Apuntes Semana 11 Clase #1

14/10/2025

Alex Steven Naranjo Masís
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Email: alnaranjo@estudiantec.cr

***Resumen*—Este documento recopila los apuntes de la clase del martes 14 de octubre de 2025 para el curso de Inteligencia Artificial. Se abordaron los fundamentos de las Redes Neuronales Convolucionales (CNN), explicando el funcionamiento de los filtros, el campo receptivo, el *stride*, el *padding* y las capas de *pooling* para la extracción de características en imágenes. Además, se estudiaron arquitecturas clásicas como LeNet, AlexNet, GoogleNet/Inception, VGG16, ResNet y DenseNet. Finalmente, se introdujeron los conceptos de *embeddings*, visualización de activaciones y autoencoders, analizando sus aplicaciones en reducción de dimensionalidad, detección de anomalías, eliminación de ruido y super-resolución, junto con consideraciones prácticas de entrenamiento y selección de hiperparámetros.**

***Index Terms*—Redes Neuronales Convolucionales, Pooling, Embeddings, Visualización, Autoencoder, Deep Learning**

I. Introducción

Las Redes Neuronales Convolucionales (CNN) son un pilar en la visión por computadora moderna, pues permiten extraer automáticamente características jerárquicas de las imágenes. Comprender sus componentes como filtros, campos receptivos, *stride*, *padding* y *pooling* es esencial para diseñar arquitecturas eficientes. Por su parte, los autoencoders complementan este aprendizaje al representar la información de forma comprimida, sin necesidad de etiquetas externas, y habilitan tareas de aprendizaje no supervisado/semisupervisado.

II. Fundamentos de Redes Neuronales Convolucionales (CNNs)

A. Filtros (Kernels) y Campos Receptivos

Un filtro 2D de tamaño $k \times k$ se desliza sobre la imagen (o mapa de activación) para producir un *feature map*. Para una entrada RGB $H \times W \times C_{in}$ y C_{out} filtros, cada filtro tiene tamaño $k \times k \times C_{in}$ y produce un canal en la salida.

- **Filtro Gaussiano:** suaviza la imagen (blur) y reduce ruido; resalta contornos al combinarse con operadores de gradiente.
- **Campo Receptivo (RF):** región de la entrada que "ve" una neurona de una capa dada. Aumenta con la profundidad. Si encadenamos capas con kernel k_i y *stride* s_i, el RF efectivo crece de forma acumulativa.

Parámetros y costo: el número de parámetros en una capa conv es $k^2 \cdot C_{in} \cdot C_{out} + C_{out}$ (sesgo). La complejidad computacional se aproxima por $H_{out} \cdot W_{out} \cdot k^2 \cdot C_{in} \cdot C_{out}$.

B. Parámetros de la Convolución: Stride, Padding y Tamaño de Salida

Para una entrada 1D de longitud m, kernel k, *padding* p y *stride* s, la salida es:

$$\text{out} = \left\lfloor \frac{m + 2p - k}{s} \right\rfloor + 1.$$

En 2D se aplica por dimensión (alto y ancho). El *padding* simétrico típico para "conservación de tamaño" con $s = 1$ es $p = \frac{k-1}{2}$ (si k es impar). El *stride* > 1 reduce la resolución espacial.

C. Pesos Compartidos y Eficiencia

La *compartición de pesos* aplica el mismo kernel en todas las posiciones espaciales, reduciendo parámetros frente a capas densas. En primeras capas, la red aprende bordes y texturas; en capas profundas, patrones semánticos más abstractos.

D. Capa de Pooling

Reduce la resolución espacial conservando canales:

- **Max Pooling:** retiene el valor máximo de cada ventana.
- **Average Pooling:** promedia los valores.

Regla práctica: *pooling* 2×2 con *stride* 2 para reducción a la mitad. Mantiene $D = C_{in}$ y reduce H, W.

E. Activaciones, Normalización y Regularización

- **Activación:** ReLU es estándar en CNN modernas (evita saturación y acelera entrenamiento). Tanh/sigmoid pueden usarse en salidas específicas.
- **Batch Normalization (BN):** estabiliza la distribución de activaciones, permite mayores tasas de aprendizaje y acelera la convergencia.
- **Regularización:** Dropout (típico en capas densas), L2 (*weight decay*) y *data augmentation* reducen sobreajuste.

F. Capa Fully-Connected (MLP) y Clasificación

Tras extraer mapas de activación, se aplica *flatten* (o *global average pooling*) y capas densas para clasificación. En problemas multi-clase se usa *softmax* y pérdida de entropía cruzada.

III. Arquitecturas Convolucionales

A. LeNet-5

Pionera (LeCun, 1998) para dígitos manuscritos (MNIST). Dos bloques conv+pooling y capas densas. Introdujo la viabilidad práctica de CNNs.

B. *AlexNet (2012)*

Krizhevsky et al. popularizan ReLU, *dropout*, entrenamiento en múltiples GPUs y kernels grandes (11×11, 5×5, 3×3) en entradas 224×224. Disparó la adopción de *deep learning* a gran escala.

C. *ZFNet y Visualización Intermedia*

Ajusta tamaños de kernel/stride y estudia *feature maps* internos para entender qué aprende cada capa, motivando prácticas de diseño y depuración.

D. *GoogLeNet / Inception*

Módulos con ramas paralelas (1×1, 3×3, 5×5 + *max pooling*); reduce parámetros (de ∼60M a ∼4M) usando cuellos 1×1 y *global average pooling* al final.

E. *VGG-16*

Filosofía de simplicidad: solo 3×3 + profundidad (16/19 capas). A pesar de muchos parámetros, es un *baseline* didáctico muy usado.

F. *ResNet (Redes Residuales)*

Skip connections ($y = F(x) + x$) permiten entrenar redes muy profundas mitigando *vanishing gradient*. Bloques *basic/bottleneck* se apilan eficientemente.

G. *DenseNet*

Conexiones densas "todas con todas" dentro del bloque; fomenta reutilización de características, mejora el flujo de gradiente y reduce parámetros a igual rendimiento.

IV. Explicabilidad del Modelo y Embeddings

A. *Visualización de Activaciones y Filtros*

Observar *feature maps* muestra qué regiones activan cada neurona. En capas iniciales, activaciones recuerdan bordes/colores; en capas profundas, partículas semánticas más complejas.

B. *Embeddings y Reducción de Dimensionalidad*

Los *embeddings* son vectores en $\mathbb{R}^d$ que capturan semántica. Vectores de clases similares tienden a agruparse en el espacio latente.

- **t-SNE:** proyección no lineal a 2D/3D preservando vecindarios locales.
- **PCA:** proyección lineal; útil como *baseline* o preprocesamiento.

C. *Mapas de Activación (Heatmaps)*

Heatmaps señalan zonas que más influyen en la predicción (útil en aplicaciones médicas/industriales para justificar decisiones).

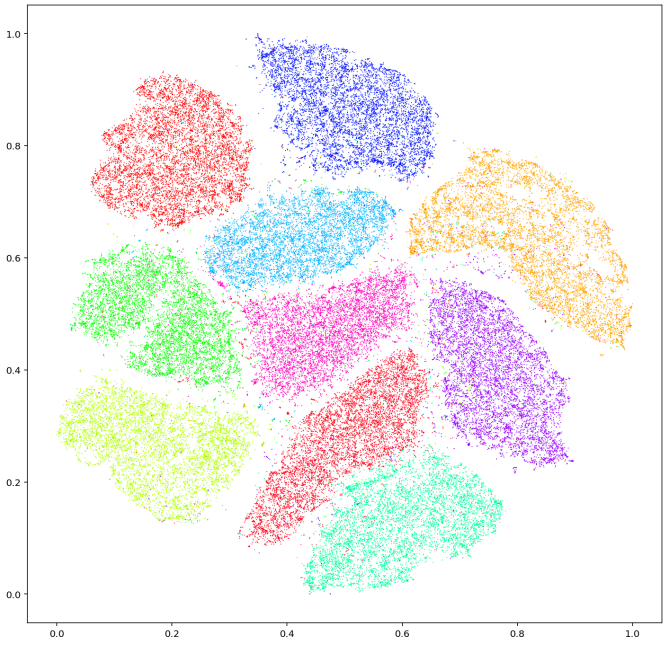

Figura 1. Representación de *embeddings* mediante t-SNE.

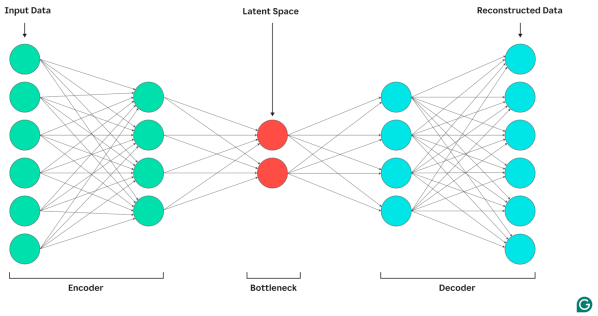

Figura 2. Estructura básica de un Autoencoder.

V. Autoencoders (Codificadores Automáticos)

A. *Estructura General y Objetivo*

$$\text{Encoder} \rightarrow \text{Espacio Latente} \rightarrow \text{Decoder}$$

Aprenden a reconstruir la entrada. Aunque la señal de entrenamiento es auto-supervisada (salida = entrada), se consideran típicamente métodos no supervisados por no requerir etiquetas externas.

B. *Componentes y Variantes*

Encoder: reduce espacialidad y comprime información (conv + *downsampling*).
Latente: vector/tensor compacto; su tamaño controla capacidad vs. compresión.

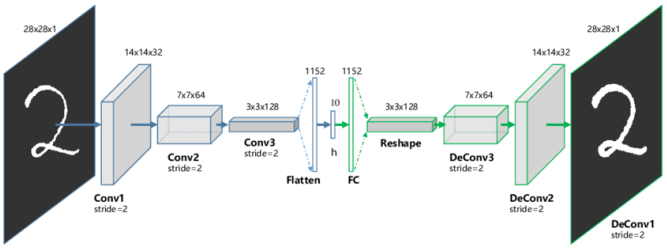

Figura 3. Ejemplo conceptual de super-resolución con autoencoder.

Decoder: reconstruye con *upsampling* o convoluciones transpuestas.
Variantes: *Denoising* (entrenar con entrada ruidosa y salida limpia), *Sparse* (regulariza latente), *Under/Overcomplete*. *(Nota: VAEs y GANs exceden el alcance de esta clase, pero se relacionan con lo generativo.)*

C. Funciones de Pérdida Comunes

- **MSE** (*Mean Squared Error*): reconstrucción píxel a píxel (continuo).
- **MAE**: más robusto a *outliers*.
- **BCE**/*BCEWithLogits*: para imágenes normalizadas/binarizadas.
- **Perceptual/SSIM** (opcional): mejor correlación perceptual que MSE.

D. Aplicaciones

- **Reducción de dimensionalidad** y almacenamiento eficiente en BBDD vectoriales.
- **Detección de anomalías:** entrenar con datos "normales"; altas pérdidas de reconstrucción sugieren anomalías.
- **Eliminación de ruido (Denoising).**
- **Super-resolución:** reconstruir versiones de mayor resolución.

E. Hiperparámetros Relevantes

- **Tamaño del latente:** más pequeño = mayor compresión/menor fidelidad; más grande = mayor capacidad/costo.
- **Profundidad del encoder/decoder** y tipo de *upsampling* (nearest/bilinear vs. *ConvTranspose2d*).
- **Pérdida de reconstrucción** (MSE/MAE/BCE/SSIM) según dominio.

VI. Buenas Prácticas de Entrenamiento y Diseño

A. Preprocesamiento y Aumento de Datos

- Normalización por canal (media/desviación del *dataset*).
- *Data augmentation* moderado: flips, *crops*, ligeros *jitter*s; evita *overfitting*.

B. Optimización y Regularización

- **Optimizadores:** SGD+momentum (control fino), Adam (rápida convergencia).
- **LR scheduling:** *step*/*cosine*/*plateau*.
- **Regularización:** L2 (*weight decay*), *dropout* (sobre todo en densas), *early stopping*.

C. Reglas Prácticas de Arquitectura

- Dimensiones divisibles entre 2 para facilitar *pooling*.
- Preferir kernels pequeños (3×3 / 5×5) y apilar profundidad para mayor no linealidad.
- Usar *global average pooling* antes de densas para reducir parámetros.
- Insertar BN después de conv y antes de ReLU para estabilidad.

D. Notas de Implementación

En *frameworks* como PyTorch, la reconstrucción en decoders suele emplear `ConvTranspose2d` o `Upsample`+1 $\times$ 1 conv; para clasificación, `CrossEntropyLoss` (con `LogSoftmax` interno) es estándar.

VII. Conclusiones

Las CNN han transformado la visión por computadora al extraer jerarquías de características de manera automática y eficiente. No obstante, su interpretabilidad sigue siendo un reto; técnicas de visualización, *embeddings* y *heatmaps* ayudan a entender y validar decisiones. Los autoencoders extienden estos conceptos hacia la compresión, reconstrucción y generación de datos, habilitando aplicaciones prácticas como reducción de dimensionalidad, detección de anomalías y super-resolución. Una ingeniería cuidadosa de arquitectura, *data* y entrenamiento es clave para un desempeño robusto.

Referencias

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[2] C. Szegedy et al., "Going deeper with convolutions," *CVPR*, 2015.