# REDES Neuronales Convolucionales CNN

Una **Red Neuronal Convolucional (CNN)** es un tipo especial de red neuronal diseñada específicamente para trabajar con datos que tienen una estructura espacial o temporal, como imágenes o secuencias.

En lugar de conectar todas las neuronas entre capas como en una ANN (Fully Connected Network), una CNN utiliza **convoluciones** para analizar partes específicas de los datos de entrada (por ejemplo, regiones de una imagen).

## ¿Qué es una CNN?

**Capa de Convolución (Convolutional Layer)**: Aplica filtros (kernels) sobre la imagen para detectar patrones como bordes y formas.

**Función de Activación**: Decide si un patrón detectado es importante para avanzar en el proceso.

**Capa de Agrupación (Pooling Layer)**: Reduce el tamaño de la información para hacerla más manejable.

**Capa Totalmente Conectada (Fully Connected Layer)**: Toma las partes importantes de la imagen y decide la categoría final (por ejemplo, "perro" o "gato").

**Regularización y Dropout**: Ayuda a evitar que la red se sobreajuste a los datos apagando aleatoriamente algunas neuronas durante el entrenamiento.

**Capa de Salida**: Proporciona la decisión final, como "perro" o "gato".

## 

## 1.Capas convolucionales y de agrupación

**- \*\*Convolucionales:\*\*** Estas capas aplican filtros convolucionales a las entradas.Las convoluciones **son eficaces para extraer características locales, como bordes, texturas y patrones pequeños, y permiten que la red aprenda representaciones jerárquicas de las características.**

**- \*\*De Agrupación (Pooling):\*\* Estas capas reducen la dimensionalidad de las características extraídas por las capas convolucionales.** Operan deslizando una ventana sobre la salida de la capa convolucional y **tomando el valor máximo (o promedio) dentro de esa ventana.**

**Esto reduce el número de parámetros en la red y ayuda a controlar el sobreajuste,** al tiempo que mantiene las características más importantes de la imagen.

### # Definir la arquitectura de la CNN

**La línea de código model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1))) agrega una capa convolucional a un modelo secuencial en Keras. Explicación breve:**

* **model.add: Añade una nueva capa al modelo.**
* **Conv2D: Especifica una capa convolucional.**
* **32: Número de filtros (detectan diferentes características).**
* **(3, 3): Tamaño de cada filtro (3x3).**
* **activation='relu': Función de activación ReLU (convierte los valores negativos a 0).**
* **input\_shape=(28, 28, 1): Entrada de imágenes de 28x28 píxeles en escala de grises (1 canal).**

## 2. FUNCION ACTIVACIÓN

**Cuando aplicamos una función de activación después de la convolución, añadimos no linealidad al modelo. Esto permite que la red:**

1. **Aprenda patrones más complejos en los datos, como curvas o combinaciones de características.**
2. **Resuelva problemas difíciles que requieren relaciones no triviales.**

**ReLU (Rectified Linear Unit)**: La función más común en redes neuronales. Convierte todos los valores negativos en cero y deja los valores positivos sin cambios. Ayuda a mejorar la eficiencia del aprendizaje y mitiga el problema del desvanecimiento del gradiente.

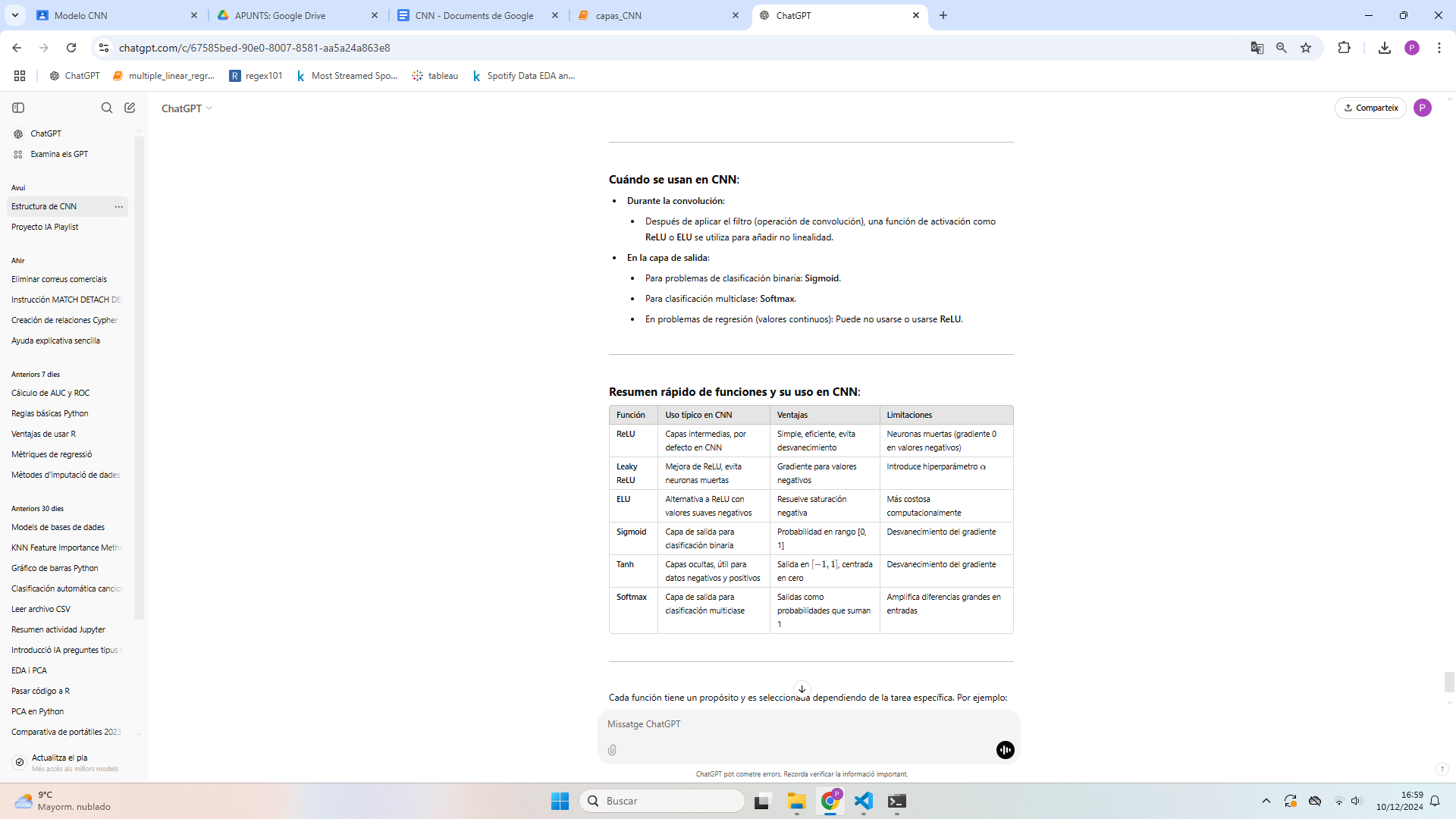
**Leaky ReLU**: Variante de ReLU que permite un pequeño gradiente para valores negativos (en lugar de cero), lo que ayuda a evitar el problema de neuronas "muertas", donde las neuronas no aprenden.

**ELU (Exponential Linear Unit)**: Similar a ReLU, pero utiliza una función exponencial para los valores negativos, lo que permite una transición más suave en esa zona y mejora el rendimiento en algunas redes profundas.

**Sigmoide**: Devuelve un valor entre 0 y 1, ideal para problemas de clasificación binaria, como "gato" o "no gato". Se utiliza comúnmente en la capa de salida.

**Tanh (Tangente Hiperbólica)**: Similar a la sigmoide, pero su rango de salida es de -1 a 1. Es útil en capas ocultas, ya que centra los datos en torno a cero, lo que puede mejorar el entrenamiento.

**Softmax**: Se usa en clasificación multiclase, asignando probabilidades a cada clase en la capa de salida. La suma de todas las probabilidades es igual a 1, lo que facilita la selección de la clase más probable.



## 3 .Capas totalmente conectadas

Las capas totalmente conectadas (también llamadas capas densas) son similares a las de redes neuronales tradicionales. Cada neurona de estas capas está conectada a todas las neuronas de la capa anterior.

espués de que las capas convolucionales y de agrupación han extraído patrones locales, las capas FC combinan estas características para formar representaciones globales. Esto les permite tomar decisiones finales, como asignar una clase en tareas de clasificación o predecir un valor en tareas de regresión. Estas capas son esenciales para que la red haga la interpretación final de los datos procesados.

model.add(layers.Flatten())

**- \*\*Capa de Aplanamiento (Flatten):\*\*** Esta capa se utiliza para transformar los datos de entrada en un vector unidimensional. Toma la salida de la última capa convolucional o de agrupación, que es un tensor tridimensional (o un tensor con más dimensiones si hay capas convolucionales apiladas), y lo "aplana" en un vector de una dimensión, conservando todos los elementos en el mismo orden. Esencialmente, toma una matriz o tensor multidimensional y la convierte en un vector unidimensional lineal.

La razón principal para agregar una capa de aplanamiento después de las capas convolucionales y de agrupación en una CNN es preparar los datos para las capas totalmente conectadas. Las capas convolucionales y de agrupación extraen características espaciales de la entrada, pero las capas totalmente conectadas requieren una entrada unidimensional

**- \*\*Capa Totalmente Conectada (Dense layer):**

Modificar el número de neuronas en una capa densa puede tener un impacto significativo en el rendimiento y la capacidad de aprendizaje de la red neuronal. Aquí hay algunas consideraciones sobre cómo el número de neuronas afecta el modelo:

**1. \*\*Número de Parámetros:\*\*** El número de neuronas en una capa densa determina el número de parámetros entrenables en esa capa. Cada neurona está conectada a todas las neuronas de la capa anterior, por lo que el número total de conexiones (y por lo tanto de parámetros) aumenta con el número de neuronas. **Añadir más neuronas aumentará la capacidad del modelo para aprender relaciones más complejas** en los datos, **pero también aumentará el riesgo de sobreajuste**

**2. \*\*Complejidad del Modelo:\*\***

El número de neuronas en una capa densa afecta la complejidad y la capacidad de representación del modelo.

**3. \*\*Costo Computacional:\*\***

Más neuronas en una capa densa significan más operaciones computacionales durante el entrenamiento y la inferencia

La **función softmax** se utiliza en la **capa de salida** de una red neuronal para **clasificación multiclase** porque:

1. **Probabilidades Normalizadas**: Convierte las salidas en **probabilidades**, donde la suma de todas es 1, lo que facilita la interpretación de los resultados.
2. **Clasificación Multiclase**: En problemas con varias clases, softmax asigna una **probabilidad** a cada clase. La clase con la probabilidad más alta es la predicción final.
3. **Entrenamiento con Cross-Entropy**: Se combina con la **entropía cruzada** para medir el error durante el entrenamiento y ajustar los pesos de manera eficiente.
4. **Cálculos Eficientes**: La derivada de softmax es fácil de calcular, lo que mejora el proceso de optimización en el entrenamiento.

Además de las capas convolucionales, de agrupación, de aplanamiento (flatten) y totalmente conectadas, existen varias otras capas

## 3- OTRAS CAPAS

1. \*\*Capa de Normalización por lotes (Batch Normalization):

**Normalización de Activaciones**: Durante el entrenamiento, las activaciones de las capas pueden volverse muy grandes o muy pequeñas, lo que puede hacer que el modelo sea más difícil de entrenar. **Batch Normalization** normaliza estas activaciones (es decir, las pone en una escala estándar) para cada mini-lote de datos, asegurando que tengan una media cercana a 0 y una desviación estándar cercana a 1.

2. \*\*Capa de Regularización (Dropout):\*\*

La capa de dropout se utiliza para reducir el sobreajuste en la red neuronal. Durante el entrenamiento, aleatoriamente "apaga" un porcentaje de las unidades de la capa anterior, lo que ayuda a prevenir la dependencia excesiva entre las unidades y mejora la generalización del modelo.

3. \*\*Capa de Normalización por Instancia (Instance Normalization):\*

La **normalización por instancia** normaliza cada **imagen de manera individual**. Es decir, cada imagen se trata por separado, sin depender de las demás.

4. \*\*Capa de Convolución Transpuesta (Transpose Convolution):\*\*

En una **capa de convolución transpuesta**, se aplica un filtro similar, pero en lugar de reducir el tamaño, se usa para **expandir** la imagen. Este proceso "aumenta" las dimensiones de la entrada (por ejemplo, de una imagen de 14x14 a 28x28).

La **Capa de Atención (Attention Layer)** es una técnica poderosa utilizada en redes neuronales para **enfocar el modelo en las partes más importantes de los datos de entrada**, dándole más peso a las áreas relevantes mientras ignora las menos importantes.

Por ejemplo, si una red está procesando una imagen para clasificarla, algunas partes de la imagen (como los objetos principales) serán más importantes que otras partes (como el fondo). La capa de atención permite que la red "preste más atención" a esas áreas clave, mejorando la capacidad de la red para tomar decisiones.

La **Capa de Residuos (Residual Layer)**

es una técnica utilizada en redes neuronales profundas para mejorar el entrenamiento y rendimiento de redes muy profundas, como las arquitecturas **ResNet**

**a Capa de Convolución Separable (Depthwise Separable Convolution)**

La Capa de Convolución Separable (Depthwise Separable Convolution) es una variante optimizada de la convolución estándar utilizada en redes neuronales convolucionales (CNNs). Su objetivo principal es hacer que las operaciones de convolución sean más eficientes computacionalmente, especialmente en aplicaciones con recursos limitados, como dispositivos móviles o sistemas con capacidad de procesamiento baja.

## 5. Compilar el modelo

a función compile() permite establecer tres aspectos fundamentales para entrenar una red neuronal: el **optimizador**, la **función de pérdida** y las **métricas**.

# Entrenar el modelo

### 1. **Optimizer (Optimizador)**:

son algoritmos utilizados en el entrenamiento de redes neuronales para ajustar los pesos del modelo de manera que minimicen la función de pérdida

. **SGD (Stochastic Gradient Descent) - Descenso de Gradiente Estocástico**

**El optimizador SGD actualiza los pesos en función del gradiente de la función de pérdida. Lo hace tomando una muestra aleatoria de los datos (en lugar de usar todo el conjunto de entrenamiento) y calculando el gradiente de la pérdida con respecto a los pesos.**

**2. Adam (Adaptive Moment Estimation)**

**¿Qué hace? Adam es un optimizador más avanzado que combina lo mejor de dos optimizadores: AdaGrad y RMSProp. Calcula tasas de aprendizaje adaptativas para cada parámetro, basándose en los momentos de primer y segundo orden del gradiente.**

**3. RMSProp (Root Mean Square Propagation)**

**Similar a Adam, pero ajusta la tasa de aprendizaje basándose en el promedio de las magnitudes de los gradientes.**

**Otros optimizadores: Hay muchos más disponibles como adagrad, adadelta, etc., cada uno con características especiales para distintos problemas.**

### 2. Loss (Pérdida):

**sparse\_categorical\_crossentropy**: Se utiliza comúnmente para problemas de clasificación multiclase cuando las etiquetas están codificadas como enteros (no one-hot encoded).

**categorical\_crossentropy**: Similar a la anterior, pero se usa cuando las etiquetas están en formato one-hot encoded.

**mean\_squared\_error**: Se utiliza en problemas de **regresión** donde se predicen valores continuos.

**Otras funciones de pérdida**: Existen más funciones de pérdida para tareas específicas, como binary\_crossentropy para clasificación binaria, o hinge para SVM.

### 3. **Metrics (Métricas)**:

* **accuracy**: Mide la proporción de predicciones correctas. Es muy común en problemas de clasificación.
* **precision, recall, f1-score**: Métricas adicionales que te dan información más detallada sobre el rendimiento de clasificación, especialmente útil cuando las clases están desbalanceadas.
* **mean\_absolute\_error**: Común en problemas de regresión, mide el error medio absoluto entre las predicciones y las etiquetas reales.

Configuraciones adicionales en compile():

1. **loss\_weights (Pesos de Pérdida)**

**Ejemplo en el modelo: Si estás entrenando una red multitarea (como un modelo que hace clasificación y predicción de valores numéricos), puedes decirle al modelo cuánta importancia tiene cada función de pérdida.**

**Imagina que tienes dos amigos: uno está a cargo de la comida y otro de la música en la fiesta. La comida es más importante para el éxito de la fiesta, así que le das más responsabilidad (peso) al amigo de la comida.**

**2. sample\_weight\_mode (Ponderación de Muestras)**

**Ejemplo en el modelo: Cuando entrenas un modelo, algunas muestras pueden ser más importantes que otras, por ejemplo, si tienes datos desbalanceados (como clases raras en clasificación). Puedes dar más peso a ciertas muestras para que el modelo preste más atención a ellas.**

**o que hace: Permite que algunas "muestras" sean más importantes que otras. Si en tu fiesta, algunas personas son VIP (más importantes), les puedes dar más peso (más atención).**

**3. weighted\_metrics (Métricas Ponderadas)**

**Define cómo se deben evaluar las "métricas" (como la precisión). Algunas métricas pueden ser más importantes que otras, y puedes darle más peso a esas métricas.**

**4. run\_eagerly (Ejecución Ansiosa)**

**Imagina que quieres ver cómo va la fiesta en tiempo real y corregir cualquier problema inmediatamente (como un invitado que no está disfrutando de la comida o la música). En lugar de esperar al final, quieres ver los resultados en cada paso.**

## 

## 6 Entrenar el modelo

## 

**model.fit()**: Función para entrenar el modelo ajustando sus parámetros con los datos de entrenamiento.

**train\_images y train\_labels**: Datos de entrada y sus etiquetas usadas para entrenar.

**epochs=10**: Número de veces que el modelo recorre todo el conjunto de entrenamiento.

**validation\_data**: Conjunto de datos (imágenes y etiquetas) usado para evaluar el modelo después de cada época.

## 