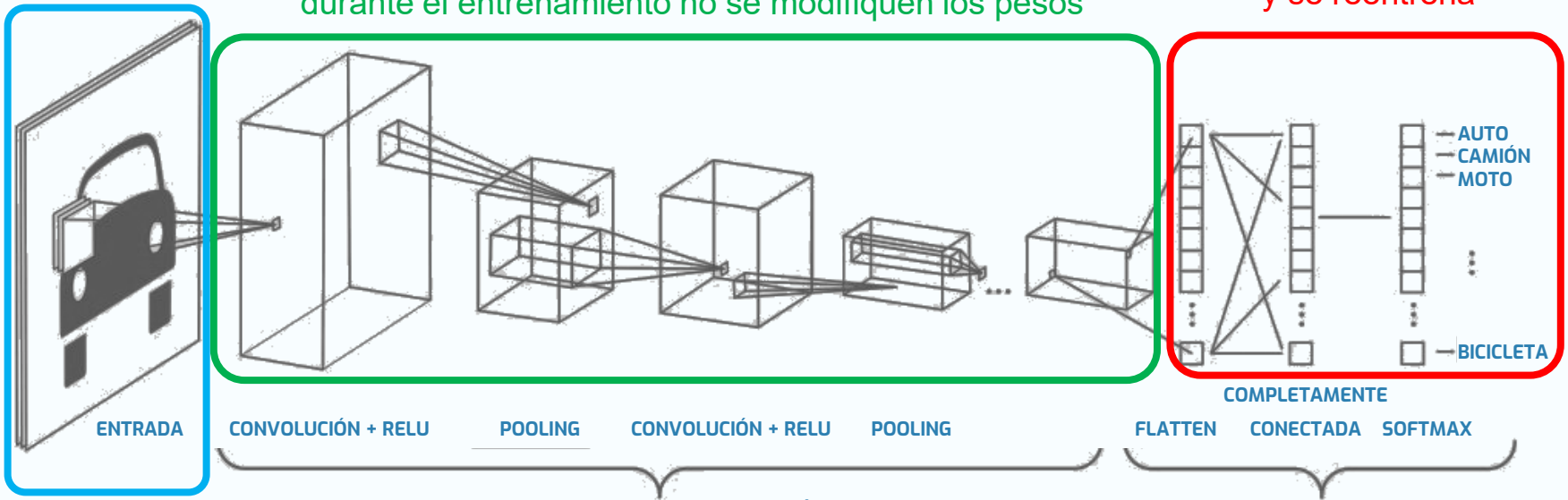
- Pasos para efectuar el **ajuste fino (Fine Tuning)**:
  - Añadir un clasificador (RNA) sobre un sistema preentrenado
  - Fijar la base convolucional y entrenar la red
  - Entrenar conjuntamente el clasificador añadido y la base convolucional

Consultar Capítulo 8 del libro "The Deep Learning with Keras Workshop. An Interactive Approach to Understanding Deep Learning with Keras (2020)"

Redes Neuronales Convolucionales

Se mantiene este bloque y se “congela ”para que durante el entrenamiento no se modifiquen los pesos

Este bloque se reemplaza y se reentrena



Se utiliza un dataset propio

El bloque convolucional extrae de las nuevas imágenes las características aprendidas anteriormente. El bloque de clasificación las utiliza para entrenar

Ajuste Fino de Modelos - Ventajas

- Ahorro de Tiempo y Recursos
  - No necesitas entrenar una red desde cero
  - Requiere menos poder computacional
  - Menor tiempo de entrenamiento

- Menor Cantidad de Datos Necesarios
  - Funciona bien con datasets pequeños
  - El modelo tiene características básicas aprendidas de las capas convolucionales
  - Solo "refina" el conocimiento para tu tarea específica

- Mejor Generalización
  - Los modelos pre-entrenados ya han aprendido características generales robustas
  - Menor riesgo de overfitting en datasets pequeños
  - Mejor desempeño inicial