Apuntes sobre IA (borrador)

Sergio de Mingo

2 de febrero de 2026

Contents

1 Redes Neuronales Convolucionales

Anteriormente aplanamos una imagen de 28×28 a 784. Al hacer eso, la red ya no sabe si el píxel 1 está al lado del píxel 2 o del píxel 28. Simplemente ve una lista de números. Piensa en esto: ¿Cómo podrías tú reconocer una cara si te dieran todos los píxeles de una foto mezclados en una bolsa? Sería imposible. Las Redes Neuronales Convolucionales o CNN vienen a resolver precisamente eso. Hasta ahora, las redes veían los píxeles como una lista plana. Si movíamos el dibujo de un "8" un poco a la derecha, la red se confundía porque los píxeles ya no caían en las mismas neuronas de entrada. Las CNN solucionan esto imitando el córtex visual humano. En lugar de conectar cada píxel a una neurona, usamos un filtro. Como si fuera una pequeña rejilla de 3×3 que se desliza sobre la imagen. Este filtro no mira toda la imagen a la vez; mira solo un pequeño trozo, extrae una característica (como una línea vertical o un borde) y se mueve al siguiente píxel. La red ahora no aprende a reconocer «píxeles en tal posición», sino que aprende los valores de ese filtro que detectan rasgos importantes (bordes, curvas, texturas). La estructura maestra de una CNN podría resumirse en los siguiente puntos:

- **Capa de convolución**: Es el motor. Aquí es donde los filtros escanean la imagen. Si aplicas 32 filtros diferentes, obtendrás 32 mapas de características (versiones de la imagen donde resaltan cosas distintas).
- **Capa de pooling**: Es el sintetizador. Su trabajo es reducir el tamaño de la imagen. Toma, por ejemplo, un cuadrado de 2×2 y se queda solo con el valor más alto (el más brillante). Hacemos esto porque si detectamos un borde, no nos importa el píxel exacto, nos importa saber que ahí hay un borde. Esto hace que la red sea invariante a la traslación.
- **Capa totalmente conectada**: Al final de la red, después de que los filtros hayan extraído toda la información, volvemos a usar las capas que ya conoces para tomar la decisión final (¿Es un 8 o es un 3?).

A nivel global podemos resumir las funciones de las capas de la siguiente manera. Mientras que las capas iniciales detectan cosas simples como líneas, puntos y colores, las capas intermedias combinan esas líneas para detectar formas como círculos, cuadrados, ojos, etc. Por último, las capas finales combinan esas formas para detectar objetos complejos como caras, coches, números, etc.

Vamos a empezar la red trabajando con una entrada en forma de matriz de 28x28. Ya no trabajamos con un vector plano como en las redes anteriores. Usaremos para esta primera capa 32 filtros de 3x3. De esa capa salimos con una matriz de 26x26 pues al aplicar filtros de 3x3 y deslizarlos por toda la imagen, cuando llegamos a los bordes al tener que evitar que estos filtros se salgan perdemos el equivalente a un marco de 1 por cada lado de la matriz. De ahí que la matriz de 28x28 se quede en una de 26x26.

En la capa de *pooling* es donde se produce una compresión. Aplicando un Max Pooling de 2x2 a la salida de 26x26

de la capa anterior, el pooling divide la imagen en bloques de 2x2 y se queda con un máximo de uno. Esto reduce a la mitad la matriz con un resultado de 13x13. Resumimos la arquitectura indicando los tensores usados:

1. **Entrada**: [1x28x28] Tensor de 28x28 para un canal de color.

2. **Tras la convolución** con 32 filtros de 3x3: Obtenemos [32x26x26], lo que equivale a 32 versiones distintas de la imagen original, una por filtro usado.

3. **Tras el Pooling** de 2x2: Obtenemos [32x13x13]

Aunque ahora tenemos más datos (32 mapas en lugar de 1), estos datos están jerarquizados. Cada uno de los 32 canales se especializa en algo: el canal 1 puede detectar líneas horizontales, el canal 2 detecta esquinas, etc

Pero antes de llegar a la salida (los 10 números del MNIST), tenemos que volver al mundo que ya conoces. Si al final de las capas de convolución nos queda un bloque de $32 \times 13 \times 13$, lo «aplanamos» multiplicando todas sus dimensiones: $32Ö13Ö13 = 5408 neuronas$. Esas 5408 neuronas entrarán en una capa `nn.Linear` clásica.

```
class NetCNN(nn.Module):
    def __init__(self):
        super(NetCNN, self).__init__()

        # 1. CAPAS CONVOLUCIONALES (Extraccion de rasgos)
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)

        # 2. CAPAS LINEALES (Clasificacion)
        self.fc1 = nn.Linear(5 * 5 * 64, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 5 * 5 * 64)

        x = F.relu(self.fc1(x))
        x = self.fc2(x) # Salida de 10 neuronas

        return F.log_softmax(x, dim=1)
```

Vamos a detallar ahora los aspectos mas interesantes del código. Comenzamos con las dos capas convolucionales: `conv1` y `conv2`. ¿Por qué dos? La respuesta está en la jerarquía de la visión. Imagina que estás intentando reconocer una cara. Con una sola capa (`conv1`), la red solo ve palitos y puntos (bordes). Es imposible reconocer una cara solo sabiendo que hay 100 líneas verticales. Necesitas saber si esas líneas forman un ojo, una nariz o una boca. Ahí es donde entra la `conv2`. La clave de esta segunda capa es que no mira la imagen original. Mira los mapas de características que creó la primera capa. Por ejemplo, mientras que `conv1` detecta rasgos simples (bordes, diagonales, manchas de color), `conv2` mira dónde hay bordes y los combina. Por ejemplo: «Si aquí hay una curva hacia arriba y debajo una línea horizontal, entonces aquí hay un semicírculo». Sin esa segunda capa, tu red sería como alguien que ve las letras de un libro pero es incapaz de juntarlas para formar palabras. Respecto a los canales (el primer parámetro de la función `nn.Conv2d()`) en la `conv1` pusimos 32 filtros, pero en la `conv2` pusimos 64. Al aumentar los canales en la segunda capa, le damos a la red un «vocabulario» mucho más rico para describir lo que está viendo (curvas cerradas, cruces, ángulos agudos, etc.). Estamos aumentando la semántica de la red. Resumiendo, si quitamos la `conv2`, la red intentaría pasar directamente de «detectar bordes» a «decidir si es un 8». Como no ha tenido una capa intermedia para entender qué es un «círculo», le costará muchísimo generalizar. Tendría que compensarlo con una capa lineal (`nn.Linear`) gigantesca y muy ineficiente.

Si vamos a la función `forward()` vemos que tras la primera convolución aplicamos un *pooling* y tras la segunda otro.

Al poner capas una tras otra (especialmente con un Pooling de por medio), cada neurona de la capa 2 está viendo un área mucho más grande de la imagen original que una neurona de la capa 1. Una neurona en `conv1` solo mira 3×3 píxeles. No sabe si está en la parte de arriba o de abajo de un número. Tras el *pooling* y la `conv2`, una sola neurona de esa segunda capa puede estar recibiendo información que originalmente cubría un área de 10×10 o más. Por eso se dice que esta segunda capa tiene «perspectiva». Puede entender la geometría del número, no solo los píxeles sueltos. A esto se le llama también campo receptivo.