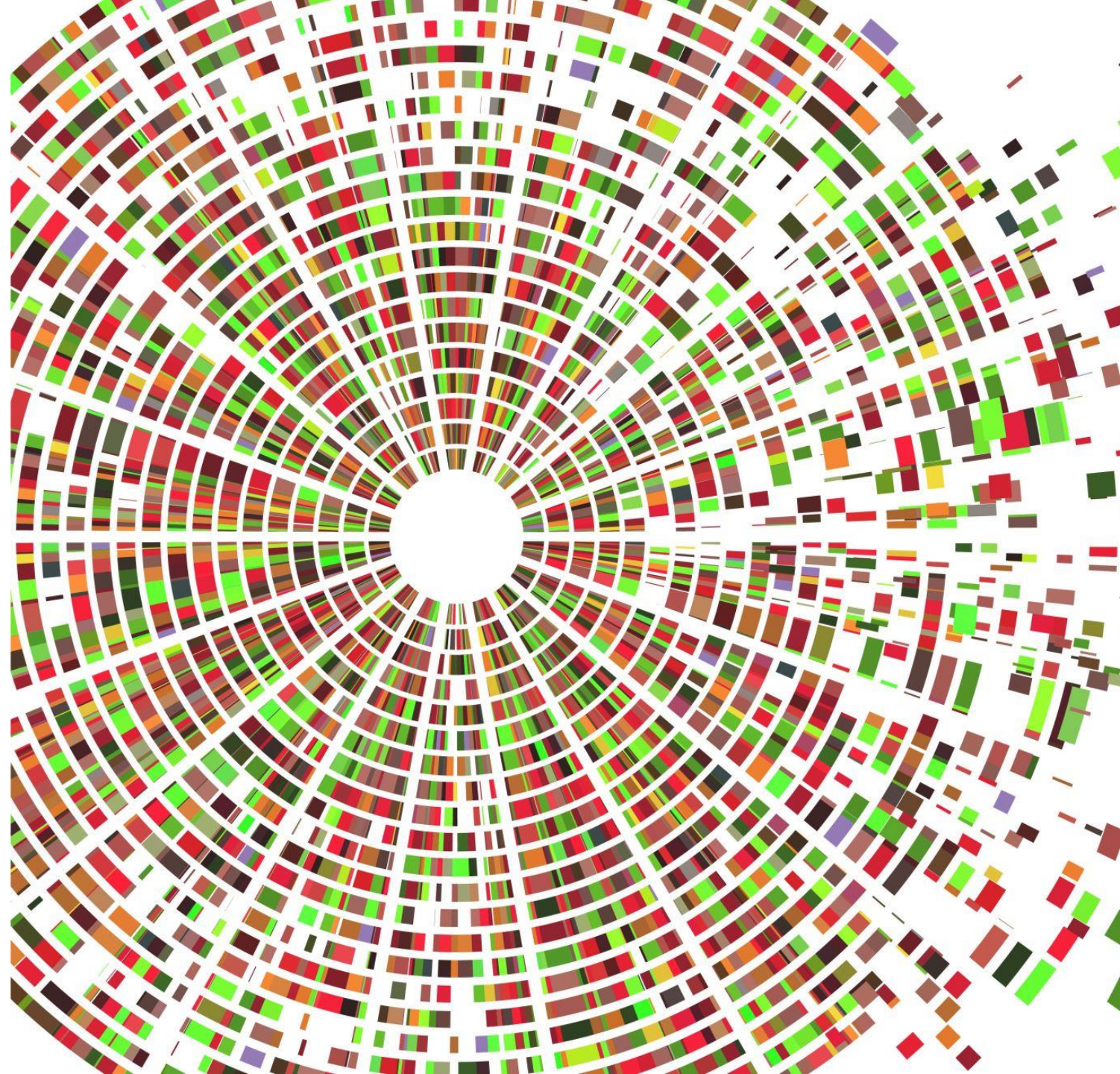
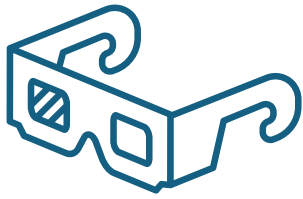


Métodos avanzados de ciencia de datos

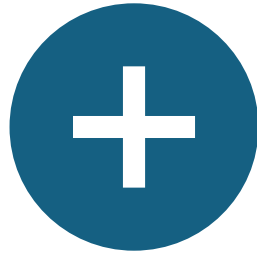
Prof. Emily Díaz



Contenido



Arquitecturas CNN
famosas



Aumentación de datos



Segmentación de
imágenes

Arquitecturas famosas de CNN



LeNet (1998)

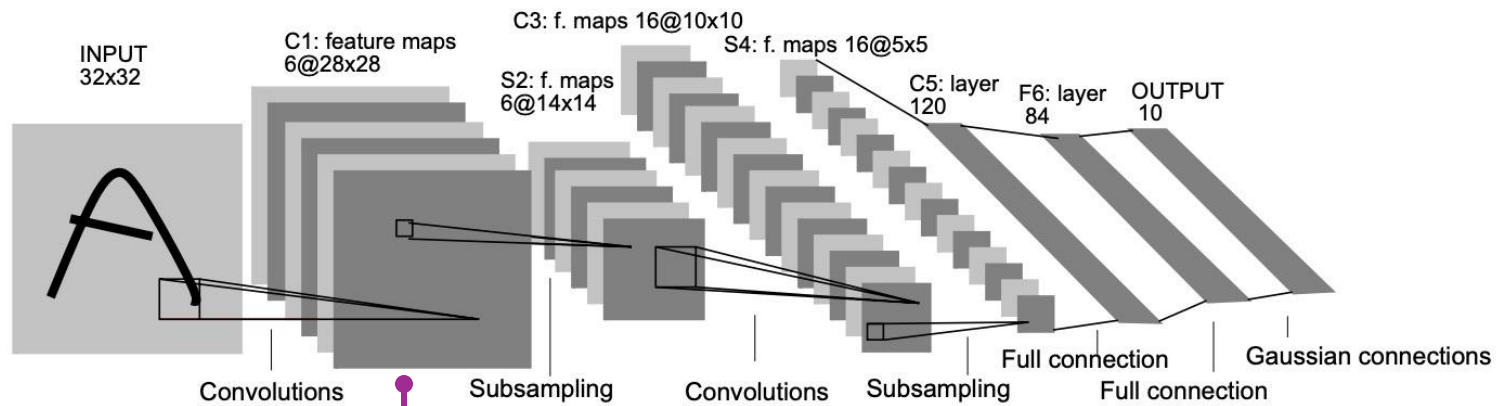


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Tiene 6 filtros y crea mapas de características de 28x28

¿Cuánto es el tamaño del kernel en C1 para obtener el 28x28? Sin relleno y paso = 1

1. (C1): Convolución
2. (S1): Activación (Sigmoid/Tanh/ReLU)
3. (S2): Agrupamiento promedio
4. (C3) Convolución
5. (S3): Activación (Sigmoid/Tanh/ReLU)
6. (S4) Agrupamiento promedio
7. (C5) Convolución
8. Activación (Sigmoid/Tanh/ReLU)
9. (F6) Capa totalmente conectada
10. (OUTPUT) Capa de salida (softmax)

LeNet (1998)

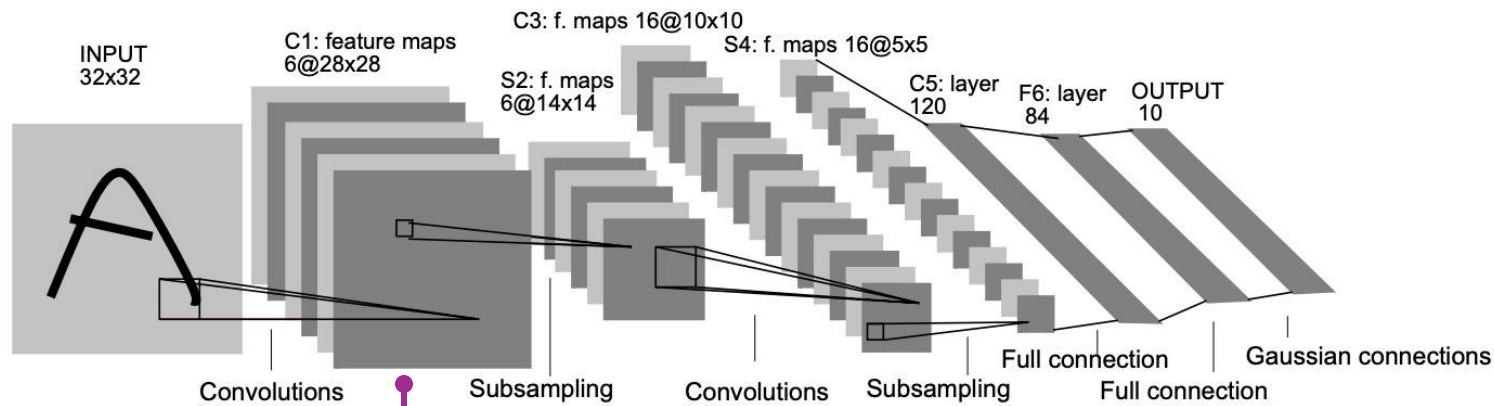


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Tiene 6 filtros y crea mapas de características de 28x28

$$\begin{aligned} 28 &= (I - K) / S + 1 \\ 32 - 27 * 1 &= K \\ K &= 5 \rightarrow 5 \times 5 \end{aligned}$$

1. (C1): Convolución
2. (S1): Activación (Sigmoid/Tanh/ReLU)
3. (S2): Agrupamiento promedio
4. (C3) Convolución
5. (S3): Activación (Sigmoid/Tanh/ReLU)
6. (S4) Agrupamiento promedio
7. (C5) Convolución
8. Activación (Sigmoid/Tanh/ReLU)
9. (F6) Capa totalmente conectada
10. (OUTPUT) Capa de salida (softmax)

LeNet (1998)

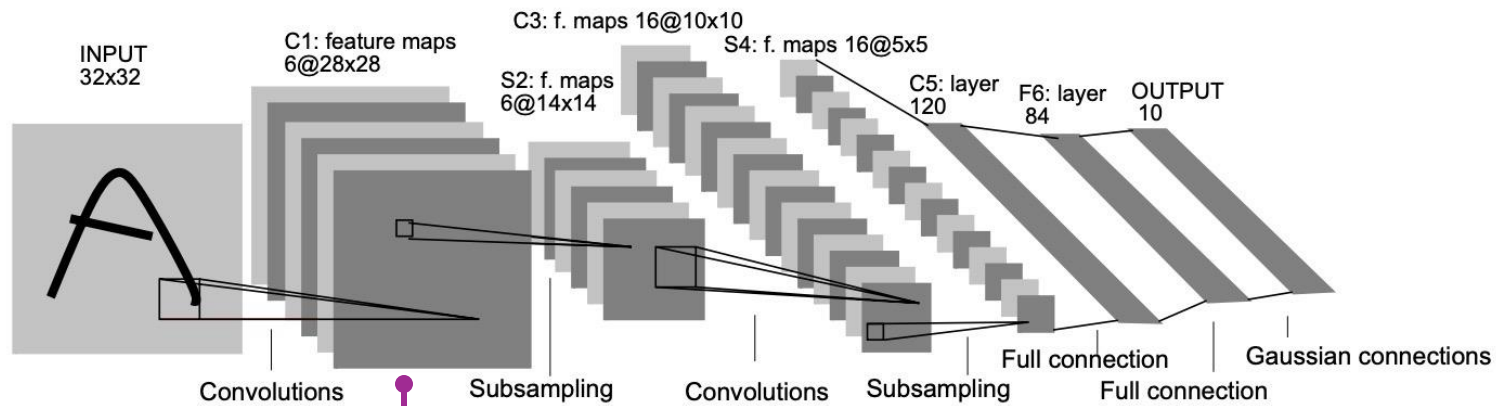


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Tiene 6 filtros y crea mapas de características de 28x28

¿Cuánto es el tamaño del kernel de agrupamiento en S2 para obtener el 14x14? Con paso = 2

1. (C1): Convolución
2. (S1): Activación (Sigmoid/Tanh/ReLU)
3. (S2): Agrupamiento promedio
4. (C3) Convolución
5. (S3): Activación (Sigmoid/Tanh/ReLU)
6. (S4) Agrupamiento promedio
7. (C5) Convolución
8. Activación (Sigmoid/Tanh/ReLU)
9. (F6) Capa totalmente conectada
10. (OUTPUT) Capa de salida (softmax)

LeNet (1998)

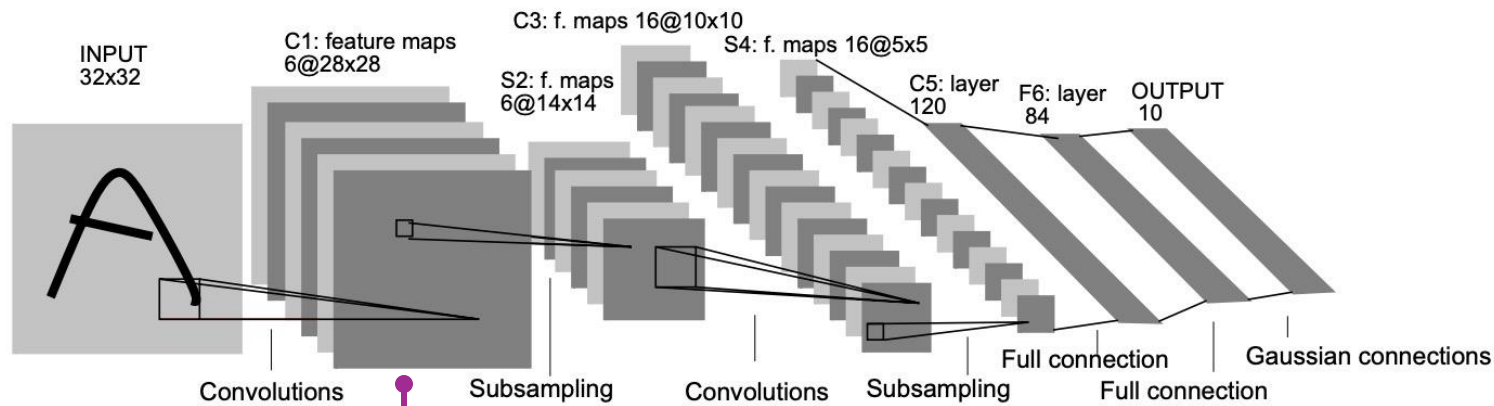


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

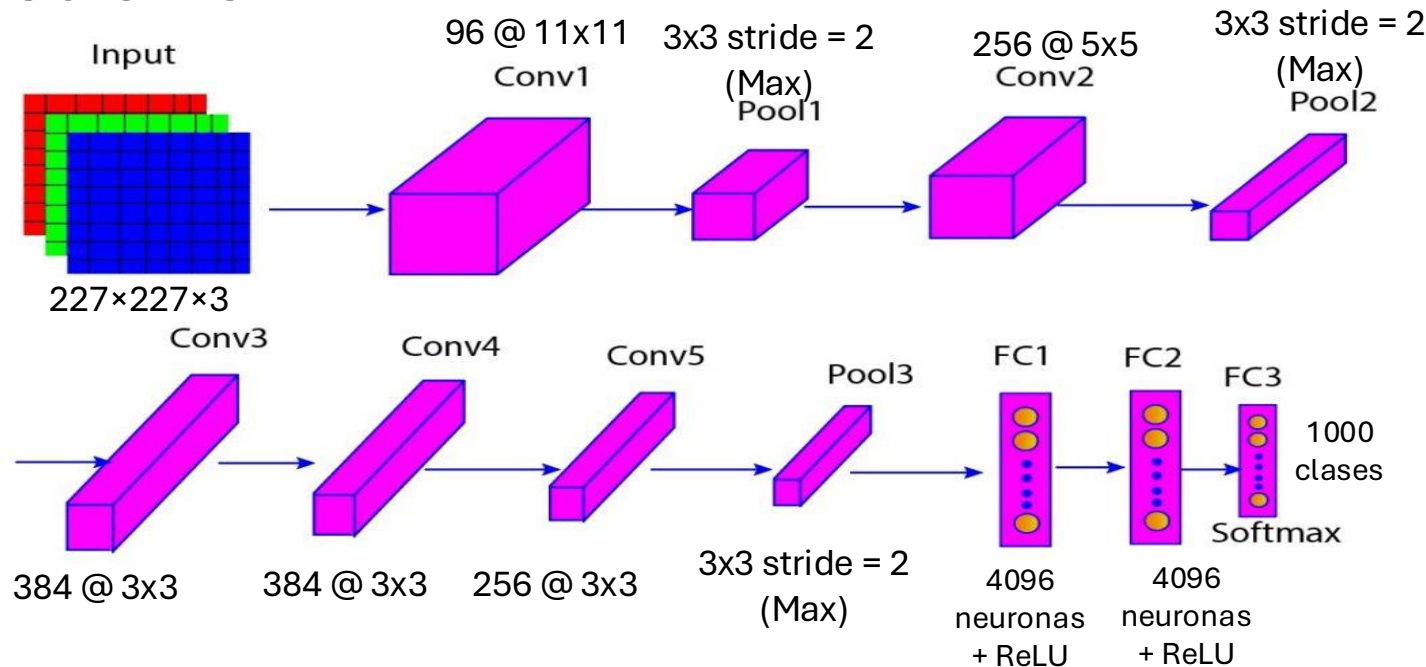
Tiene 6 filtros y crea mapas de características de 28x28

$$\begin{aligned} 14 &= (28-K)/2 + 1 \\ K &= 28 - ((14-1) * 2) \\ K &= 2 \rightarrow 2 \times 2 \end{aligned}$$

1. (C1): Convolución
2. (S1): Activación (Sigmoid/Tanh/ReLU)
3. (S2): Agrupamiento promedio
4. (C3) Convolución
5. (S3): Activación (Sigmoid/Tanh/ReLU)
6. (S4) Agrupamiento promedio
7. (C5) Convolución
8. Activación (Sigmoid/Tanh/ReLU)
9. (F6) Capa totalmente conectada
10. (OUTPUT) Capa de salida (softmax)

AlexNet (2012)

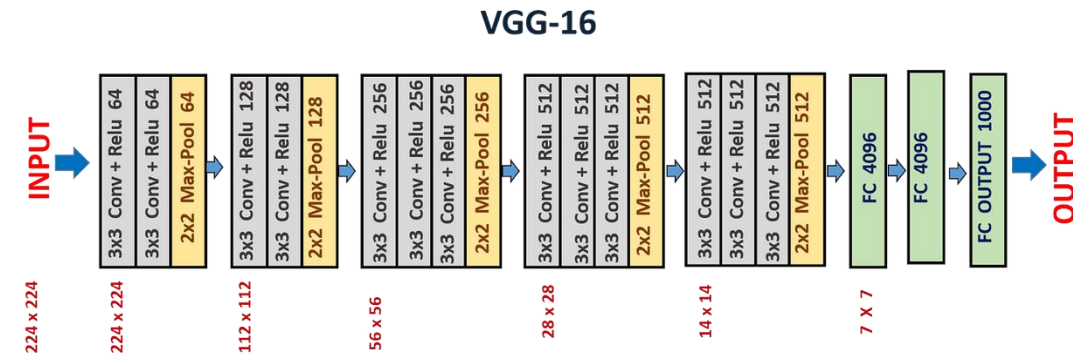
- Ganó el concurso de ImageNet por un gran margen ese año
- Diseñado con imágenes más grandes en mente
- Introdujo **ReLU** para mitigar el problema de gradiente desvaneciente



¿Por qué FC1 tiene 4096 neuronas?
El resultado antes de esta capa era de:
 $6 \times 6 \times 256$

VGG (2014)

- Propuesta en 2014 por el Visual Geometry Group (VGG) de la **Universidad de Oxford**, la arquitectura VGG, en particular **VGG16 y VGG19**, ganó popularidad por su **simplicidad y eficacia** en tareas de clasificación de imágenes.
- Las redes VGG son **significativamente más profundas** que los modelos anteriores, como AlexNet.
- El modelo VGG más popular, VGG16, tiene 16 capas (**13 capas convolucionales + 3 capas completamente conectadas**).
- A diferencia de las redes anteriores que utilizaban filtros de convolución más grandes (por ejemplo, 11x11 en AlexNet), VGG **utiliza filtros pequeños de 3x3 para todas las capas convolucionales**.
- Contiene: 13 convoluciones, 5 max pooling y 3 capas totalmente conectadas (incluyendo la de clasificación)



ResNet (2015)

- Introducido en 2015 por Kaiming He y sus colegas de **Microsoft Research**.
- ResNet (Red Residual) introdujo el concepto de aprendizaje residual a través de **conexiones salteadas**, lo que **permite el entrenamiento de redes mucho más profundas** (con cientos o incluso miles de capas) **sin sufrir el problema del gradiente de desaparición**.

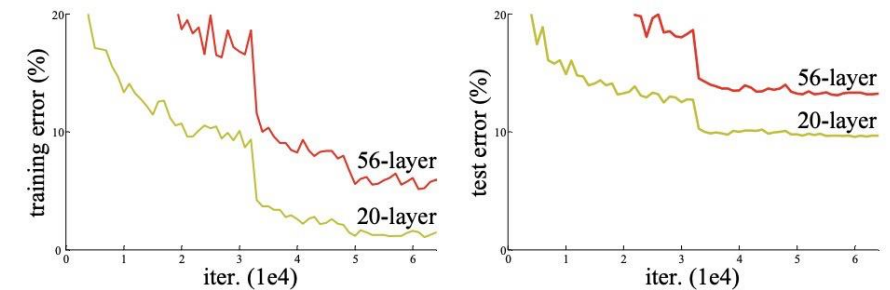
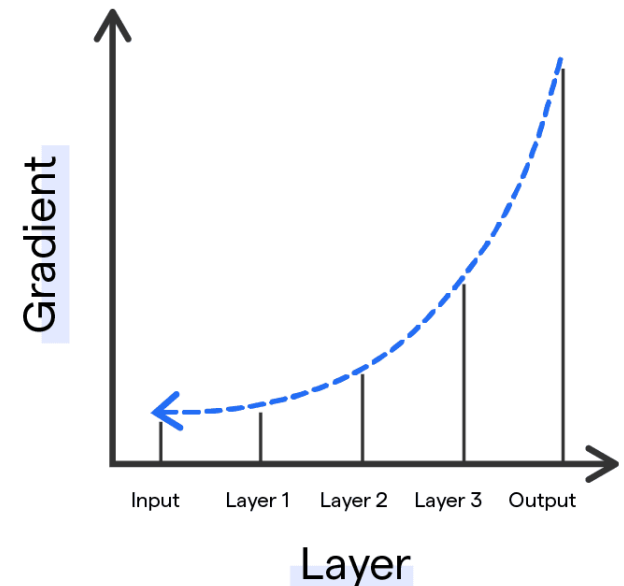


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

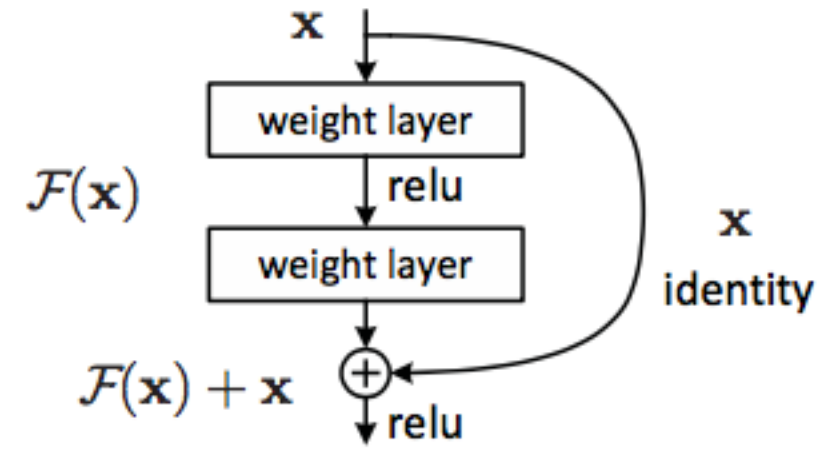
Desvanecimiento del gradiente

- El problema del desvanecimiento del gradiente es un problema común que se presenta al **entrenar redes neuronales profundas** (muchas capas).
- Se refiere a la situación en la que los **gradientes utilizados para actualizar los pesos durante la retropropagación se vuelven extremadamente pequeños** a medida que se propagan hacia atrás a través de la red.
- Esto da como resultado **un aprendizaje muy lento o incluso la incapacidad de la red para aprender**.
- Los gradientes se calculan mediante la **regla de la cadena**, multiplicando **muchas derivadas parciales** (una por cada capa).
- Las funciones de activación, como la sigmoide o la tanh, comprimen sus valores de entrada en rangos pequeños y sus **derivadas son pequeñas para la mayoría de los valores de entrada**. Por ende, generan este problema

Vanishing Gradient Problem

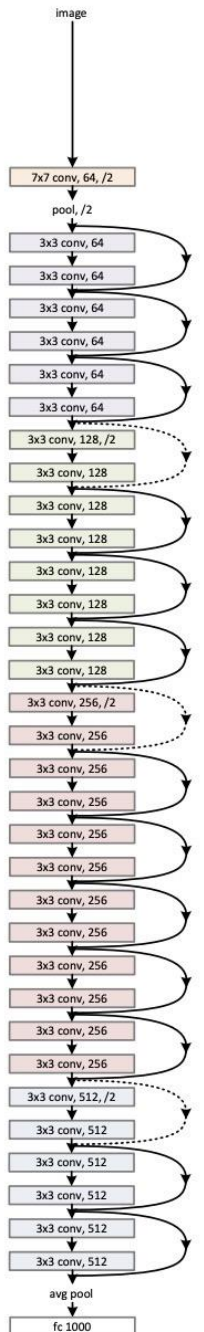


ResNet (2015)



- La arquitectura añade las **innovadoras conexiones residuales** (saltar conexiones). **Agregan conexiones de acceso directo que omiten una o más capas**
- La más utilizada es **ResNet-50**, pero hay versiones de 18, 32, 101 y 152 capas.
- ResNet resuelve el **problema de desvanecimiento** del gradiente al permitir que fluyan directamente a través de las conexiones de salto, lo que hace que el entrenamiento de redes profundas sea factible.
- Utiliza **Batch Normalization** (Normalización de lote):
 - A medida que avanza el entrenamiento, la distribución de las activaciones en las capas intermedias puede cambiar. Esto puede hacer que el entrenamiento sea lento e inestable porque cada capa debe adaptarse constantemente a nuevas distribuciones de entradas.
 - Durante el entrenamiento, para cada minilote de datos, las activaciones de una capa (para cada neurona) se normalizan para tener una media de 0 y una varianza de 1.
 - Esto ayuda a acelerar el proceso de entrenamiento, en algunos casos sirve como regularización y mejora el flujo del gradiente

34-layer residual



¿Por qué funcionan las conexiones residuales?

Optimización más sencilla: Las conexiones residuales permiten que la red **propague información de forma más directa a través de las capas**, lo que permite una optimización más sencilla de las redes profundas. **La red puede "omitir" el aprendizaje de ciertas capas si no son necesarias**, lo que reduce el riesgo de **sobreajuste** o de **quedarse atascada en mínimos locales incorrectos**.

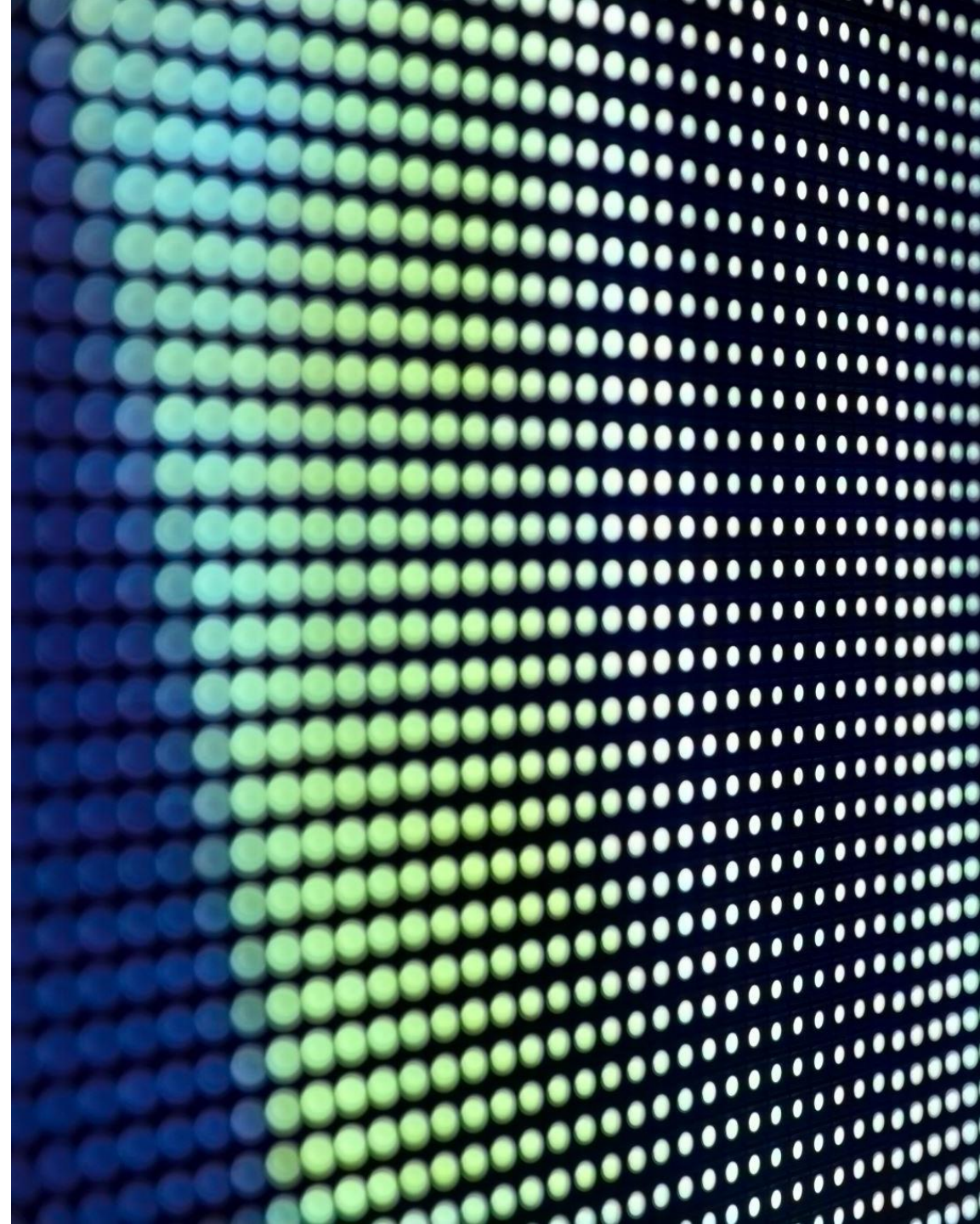
Flujo de gradiente: Las conexiones de omisión permiten que los gradientes fluyan directamente a través de la red durante la retropropagación, lo que **evita que desaparezcan a medida que pasan por muchas capas**.

Mapeo de identidad: Si una capa de la red no contribuye al resultado final (es decir, no es necesaria), las conexiones **residuales permiten que la red aprenda un mapeo de identidad (es decir, la capa no hace nada) sin penalizar el modelo**. Esto es crucial para entrenar redes muy profundas, ya que permite "omitir" capas innecesarias.

Comparación de los modelos vistos

Característica	LeNet (1998)	AlexNet (2012)	VGG (2014)	ResNet (2015)
Tipo de Arquitectura	CNN superficial (5 capas)	CNN profunda (8 capas)	CNN muy profunda (16 o 19 capas)	CNN muy profunda con conexiones residuales (hasta 152 capas)
Número de Capas	5 (Convolutacional + Totalmente Conectadas)	8 (5 Convolucionales + 3 Totalmente Conectadas)	16 (VGG16) / 19 (VGG19) (Convolutacional + FC)	18, 34, 50, 101, 152 capas (Convolutacional + FC)
Tamaño de Entrada	32x32x1 (escala de grises)	227x227x3 (RGB)	224x224x3 (RGB)	224x224x3 (RGB)
Tamaños de Filtro Convolutacional	5x5	11x11, 5x5, 3x3	3x3 en toda la red	3x3, 1x1 (en bloques de embotellamiento)
Max Pooling	Pooling 2x2	Pooling 3x3 (stride 2)	Pooling 2x2	Pooling 2x2
Función de Activación	Tanh	ReLU	ReLU	ReLU
Número de Parámetros	~60,000	~60 millones	VGG16: ~138 millones	ResNet-50: ~25.6 millones
Característica Única	Primer CNN para reconocimiento de dígitos (MNIST)	Red más profunda usando ReLU, entrenada en GPUs	Filtros pequeños de 3x3, red muy profunda	Aprendizaje residual a través de conexiones de salto
Error Top-5 en ImageNet	No probado en ImageNet	16.4% (ganador del concurso 2012)	7.3% (VGG16)	3.57% (ResNet-152) (ganador del concurso 2015)
Tiempo de Entrenamiento	Rápido (pequeño conjunto de datos)	Lento (requiere aceleración de GPU)	Muy lento (requiere poder computacional significativo)	Más rápido debido al aprendizaje residual
Conjunto de Datos	MNIST	ImageNet	ImageNet	ImageNet
Casos de Uso	Reconocimiento de dígitos (escritos a mano)	Clasificación de imágenes general, tareas de visión a gran escala	Clasificación de imágenes general	Clasificación de imágenes general, detección de objetos, segmentación
Desventajas	Muy simple, ineficaz para grandes conjuntos de datos	Alto costo computacional, gran uso de memoria	Número extremadamente grande de parámetros	Requiere una implementación más avanzada, aún alto costo computacional en versiones más profundas

Aumento de
datos

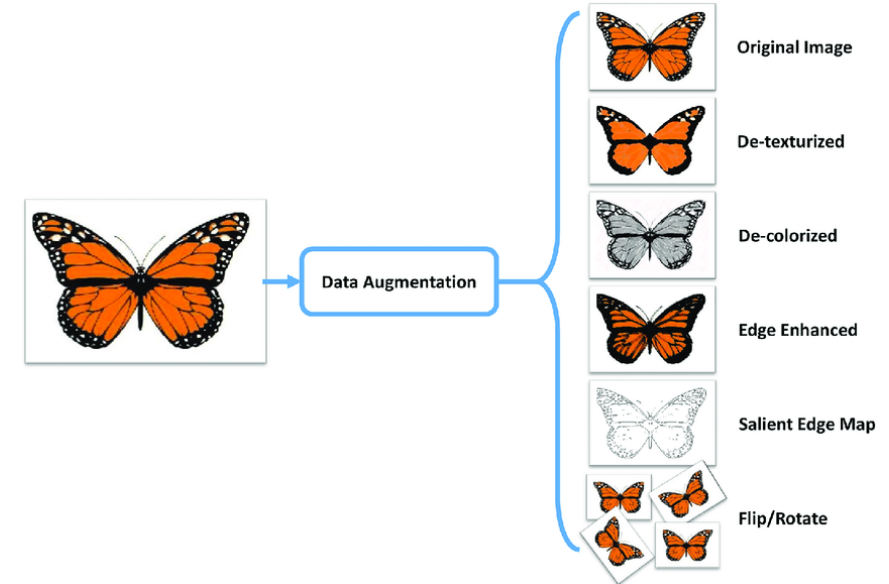


¿Qué hacer cuando no se tiene acceso a más datos y se necesita mayor diversidad de imágenes?

¿Qué opciones hay?

¿Qué es aumentación de datos en el contexto de Deep Learning aplicado a imágenes?

- Se refiere al proceso de **aumentar artificialmente el tamaño y la diversidad del conjunto de datos de entrenamiento** mediante la aplicación de **transformaciones aleatorias a las imágenes**.
- Estas transformaciones crean variaciones de las imágenes existentes **sin cambiar sus etiquetas o categorías**.
- Por ejemplo, al **rotar, voltear o agregar ruido ligeramente a una imagen**, creamos nuevas versiones que aún pertenecen a la misma clase.
- **Esto mejora la generalización de los modelos** de aprendizaje profundo al ayudarlos a aprender características más sólidas a partir de datos diversos, lo que **reduce el sobreajuste**.



¿Por qué es importante?



Mejora la generalización de los modelos: los modelos entrenados con datos aumentados tienden a generalizar mejor los datos no vistos. Esto **evita el sobreajuste**.



Amplía los conjuntos de datos pequeños: cuando no tenemos acceso a conjuntos de datos grandes y etiquetados, la ampliación puede ampliar artificialmente el tamaño del conjunto de datos.



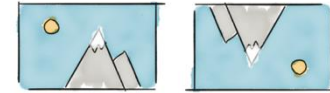
Mejora la robustez: la ampliación obliga a los modelos a ser **menos sensibles a orientaciones específicas**, condiciones de iluminación o distorsiones. Esto da como resultado modelos que son más robustos a las variaciones en los datos del mundo real.



Rentable: es una **forma económica** de mejorar el rendimiento sin necesidad de recopilar o etiquetar manualmente más datos.

Técnicas de aumentación populares

- **Voltear/Flip (horizontal/vertical):** reflejar la imagen a lo largo del eje vertical u horizontal.



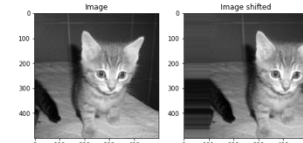
- **Rotar:** rotar la imagen en un grado aleatorio dentro de un rango especificado.



- **Acercar/Zoom:** acercar o alejar la imagen manteniendo la relación de aspecto.



- **Desplazar/Shift (traslación):** mover la imagen a lo largo del eje x o y.



- **Corte transversal/Shearing:** aplicar una transformación transversal, que sesga la imagen a lo largo de un eje.

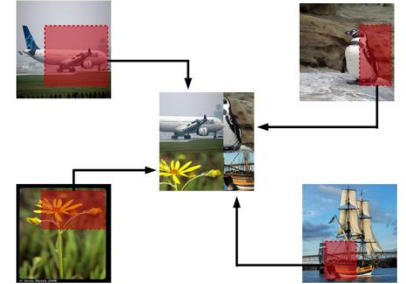


- **Ajuste de brillo:** modificar aleatoriamente el brillo de la imagen.

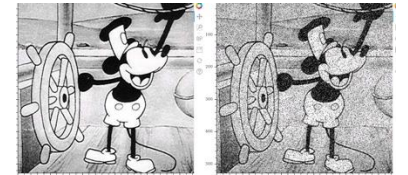


Técnicas de aumentación populares

- **Recortar/Cropping:** recortar aleatoriamente partes de la imagen.



- **Inyección de ruido:** agregar ruido aleatorio para simular datos de entrada granulosos o distorsionados.



- **Vibración de color/Jittering:** modificar aleatoriamente el tono, la saturación o el balance de color.



- **Recortar/Cutout:** enmascarar aleatoriamente una pequeña región cuadrada de la imagen para simular oclusión.



Consideraciones y mejores prácticas (1/2)



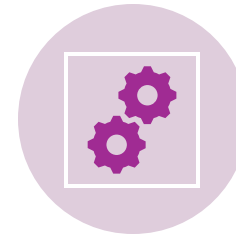
NO USARLO DE MÁS:
SOLO APLIQUE
IMITACIONES DE
VARIACIONES
REALISTASL. EL
AUMENTO EXCESIVO
PUEDE DISTORSIONAR
DEMASIADO SUS
DATOS



COMBINE TÉCNICAS:
USE DISTINTAS
TÉCNICAS DE
AUMENTO PARA
CREAR UNA VARIEDAD
DE IMÁGENES



**USE
PROCESAMIENTO
POR LOTES:** APLICAR
AUMENTOS SOBRE LA
MARCHA DURANTE EL
ENTRENAMIENTO
USANDO
GENERADORES POR
LOTES PARA EVITAR
SOBRECARGAR LA
MEMORIA



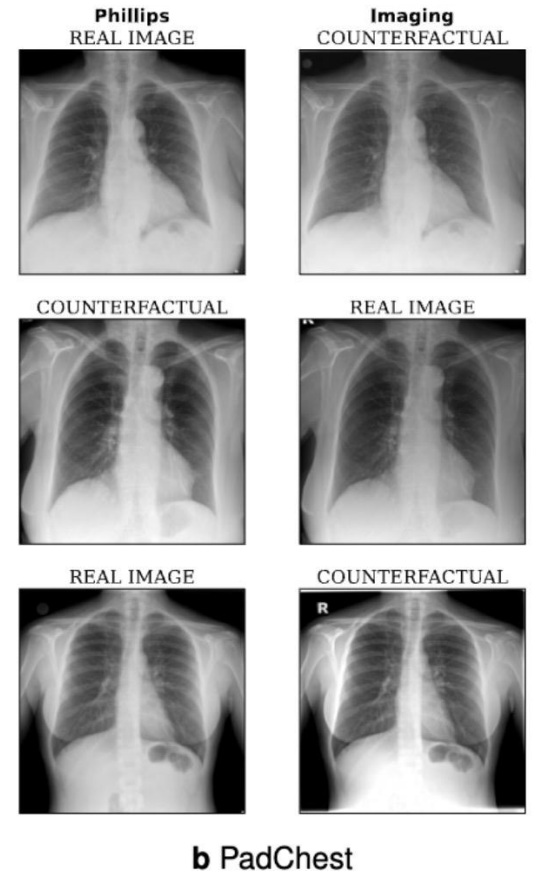
**COMENZAR CON
TRANSFORMACIONES
BÁSICAS**



**CHEQUEO DE LOS
RESULTADOS:**
VISUALICE ALGUNAS
DE LAS MUESTRAS
AUMENTADAS PARA
ASEGURARSE DE QUE
AÚN TENGAN SENTIDO
Y NO ESTÉN
DEMASIADO
DISTORSIONADAS.

Consideraciones y mejores prácticas (2/2)

- Solo se aplica al entrenamiento... ¿Por qué?
- No es mejor que añadir nuevas imágenes, pero es lo mejor que se puede hacer en ausencia de más observaciones
- Hay que equilibrar y no abusar el uso de esto (tasa de reales vs artificiales)
- Existen técnicas nuevas y más avanzadas de aumentar el número de imágenes dependiendo del contexto
- En caso de desbalance: tratar de balancear por medio de aumentación de la clase minoritaria



Roschewitz, M., Ribeiro, F. D. S., Xia, T., Khara, G., & Glocker, B. (2024). Robust image representations with counterfactual contrastive learning.

Ejemplo de código

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image

# Create an instance of ImageDataGenerator with some augmentations
datagen = ImageDataGenerator(
    rotation_range=40,    # Rotate images by up to 40 degrees
    width_shift_range=0.2, # Shift image horizontally by up to 20% of width
    height_shift_range=0.2, # Shift image vertically by up to 20% of height
    shear_range=0.2,     # Shear transformations
    zoom_range=0.2,      # Zoom in and out by up to 20%
    horizontal_flip=True, # Randomly flip the image horizontally
    fill_mode='nearest'  # Fill any missing pixels after transformations
)

# Load an example image (replace 'your_image_path' with your actual file path)
img_path = 'your_image_path.jpg'
img = image.load_img(img_path, target_size=(150, 150))
x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)

# Generate batches of augmented images
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0: # Display just 4 examples
        break
plt.show()
```

• **ImageDataGenerator:** esta clase se utiliza para aplicar ampliaciones de datos a lotes de imágenes. Puede ponerse varias y aleatoriamente se aplicarán.

- **Parámetros:** se aplican varias técnicas de ampliación, como rotación, desplazamiento de ancho y alto, corte, zoom y volteo

• **Método Flow:** el método .flow() genera imágenes ampliadas lote por lote

Este ejemplo muestra el aumento una sola imagen mediante varios métodos y los visualiza

Es más educativo, normalmente lo hacemos por lotes y solo visualizamos al principio, no todos los datos

Proyecto

+

•

○

Recordatorio sobre proyecto

Partes importantes que realizar:

- Preparación de datos para red neuronal:
 - División de datos en entrenamiento/validación/test
 - Ajuste de imágenes (tamaño)
 - Aumento de datos
- Construir al menos 3 redes neuronales con distintas arquitecturas para clasificar las imágenes:
 - MLP
 - CNN
- En cada modelo, al menos optimizar uno de los hiperparámetros (e.g intentar distintos tipos de optimizador o de valores de tasa de aprendizaje, etc)
- Medir el rendimiento en un set de datos separado (test)
- Visualizar para de interpretación de resultados en algunos casos (unos 3-5 casos con técnicas de interpretabilidad). Mostrar para el modelo con mayor rendimiento



Segmentación de imágenes

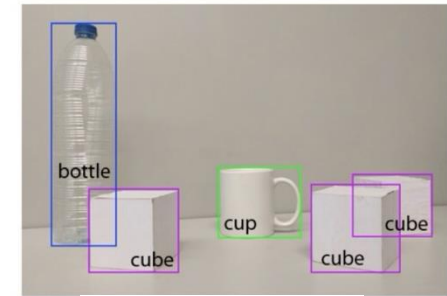


Segmentación de imágenes

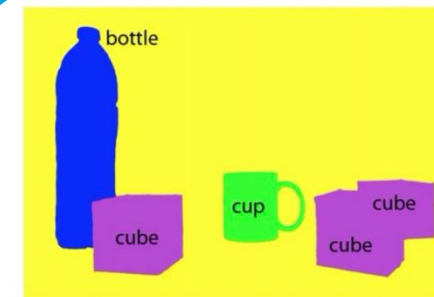
- Implica dividir una imagen en regiones o segmentos distintos, donde cada **segmento corresponde a un objeto o parte particular de la imagen**.
- A diferencia de la clasificación de imágenes tradicional, donde un modelo asigna una sola etiqueta a una imagen completa, **la segmentación asigna una etiqueta a cada píxel de la imagen**.
- Se tiende a utilizar **CNN** para identificar automáticamente los límites y las regiones de una imagen, lo que produce una predicción píxel por píxel que puede distinguir diferentes objetos, fondos o regiones.
- Los dos tipos más comunes:
 - **Semántica**: A cada píxel de la imagen se le asigna una etiqueta de clase, pero no hay distinción entre diferentes instancias de la misma clase.
 - **Instancia**: Identifica diferentes instancias del mismo objeto.



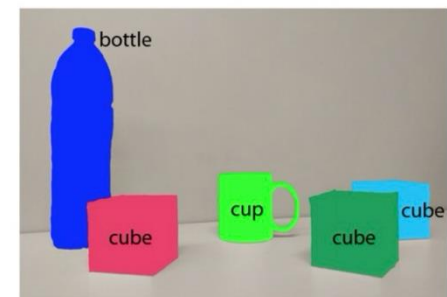
(a) **Clasificación**



(b) **Detección**



(c) **Segmentación semántica**

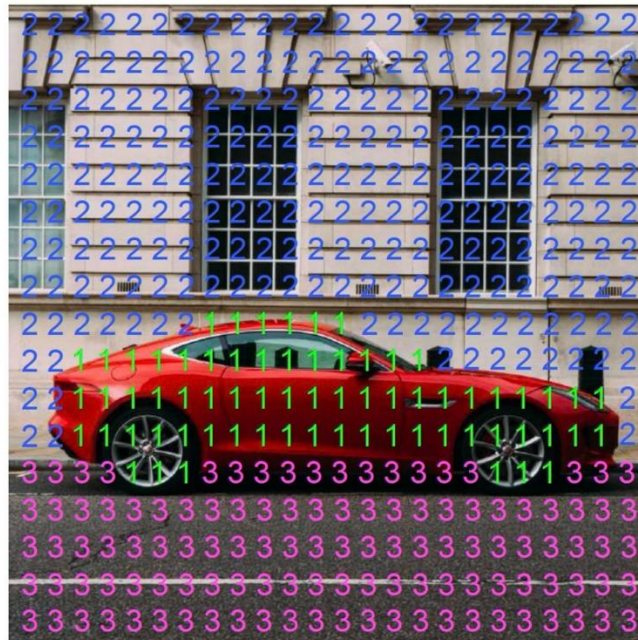


(d) **Segmentación de instancia**

Segmentación de imágenes

- A diferencia de la clasificación, que simplemente indica qué objetos están presentes, la segmentación proporciona la ubicación exacta y la forma de los objetos en la imagen, lo que la hace mucho más informativa para aplicaciones del mundo real.
- Algunas de las arquitecturas más usadas para segmentación:
 - U-Net
 - Mask R-CNN
 - SegNet
 - Transformers
- Es una herramienta esencial para industrias como la atención médica, la conducción autónoma y el análisis de imágenes satelitales.

¿Cuál es la variable respuesta en este caso?

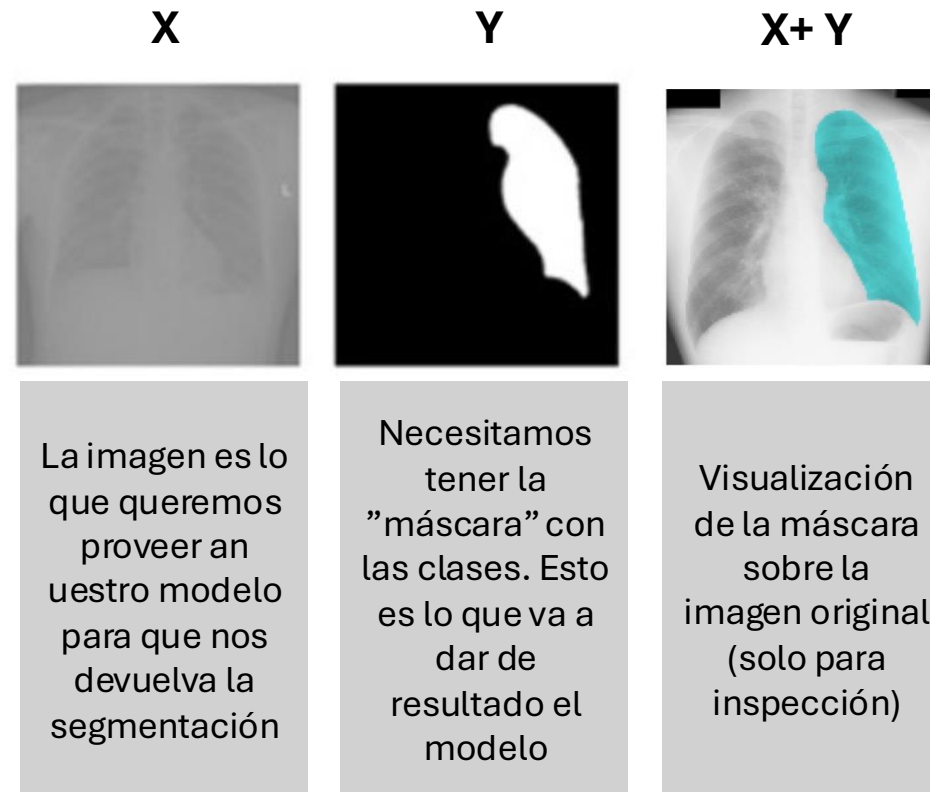


- 1 – Car
- 2 – Building
- 3 – Road



A segmentation map

Segmentación: Clasificación a nivel de pixel



- Puede ser una máscara binaria (0s y 1s) o multi-clase (cada pixel tiene un número de clase asignado)

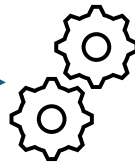
Clasificación a nivel de pixel

Datos de entrada



Entrenamiento

Red
Neuronal



Resultado (mapa
de clases
predicho)



Para cada pixel, busca la probabilidad más alta de cada clase y así lo clasifica



Mapa de clases
real

Para cálculo del
error y mejora de
la red

¿De qué tamaño es el mapa de clases?

DICE: Métricas de rendimiento y función de pérdida

DICE (DSC): **Medida de superposición entre dos conjuntos**, da más peso a la superposición entre los píxeles predichos y los de real al considerar tanto el tamaño de la intersección como el tamaño total de los conjuntos.

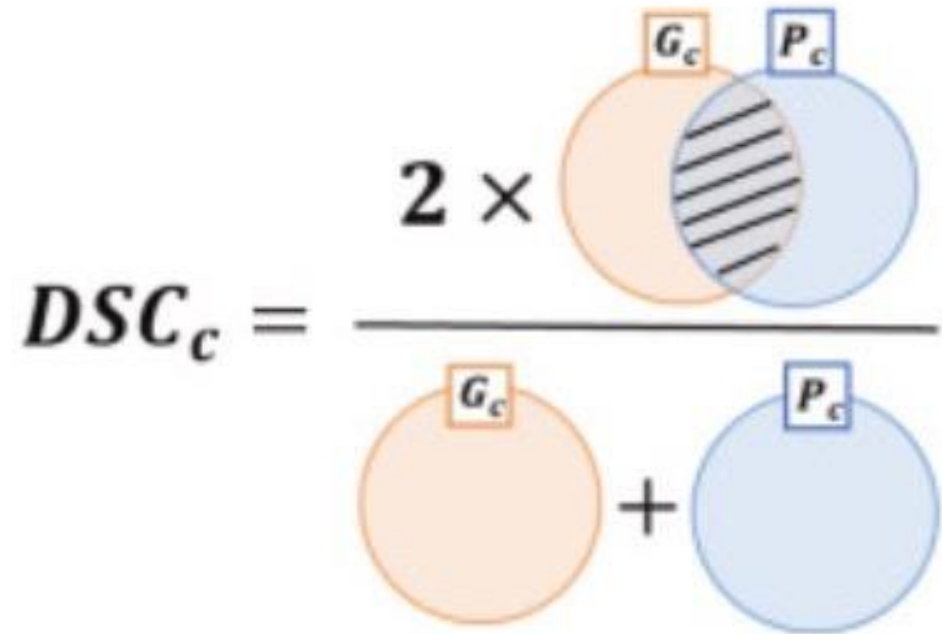
Un DICE de 1 es traslape perfecto (bueno) y de 0 nada de superposición (malo)

$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$

$|A|$ = es el número de píxeles en la segmentación predicha

$|B|$ = es el número de píxeles en la segmentación real

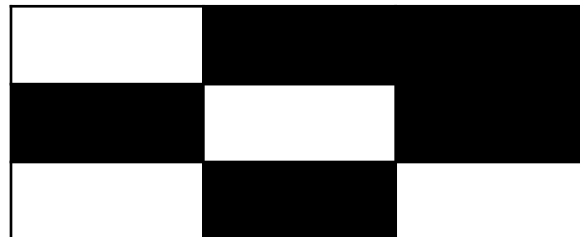
$|A \cap B|$ = es el número de píxeles en la intersección entre las regiones predichas y de la verdad fundamental.


$$DSC_c = \frac{2 \times \text{Intersection}(G_c, P_c)}{|G_c| + |P_c|}$$

Ejemplo de cálculo DICE – Mapa de etiquetas binario

Valor real

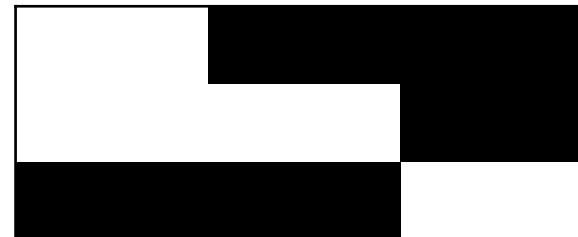
1	0	0
0	1	0
1	0	1



**Visto como
máscara/imagen**

Predicho

1	0	0
1	1	0
0	0	1



**Visto como
máscara/imagen**

Dice

$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$

$$DSC = \frac{2 * 3}{4 + 4} = \frac{6}{8} = 0.75$$

*Clase “0” no se cuenta como
acierto. Es llamada
“background”/“fondo”*

Ejemplo de cálculo DICE con varias clases

Valor real

0	1	1
0	2	0
2	0	1



**Visto como
máscara/imagen**

Predicho

0	1	0
2	2	0
2	0	1



**Visto como
máscara/imagen**

$$DSC_1 = \frac{2 * 2}{3 + 2} = \frac{4}{5} = 0.8$$


$$DSC_2 = \frac{2 * 2}{2 + 3} = \frac{4}{5} = 0.8$$

$$DSC_{media} = \frac{0.8 + 0.8}{2} = 0.8$$


Calculamos por separado cada clase y la luego agregamos con la media. También podría usarse ponderación si existe un criterio


Comparación de clasificación y segmentación

	Image Classification	Image Segmentation
Objetivo	Asignar una etiqueta a toda la imagen	Asignar una etiqueta para cada píxel en la imagen
Resultado	Probabilidades de cada clase o etiqueta predicha	Clasificación píxel por píxel (etiqueta o probabilidad para cada píxel)
Arquitectura	Típicamente CNNs: VGG, ResNet, etc	Codificador-decodificador: U-Net, FCN, DeepLab, etc
Función de pérdida	Entropía cruzada categórica	Entropía cruzada categórica o pérdida DICE



Arquitecturas para segmentar tiende a tener 3 componentes

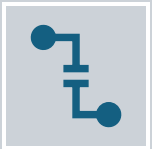


- **Codificador:** al igual que las redes de clasificación, extrae características a través de capas convolucionales y de agrupamiento.
 - **Decodificador:** aumenta el tamaño de las características hasta el tamaño de la imagen de entrada original, lo que crea predicciones a nivel de píxel.
 - **Conexiones de salto:** muchos modelos de segmentación (por ejemplo, U-Net) utilizan conexiones de salto para retener información de alta resolución al pasar directamente las características del codificador al decodificador.
- 

Modelo clásico U-net para segmentación semántica



Popular en la segmentación de imágenes biomédicas.

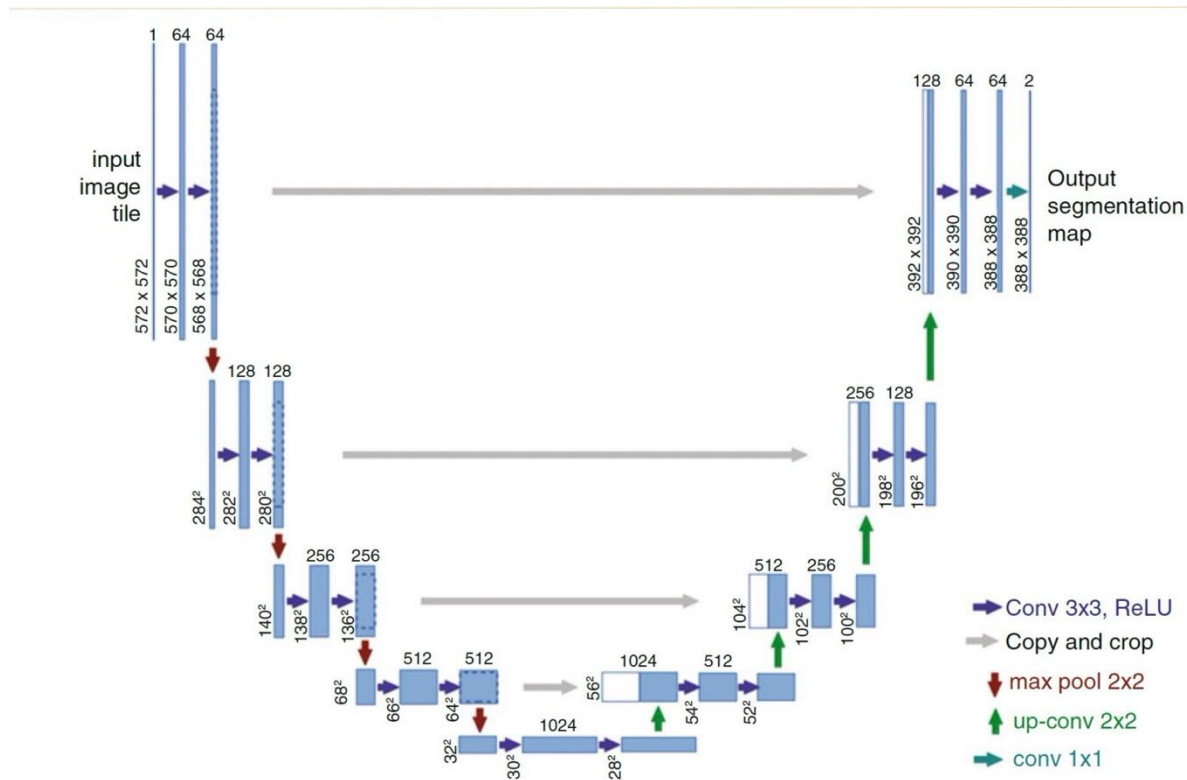


U-Net tiene una arquitectura simétrica con una estructura de codificador-decodificador, donde el codificador captura las características y el decodificador realiza un muestreo ascendente de la imagen para producir predicciones píxel por píxel.



Las conexiones de salto entre las capas correspondientes en el codificador y el decodificador ayudan a retener la información espacial.

Estructura de U-Net



U-Net architecture. Blue boxes are the feature maps. Channel numbers are denoted above each box, while the tensor sizes are denoted on the lower left. White boxes show the concatenations and arrows indicate various operations.

Ronneberger O, Fischer P, Brox T (2015) U-net: convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. Springer, Berlin,

Ventajas y desventajas de U-net

Ventajas

Flexible y utilizable con la mayoría de tareas que tengan máscaras

Da buen rendimiento en general (al menos de base)

Poderosa en escenarios con limitado número de datos

Desventajas

Imágenes muy grandes necesitan alta memoria de GPU

Toma tiempo significativo de entrenamiento

No hay muchos modelos pre-entrenados con esta arquitectura