Apuntes de clase #2

Luis Felipe Calderón Pérez
Escuela de Ingeniería en Computación
Tecnológico de Costa Rica
Cartago, Costa Rica
2021048663
9-10-2025

***Resumen*—Este documento presenta los apuntes de la decima semana del curso de inteligencia artificial. Se realizó un repaso de las redes neuronales convolucionales (CNN), partiendo de las limitaciones de las redes *fully connected* y la necesidad de extraer información espacial de las imágenes. Se estudió el funcionamiento de la estructura de las capas convolucionales, *pooling* y *fully-connected* y *feature map*. Finalmente, se discutieron los principios para diseñar arquitecturas convolucionales eficientes, considerando tamaños de filtro, *stride*, *padding* y reducción de parámetros.**

***Index Terms*—CNN, capas convolucionales, *transfer learning*, arquitecturas convolucionales.**

I. Breve repaso de la clase anterior

I-A. Redes Convolucionales

Recibimos un input de características, el cual transformamos con una serie de capas ocultas.

Se había visto hasta el momento las redes *fully connected*. Teníamos el problema de que aprendemos una secuencia de píxeles, lo que nos lleva a errores si movemos los objetos de lugar o si aplicamos rotación a los objetos. Lo correcto sería sacar la información de la imagen para tomar una decisión.

También se habló del dataset de CIFAR-10, donde hay imágenes con 3 canales de 32x32. Y nos damos cuenta de que no es escalable, ya que se tendrían 120,000 parámetros que ajustar únicamente en la entrada.

Por ello se buscan métodos más eficientes; entonces llegamos a las ConvNet, en donde las neuronas se organizan en 3 dimensiones: largo, ancho y profundidad (canales). Las mismas estarán conectadas a pequeñas regiones de la capa anterior, y esto reduce el tamaño de la imagen a un vector.

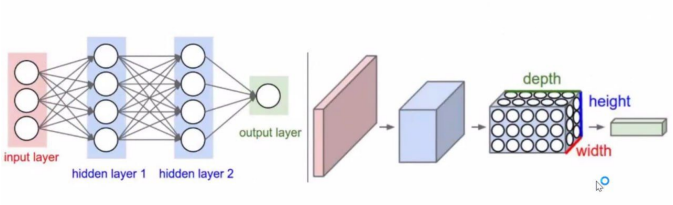

Figura 1. Arquitectura CNN vs NN [3]

En cada cara del cubo de las neuronas se tienen n filtros de tamaño acorde a las imágenes de entrada y obtenemos un *Feature map*, que es el resultado de aplicar el filtro a la imagen anterior.

I-B. Arquitectura de CNN

Requiere de 3 capas principales

1. *Convolutional layer*
   - Computa el filtro contra una imagen.
   - Recibe de entrada el ancho, largo y canales.
   - Tiene n filtros que extraen características de las imágenes.
   - Aplica una capa de activación
   - Tiene como parámetros $(wx)^n + b$
2. *Pooling layer*
   - Reduce de la imagen en ancho y largo.
   - Se encarga de aplicar el *downsampling* a lo largo del ancho y largo.

- No tiene parámetros.
- Se introduce periódicamente en medio de capas convolucionales.
- Lo más usual es usar *Max Pooling*.
- Fórmula dimensionalidad
  - Entrada WxHxD.
  - k, tamaño de kernel.
  - s, *stride*.
  - D, mantiene la profundidad.

  $w_2 = \frac{w-k}{s} + 1$

3. *Fully-Connected*

   Esta parte clasifica, ya que calcula la probabilidad de pertenecer a una clase; transformando una imagen de píxeles a probabilidad de pertenecer a una clase.

Las *CNNs* nos permiten resolver

- Clasificación de imágenes.
- Segmentación de objetos.
- Segmentación de instancias.
- Procesamiento de imágenes.

II. Capa de Convolución

Tiene varias características, están compuestas por varios filtros (w), va a tener un comportamiento local, lo deslizamos y va a seguir extrayendo las mismas features alrededor de la imagen. Esto permite que los pesos se ajusten para que sirvan en una posición como en otra.

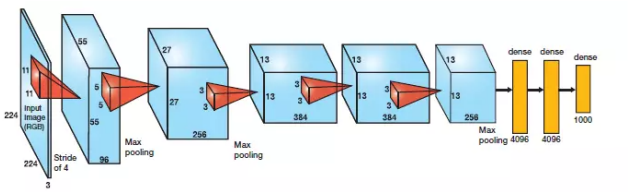

Figura 2. Arquitectura AlexNet [2]

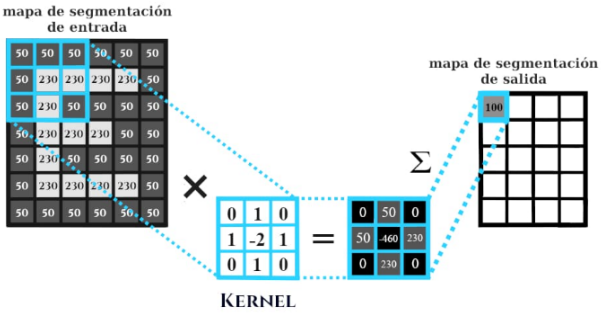

Figura 3. Funcionamiento kernel capa de convolución [3]

II-A. Filtro o Kernel

Es una matriz bidimensional de números, que transforma una imagen en el momento en que deslice ese filtro, produciendo una imagen como salida. Para aplicar los filtros se va a tener algo similar a un caso donde una imagen de entrada tiene mucho ruido y se le aplica un Gaussian Kernel (tiene una campana de Gauss) y da una imagen resultante como si tuviera un Blur. Dependiendo del filtro usado, obtenemos otras formas como resultado e influimos en el procesamiento.

II-B. Local Receptive fields

Una neurona esta conectada a un campo en específico del input. Esto es muy eficiente, ya que podría haber filtros que extraigan líneas verticales, otros horizontales y así.

II-B1. Campo receptivo: Es un filtro de nxn, donde cada neurona estará enfocada en un solo campo receptivo.

II-B2. Stride: Es la forma clásica de deslizar el filtro, según una cantidad de pasos que realiza el filtro sobre la imagen durante la convolución.

II-B3. Padding: Técnica para agregar pixeles alrededor del borde de la imagen, permitiendo controlar el tamaño de salida de la convolución. Se recomienda que el padding se llene con 0, ya que si se usa 1 podría generar ruido o mala data. Y su fórmula es $\frac{k-1}{2}$, k = tamaño del filtro.

II-B4. Cálculo de dimensiones:

- m, Cantidad de pixeles en fila/columna.
- k, Tamaño del *kernel*.
- p, Tamaño del *padding*.
- s, cantidad de pasos.

$\frac{m-k+2p}{s} + 1 =$ Dimension resultante

II-C. Pesos

Si se tuviera una imagen de 224x224x3, con un tamaño de kernel 11, stride de 4 y padding de 0, y le aplicamos la fórmula anterior da 55. Y con la arquitectura de AlexNet 2 obtenemos que aprendimos 96 de profundidad.

II-D. Pesos Compartidos

Si ya tenemos un filtro que extrae cierta caracteristica, y sirve para una posición; también sirve para otra posición. Por lo que, vamos a usar el mismo filtro para toda la imagen.

II-E. Transfer learning

Se menciona que en el paper de AlexNet después de aplicar su arquitectura y lo referente a ella. Y se dan cuenta que en las primeras capas hay figuras o información similar. Por lo que se introduce el término de *transfer learning*, que consiste en pasar el peso de las primeras capas a otra red, para ahorrar tiempo de entrenamiento.

III. Arquitecturas Convolucionales

Se componen de *Convolutional layer*, *Pooling layer* y de *Dense layer*. Se deben tomar desiciones sobre nuestra arquitectura, por ejemplo, si la convolución reduce el input debo decidir si hago o no el pooling. Y estas desiciones determinan el comportamiento del tamaño de la imagen, pero si la imagen es muy reducida, le llega poca información a la *fully connected*. Se introdujó el término de stack, que es, $INPUT \rightarrow [[conv \rightarrow relu] * n \rightarrow Pool?] * m \rightarrow [fc \rightarrow relu] * k \rightarrow fc, m \geq 0, k \geq 0$, y se menciona que en papers la cantidad de convoluciones y relu se usa la fórmula de $3 \geq n \geq 0$.

III-A. ¿Que arquitectura preferimos?

Se prefiere a las arquitecturas con convoluciones pequeñas, ya que las convoluciones grandes nos llevan a que las neuronas se computen de forma lineal y que la cantidad de pesos sea mayor.

III-B. Algunas "reglas"

- El tamaño de la imagen debería ser divisible por 2.
- Las convoluciones deben usar campos receptivos pequeños 3x3, con un stride de 1.
- Para *pooling laye*r es común *max pooling* de F=2, S=3.
- s, cantidad de pasos.

III-C. Menciones finales

Al final se mencionan arquitecturas similares a LeNet para tomar en cuenta para el proyecto, tales como, AlexNet, AFNet, GoogleNet (reduce parámetros), VGG16 y ResNet.

Nota: Embedding, información distribuida en espacio vectorial que retorna mi NN.

Referencias

[1] "ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks," CV-Tricks.com, Aug. 01, 2022. https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

[2] R. R. Abril, "Redes convolucionales," La Máquina Oráculo, Jul. 2025, [Online]. Available: https://lamaquinaoraculo.com/deep-learning/redes-neuronales-convolucionales/

[3] S. A. P. Portuguez, "Apuntes de la clase de inteligencia artificial," Cartago, Costa Rica, agosto 2025, clase del 9 de octubre del 2025.