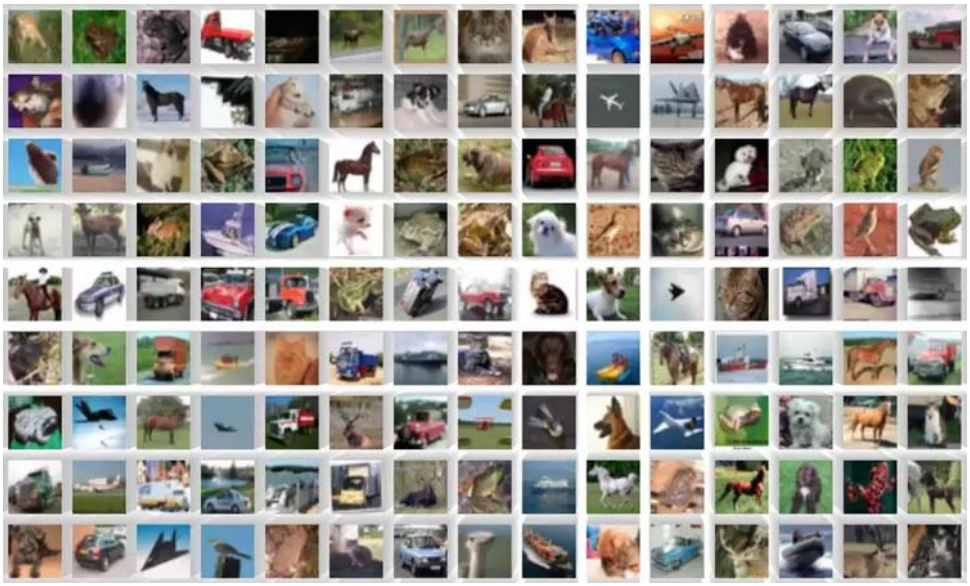
El Desafío Deep Visión

Realizado por Sergio Rosell

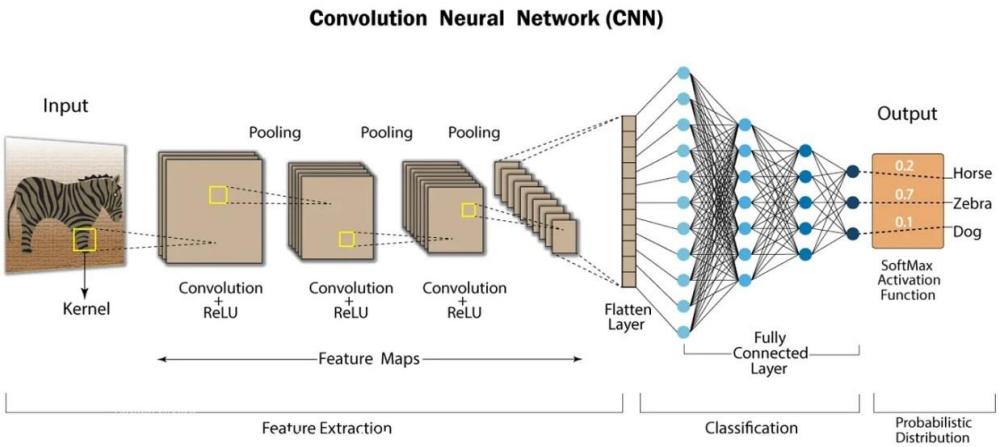
- ## *Introducción*

  En esta nueva misión nuestro objetico será explorar el conjunto de datos CIFAR-10. Se trata de 60.000 imágenes en color de baja resolución (32x32x3) clasificadas en 10 clases (avión, coche, animales…). Utilizaremos una Red CNN (Red Neuronal Convolucional) para que pueda distinguir a que grupo pertenece la imagen.



Utilizamos una red neuronal CNN ya que una red densa fallaría al aplanar la imagen y por tanto perdiendo toda la estructura espacial. Un pixel en una esquina no tiene relación con otro pixel vecino.

En cambio, en nuestra red CNN usa filtros (convoluciones) que actuaran como “detectores de rasgos” (bordes, texturas) sobre pequeñas áreas locales. Esto logra la invariancia traslacional, respeta la naturaleza espacial de las imágenes y es mucho más eficiente



- ***Convolución sin magia***

  Un filtro (o kernel) en una capa “Conv2D”, se trata de una pequeña matriz de pesos (que son los parámetros entrenables) que se desliza sobre la imagen de entrada para crear un “mapa de características”, a esto se le llama feature map.

  - **Tamaño (Kernel Size):** Dimensiones del filtro (ej. 3×3, 5×5). Un filtro 3×3 mira un vecindario de 9 píxeles a la vez.

  - **Stride (Paso):** Cuántos píxeles se mueve el filtro en cada salto. Stride=1 (lo normal) mueve el filtro de un píxel en un píxel. Stride=2 salta de dos en dos (y reduce la imagen a la mitad).

  - **Padding (Relleno):** Añadir ceros al borde de la imagen de entrada. Padding='valid' (sin relleno) hace que la salida sea más pequeña. Padding='same' añade el relleno justo para que la imagen de salida tenga el mismo tamaño que la entrada (si stride=1).

  - **Canales (Channels):** El filtro debe tener la misma profundidad (canales) que la entrada. Si la entrada es 32×32×3 (RGB), el filtro será, por ejemplo, 3×3×3. Si la capa tiene 32 filtros, aprenderá 32 rasgos diferentes.

  Por ejemplo: (1 canal, Input 5×5, Kernel 3×3, Stride 1, Padding 0)

  Input (5x5)

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  Kernel (3x3)

1	0	0
0	1	0
0	0	1

  La **salida** (tras deslizar el filtro por todas las posiciones) será una matriz de **3×3**.

  El coste Computacional no es el cálculo, sino los parámetros a entrenar. Este kernel (3×3) tiene 9 pesos + 1 bias = **10 parámetros**. Si la entrada tuviera 3 canales (RGB), el kernel sería 3×3×3, teniendo (9*3) + 1 = 28 parámetros.

- ***La importancia del pooling***

  MaxPooling y **AveragePooling** son técnicas de submuestreo (downsampling) que reducen el tamaño de los mapas de características. Ambas toman una ventana (ej. 2×2) y la colapsan a un solo valor, pero lo hacen de forma diferente:

- **MaxPooling:** Conserva solo el **valor máximo** de la ventana.
  Tiende a conservar las texturas y bordes más nítidos.

- **AveragePooling:** Conserva el **promedio** de todos los valores de la ventana.
  Tiene un efecto suavizante.

  Ambas pierden la información de la localización *exacta* del rasgo dentro de la ventana (lo que ayuda a la invariancia traslacional) y también reducen drásticamente el número de parámetros en las siguientes capas, lo cual es un potente regularizador que ayuda a combatir el sobreajuste.

- ***Métrica y pérdida adecuadas***

  Para una clasificación multiclase (10 clases) como CIFAR-10, donde las etiquetas están en formato *one-hot* (ej. [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] para "pájaro"):

1. **Función de Pérdida (Loss Function): Categorical Crossentropy** (Entropía Cruzada Categórica). Esta función mide la "distancia" entre la distribución de probabilidad que predice el modelo (softmax) y la distribución real (la etiqueta one-hot). Es la función estándar para optimizar este tipo de problemas.

2. **Accuracy** (Exactitud).
   Mide el porcentaje de imágenes que el modelo ha clasificado correctamente. Se calcula como (Aciertos Totales) / (Imágenes Totales).
   La *Exactitud* te dice *cuántas* acierta en general, pero no dice *dónde* falla. Una *exactitud* del 90% puede parecer buena, pero una **matriz de confusión** podría revelar que, aunque acierta el 99% de "coches" y "aviones", falla el 100% de las veces con los "gatos", confundiéndolos siempre con "perros". La matriz muestra exactamente qué clases se están confundiendo entre sí (ej. "Valor Real: Gato, Valor Predicho: Perro"), lo cual es vital para entender los puntos ciegos de tu modelo.

- ## *Normalización y preparación de los datos*

  Dividir por **255.0** (float) es la **normalización** más simple: escala los píxeles del rango de enteros [0, 255] al rango de flotantes [0.0, 1.0]. Esto es muy importante porque la mayoría de inicializadores de pesos y funciones de activación (como ReLU o Sigmoide) esperan entradas pequeñas y centradas cerca de cero. Si usáramos [0, 255], los gradientes explotarían en la primera pasada (*exploding gradients*), requiriendo un proceso de aprendizaje diminuto e inestable.

  La **estandarización** (restar la media y dividir por la desviación estándar *por canal*) es más robusta. Se usa cuando la iluminación o el color varían mucho entre canales (ej. la media del canal rojo en CIFAR-10 es diferente a la del azul). Esta técnica centra los datos exactamente en 0 (media=0, std=1).

  Ambas técnicas mejoran drásticamente la **estabilidad del entrenamiento** y la velocidad de convergencia. Permiten usar una curva de aprendizaje inicial más alta y saludable, ya que el paisaje de la función de pérdida se vuelve mucho más suave.

- ## *Baseline denso vs CNN*

  Una MLP (red densa) que recibe la imagen aplanada (vector de 3072 entradas: 32×32×3) es una arquitectura pésima para visión por dos razones clave.

  Primero, el **número de parámetros** explota: una sola capa oculta de 512 neuronas tendría 1.58 millones de parámetros (3072×512 + 512), llevando a un sobreajuste masivo e inmediato.

  Segundo, y más importante, carece del **"sesgo inductivo"** correcto. Al aplanar, destruye la topología 2D y la relación espacial entre píxeles. Una CNN, en cambio, tiene el sesgo inductivo espacial perfecto: asume **localidad** (los filtros 3×3 miran vecindarios) y **compartición de parámetros** (el mismo filtro detector de bordes se usa en toda la imagen), reduciendo drásticamente los parámetros. Además, el *pooling* le da **invariancia traslacional**.

  Por tanto, El MLP es extremadamente sensible al **ruido de fondo** o a pequeñas traslaciones (un coche movido 2 píxeles es una entrada *totalmente* diferente para él), mientras la CNN generaliza.

- # *Parámetros y capacidad*

  El número de parámetros en una capa Conv2D **no depende del tamaño de la imagen de entrada**, solo de sus filtros.
  Ejemplo: Si la entrada es 32×32×**3** y la capa Conv2D tiene **64** filtros de **5×5**:
  Parámetros = (5 x 5 x 3) x 64 + 64 = (75) x 64 + 64 = 4800 + 64 = 4864

  Impacto en la Capacidad:

- **Tamaño del Kernel:** Kernels más grandes (7×7) aumentan los parámetros (capacidad) pero suelen ser menos eficientes que apilar kernels pequeños (3×3).
- **Canales de Salida (Nº Filtros):** Es la forma principal de aumentar la **capacidad** de la red. Más filtros (de 32 a 64) significa que la capa puede aprender más rasgos distintos.
- **Profundidad (Más capas):** Añadir capas aumenta la capacidad de forma no lineal (permite jerarquías de rasgos: bordes -> formas -> objetos).
  Nos permite aumentar cualquiera de estos tres (kernels, filtros o profundidad) incrementa la **capacidad** (puede aprender patrones más complejos), pero también aumenta el **tiempo de entrenamiento** y eleva drásticamente el **riesgo de sobreajuste**.

- # *Regularización practica*

  Las cuatro técnicas clave para controlar el sobreajuste en CIFAR-10 son:

1. **Data Augmentation (Aumento de Datos):** La más efectiva. Crea variaciones sintéticas de las imágenes de entrenamiento (rotaciones, *flips*, *zooms*). Evita que el modelo memorice las imágenes exactas de *entrenamiento* y le enseña qué rasgos son invariantes.
2. **Dropout:** Se aplica usualmente en las capas densas (tras Flatten). Durante el entrenamiento, "apaga" aleatoriamente un porcentaje (ej. 30-50%) de las neuronas. Esto fuerza a la red a aprender representaciones redundantes y evita que unas pocas neuronas dominen la predicción.
3. **L2 Weight Decay (Regularización L2):** Añade una penalización a la función de pérdida proporcional al cuadrado del valor de los pesos. Esto "desincentiva" pesos muy grandes, favoreciendo modelos "más simples" y suaves que generalizan mejor. (Optimizándolo con AdamW es la forma moderna de aplicarlo).
4. **Early Stopping (Parada Temprana):** Monitoriza una métrica en el conjunto de validación. Si esta métrica deja de mejorar (o empieza a empeorar) durante un número determinado de épocas , el entrenamiento se detiene automáticamente, guardando el modelo en su mejor punto.

Una estrategia robusta usa **Data Augmentation** y **L2** (AdamW) como base. Se añade **Dropout** en las capas densas. **Early Stopping** se usa siempre como red de seguridad.

- # *Data Augmentation con cabeza*

Un plan de aumentos (Data Augmentation) robusto y *razonable* para CIFAR-10 debe simular variaciones realistas sin destruir la semántica de la clase.

Una estrategia muy plausible seria:

1. **RandomHorizontalFlip (p=0.5):** El aumento más importante. Un coche mirando a la izquierda o derecha sigue siendo un coche.
2. **RandomRotation (ej. 10-15 grados):** Rotaciones pequeñas. Un pájaro ligeramente inclinado es válido.
3. **RandomTranslation (ej. 10% de alto/ancho):** Traslaciones ligeras (ej. 3-4 píxeles). El objeto no siempre está perfectamente centrado.
4. **ColorJitter (leve):** Pequeñas variaciones de brillo, contraste y saturación. Simula diferentes condiciones de iluminación.
   Justificación de los límites: Los aumentos deben ser conservadores porque las imágenes son muy pequeñas (32×32). Una traslación grande (ej. 50%) podría sacar al objeto de la imagen.

Aumentos que no mejorarían el modelo:

- **RandomVerticalFlip (Flip Vertical):** Es el peor error en CIFAR-10. Generaría coches, camiones y barcos "boca abajo", algo que no existe en el mundo real (o en *test*) y confundiría totalmente al modelo (un ciervo o un gato boca abajo no es una imagen válida).
- **Rotaciones de 90 o 180 grados:** Por la misma razón.

- # *Optimizacion y Curva de aprendizaje*

SGD con Momentum es el optimizador clásico. Usa el *momentum* para suavizar la trayectoria del descenso (acumula gradientes pasados), ayudando a escapar de mínimos locales y acelerando en la dirección correcta. Es robusto y suele generalizar muy bien, pero es muy sensible a la elección de la curva de aprendizaje inicial y requiere un *scheduler* bien afinado.

Adam (o AdamW) es adaptativo. Mantiene una curva de aprendizaje individual para cada parámetro, ajustándolo según el historial (momentum y RMSProp). Converge mucho más rápido que SGD y es menos sensible a la curva de aprendizaje inicial. Es la opción *por defecto* más segura y recomendable para empezar.

Scheduler Propuesto: ReduceLROnPlateau

Este *scheduler* monitoriza una métrica (idealmente val_loss). Si val_loss deja de mejorar (se "estanca" en una meseta o *plateau*) durante X iteraciones ("paciencia"), el *scheduler* asume que el modelo está atascado y **reduce la curva de aprendizaje** (ej. multiplicándolo por 0.1 o 0.2). Esta es la **señal clave**: un estancamiento en la pérdida de validación significa que el modelo ya no puede aprender con "pasos" tan grandes.

- # *Curvas de aprendizaje*

Las curvas de entrenamiento (sobre datos de *train*) y validación (sobre datos apartados) son el "electrocardiograma" de nuestro modelo.

Patrones Típicos:

1. **Subajuste (Underfitting):** Ambas curvas (Train y Valid) se estancan en un mal valor (Loss alta / Accuracy baja). No hay brecha entre ellas. El modelo es incapaz de aprender, incluso de los datos de *train*.
    - **Ajuste:** Aumentar la capacidad del modelo (más capas, más filtros), usar una arquitectura más potente, o entrenar más tiempo.

2. **Ajuste Saludable (Good Fit):** Ambas curvas de *loss* bajan de forma suave. La val_loss baja y se estabiliza en un mínimo, manteniéndose cerca de la train_loss. Las curvas de *exactitud* suben y se estabilizan alto.
    - **Ajuste:** Modelo perfecto

3. **Sobreajuste (Overfitting):** La train_loss sigue bajando (el modelo memoriza), pero val_loss empieza a subir. Se abre una brecha significativa entre ambas. El modelo generaliza mal.
    - **Ajuste:** Aumentar la regularización. Añadir/subir **Dropout**, aplicar **Data Augmentation** más agresiva, aumentar el **Weight Decay (L2)**, o usar **Early Stopping** para parar antes de que val_loss suba.

- # *Matriz de confusión y clase difícil*

La matriz de confusión (10×10 para CIFAR-10) muestra el desglose de los errores. El **Eje Y** representa la **Clase Real** (lo que era) y el **Eje X** la **Clase Predicha** (lo que el modelo dijo).
La **diagonal principal** (de arriba-izquierda a abajo-derecha) son los **aciertos**: la celda (Gato, Gato) muestra cuántos gatos se clasificaron correctamente como gatos. Los **valores fuera de la diagonal** son los **errores**.

Pares propensos a confusión en CIFAR-10:

1. **Gato vs Perro:** El par más difícil. Ambos son mamíferos peludos de cuatro patas. En 32x32, distinguir un hocico de gato de uno de perro es muy complejo.
2. **Camión vs Coche:** Ambos son vehículos rodados, visualmente similares en baja resolución.
3. **Pájaro vs Avión:** Ambos suelen aparecer contra un fondo de cielo.

Propuesta de Mejora: Si Gato y Perro es el peor par, podemos aplicar **Data Augmentation específica**. Por ejemplo, usar Cutout (borrar parches aleatorios) fuerza al modelo a no fijarse solo en "pelaje marrón", sino a buscar rasgos distintivos como la forma de las orejas o el hocico, mejorando la distinción. Si el modelo es muy simple, **aumentar la profundidad** (más bloques Conv/Pool) también ayudaría.

- ## *Batch Size y estabilidad*

El *batch size* (tamaño del lote) define cuántas imágenes procesa el modelo antes de actualizar sus pesos.

Impacto (32 vs 64 vs 128):

- **Tiempo de Época:** Un *batch size* **grande** (128) utiliza mejor el paralelismo de la GPU y requiere menos actualizaciones de pesos por época. Por tanto, el **tiempo por época es menor**. Un *batch size* pequeño (32) es más lento.

- **Estimación de Gradiente:** Uno **grande** (128) calcula un gradiente más "preciso" (el promedio de 128 ejemplos), resultando en un descenso más estable. Uno **pequeño** (32) produce un gradiente "ruidoso".

- **Generalización:** El "ruido" de los *batch sizes* **pequeños** (32, 64) actúa como un **regularizador**, ayudando al modelo a encontrar mínimos "planos" que generalizan mejor. *Batch sizes* muy grandes (128+) tienden a converger a mínimos "afilados" (memorización) y pueden generalizar peor.

Valor Inicial (Colab): Un *batch size* de **64**.
Es mejor porque 32 es muy seguro para la generalización, pero puede ser lento. 128 es rápido, pero consume más RAM de GPU (riesgo de OOM - *Out of Memory* - en Colab) y puede generalizar peor. **64** es el equilibrio estándar: suficientemente rápido para aprovechar la GPU de Colab y suficientemente "ruidoso" para ayudar a la generalización.

- ***Practicas correctas en esta entrega***

  La entrega de la práctica debe ser reproducible, clara y analítica:

1. **Fijado de Semillas (Seeds):** Asegurar la reproducibilidad total del experimento (np.random.seed, tf.random.set_seed).
2. **Código limpio y modular:** Usar funciones claras y añadir comentarios.
3. **Análisis del Baseline:** Incluir la tabla comparativa (Params, Train Acc, Val Acc) del **MLP Denso vs. la CNN Base**.
4. **Arquitectura Final:** Un resumen (model.summary()) y una breve explicación de la arquitectura CNN final elegida.
5. **Logs de Entrenamiento:** La salida/log de la celda del model.fit() final, mostrando la mejora por iteración (loss/accuracy).
6. **Curvas de Aprendizaje:** Gráficos de (Train/Valid Loss) y (Train/Valid Accuracy) del modelo final.
7. **Matriz de Confusión:** La matriz generada sobre el conjunto de **test** (datos nunca vistos).
8. **Tabla de Hiperparámetros:** Resumen (LR usado, optimizador, *batch size*, regularización L2, Dropout %).
9. **Análisis de Errores (Bonus):** Mostrar 3-5 imágenes que el modelo clasificó *mal* y analizar por qué pudo fallar.
10. **Conclusiones y Hallazgos:** 5 puntos clave sobre qué funcionó (ej. "Data Augmentation fue crucial"), qué no funcionó y por qué la CNN superó al MLP (citando el sesgo inductivo).