Apuntes Semana 11- 08/05/2025

Elaborado por:
Yessenia Solano Retana-2021050802

***Abstract*—Este documento recopila los apuntes de la semana 11 del curso de Inteligencia Artificial, relacionados con arquitecturas convolucionales.**

Aspectos Administrativos

Sobre el proyecto 2

- Con el modelo simple ya se pueden obtener buenos resultados.
- Sobre los audios, se recomienda guardarlos y procesarlos por batchs para ahorrar espacio
- Se van a replantear los puntos extra debido a la simplicidad del modelo, el profesor va a notificar oficialmente como se hará.
- Se pide una buena justificación y pruebas en el articulo a entregar.
- Las revisiones serán la siguiente semana, semana 12.

Noticias de la Semana

OpenAI ha publicado una guía práctica para la construcción de agentes de inteligencia artificial, dirigida a equipos de producto e ingeniería que buscan desarrollar sistemas autónomos. La guía se titula *A practical guide to building agents*. Puede accederse a través del siguiente enlace:

Guía de OpenAI sobre agentes

Arquitecturas Convolucionales

Composición

Existen 3 grandes capas que se describirán a continuación, en cada una de ellas hay que tener consideraciones como tamaño, activaciones, etc.

leftmargin=*

- **Convolutional Layer**: Encargadas de extraer las características de la imagen.
- **Pooling Layer**: EL procesamiento que se le hace a las imágenes para reducir el tamaño, esto básicamente reduce progresivamente el tamaño espacial de la representación (ancho y alto), conservando las características más relevantes y disminuyendo la carga computacional.
- **Dense Layer**: Se encarga de tomar las características que se extrajeron de las imágenes y hacer un clasificador a partir de estas características.

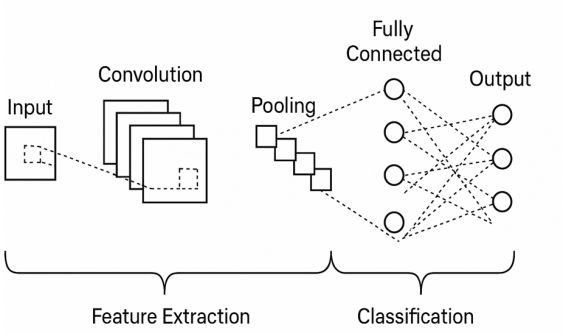

FIGURE 1. Arquitecturas convolucionales.

En la imagen anterior, en el clasificador se tiene toda aquella información valiosa que se extrajo de las imágenes. A partir de esto tenemos información ya del modelo y la salida.

Decisiones de diseño

Existen muchas decisiones de diseño que debe tomar el desarrollador por ejemplo: En este punto dependiendo de las exigencias que tenga el dataset

se puede aumentar el stack o disminuirlo para encontrar con cual hace mejor la tarea o que sea más eficiente. Aquí se pueden tener varias combinaciones como por ejemplo 3 convoluciones seguidas y solo un pooling, esta selección siempre queda a discreción del desarrollador. Esto se puede repetir n veces durante el proceso, toda esta parte corresponde a la extracción de imágenes.

- $INPUT \rightarrow [conv \rightarrow relu]^n \rightarrow Pool? \rightarrow [fc \rightarrow relu]^k \rightarrow fc$
- $3 \geq n \geq 0, \quad m \geq 0, \quad k \geq 0$
- Stacks

En esto el m a utilizar debe ser minimo 1.

Qué arquitecturas preferimos?

Es importante que prefiera stacks con convoluciones pequeñas. Suponga que tiene un stack de tres convoluciones de 3x3, donde:

- Cada neurona de la segunda convolución que tiene una vista de la imagen de entrada de 3x3
- Cada neurona de la segunda convolución que tiene un campo receptivo de 3x3, al estar conectada a la neurona anterior puede observar un total de 5x5 de la imagen.
- Cada neurona de la tercera convolución con campo receptivo 3x3 al estar conectada con la anterior puede ahora observar un total de 7x7 de la imagen.

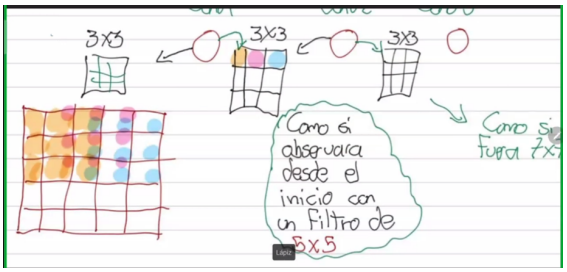

FIGURE 2. Stack de 3 convoluciones.

Como pequeña explicación de la imagen anterior, tenemos que al hacer el proceso de deslizar el kernel por la imagen vamos a ir trayendo toda esa información de 3 pixeles. Posteriormente cuando se aplique todo el proceso de la convolución se tendrá el acceso al 5x5. En la imagen la información que se recolecto son los pixeles amarillos y rosados, y los azules son los que hacen la dimensión de la imagen, así es como se obtiene el 5x5. Posteriormente si se aplica el mismo proceso con el siguiente se obtiene un 7x7. En relación con lo anterior suponga que en lugar de tener ese stack, prefiere una única convolución con un campo receptivo de 7x7, con esto tenemos:

- -Resultado de salida el mismo, que en el ejemplo anterior.
- Presenta desventajas, conectada con la anterior puede ahora observar un total de 7x7 de la imagen.

Las neuronas serán computadas con una función lineal directamente con la entrada.

- En el otro caso, el stack contiene no linealidad (*ReLU*) que hace los *features* más expresivos.

Cantidad de parámetros

Supongamos que todos los volúmenes tienen C canales:

- La convolución 7×7 tendrá:

$$C \times (7 \times 7 \times C) = 49C^2 parámetros$$

- En cambio, si usamos un stack de tres convoluciones 3×3:

$$3 \times (C \times (3 \times 3 \times C)) = 27C^2 parámetros$$

Algunas reglas

- El tamaño de la capa de entrada (que contiene la imagen) debería ser divisible por 2 muchas veces. Ejemplos comunes:
  - 32 (CIFAR-10)
  - 64 o 96 (STL-10)
  - 224 (ImageNet)

  Esto es útil para reducir imágenes a la mitad sin complicaciones con el *padding*.
- Las capas convolucionales deberían usar campos receptivos pequeños:
  - 3×3 o a lo mucho 5×5
  - Stride = 1
- **Padding** de ceros, para que no se altere el tamaño espacial. Ejemplos:
  - $F = 3, P = 1$
  - $F = 5, P = 2$
- Para la capa de *pooling*, configuraciones comunes:
  - Max-pooling con $F = 2, S = 2$
  - También es común: $F = 3, S = 2$ (requiere más cuidado con las dimensiones)
- Es raro ver filtros mayores a 3×3

Arquitecturas

LeNet: LeNet, desarrollada por Yann LeCun, fue una de las primeras redes convolucionales exitosas y, aunque hoy se combina con arquitecturas más modernas, sigue siendo utilizable. Esta es una arquitectura de red neuronal convolucional diseñada originalmente para el reconocimiento de dígitos escritos a mano.

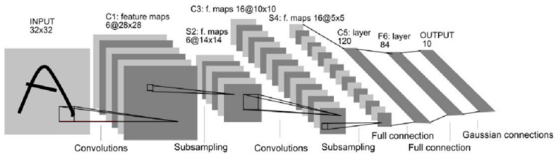

FIGURE 3. Arquitectura LeNet.

AlexNet: Es considerablemente más grande que LeNet, con más capas y filtros más complejos. Utiliza convoluciones max pooling y capas conectadas para extraer características y clasificar imágenes, reduciéndolas progresivamente hasta dimensiones como 5x5.

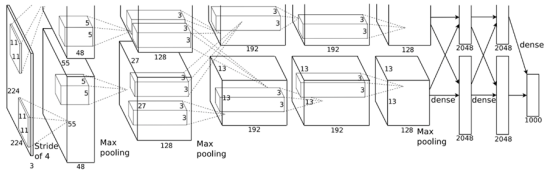

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

FIGURE 4. Arquitectura AlexNet.

ZFNet: Mejoras sobre AlexNet, con filtros más pequeños.

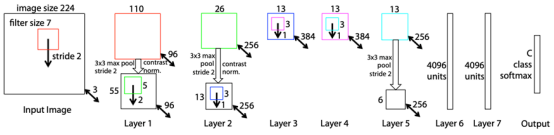

Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

FIGURE 5. Arquitectura ZFNet.

Diferencias entre ZFNet-AlexNet: AlexNet uso filtros de 11x11, 5x5, 3x3, mientras que ZFNet en la primera capa utiliza 7x7 y el resto 3x3.

GoogleNet: Reducción de parámetros con módulos Inception

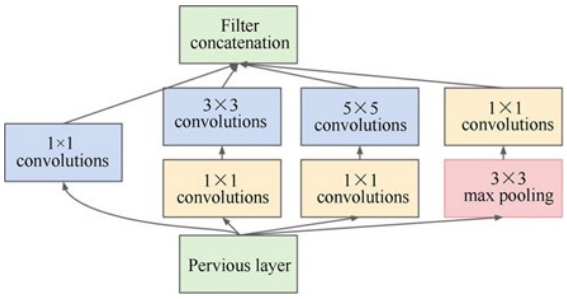

FIGURE 6. Arquitectura GoogleNet.

Reduccion de parámetros 4M (AlexNet con 60M de parámetros). Aquí crean el modelo de inception que básicamente lo que hace es procesar la capa anterior, pero con diferentes convoluciones, todo eso se hace para producir la imagen de salida de la capa.

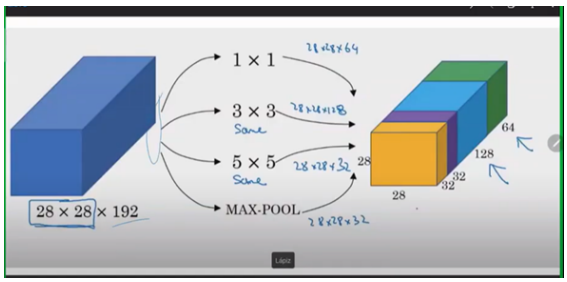

FIGURE 7. Arquitectura GoogleNet.

VGG16: Uso de muchas capas pequeñas. Aplica Deep learning para obtener mejores resultados.

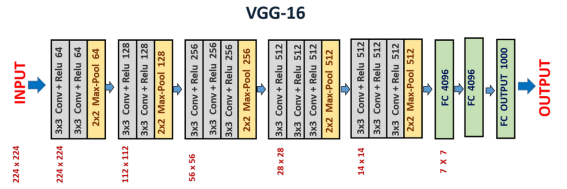

FIGURE 8. Arquitecturas – VGG16.

ResNet: En el centro se tiene un Deep learning muy profundo de 34 capas que da buenos resultados, pero tiene el problema que cuando se da el propagation se pierde mucho el gradiente, por lo que puede ser que las primeras capas no se optimicen como quisiéramos.

DensetNet: DenseNet es una arquitectura de red neuronal, en este se optimiza el volumen, en esta cada capa se conecta con todas las anteriores, lo que mejora la reutilización de características y reduce el problema del desvanecimiento del gradiente. Además, este modelo puede ser mas ligero que el anterior.

Visualización del Aprendizaje en Redes Convolucionales

Problemas de las redes neuronales

Cuando se quiere explicar cómo funciona una red neuronal convolucional al extraer información de una imagen, por ejemplo para detectar enfermedades como problemas pulmonares o COVID mediante un escaneo, se recurre a distintas técnicas de visualización. Algunos de los problemas comunes incluyen:

- **Explicabilidad**: Generalmente son vistas como una *caja negra*.
- Los *features* aprendidos no son interpretables directamente.

Visualización de Filtros

Esto es similar a las representaciones previas, pero ahora se enfocan en los filtros para observar qué es lo que la red ha aprendido.

- En este enfoque, se visualizan los pesos de la red.
- Se interpretan con mayor facilidad en la primera capa convolucional, ya que:
  - Está conectada directamente con los datos.
  - También es posible visualizar filtros más internos.
- Se deben mostrar filtros sin patrones de ruido.
  - Si la visualización presenta ruido, podría indicar que el modelo no fue entrenado correctamente.

Embeddings

- Gradualmente transformamos imágenes a una interpretación vectorial.
  - Sus clases son separables por un clasificador.
- Se pueden transformar a una representación de dimensión menor, manteniendo las distancias similares a las de la representación original de mayor dimensión.

Aquí, la red convolucional se reduce progresivamente hasta generar un vector. Estos *embeddings* son la salida de alguna de las capas internas.

Un buen modelo debe generar buenos *embeddings*; uno malo generará *embeddings* poco útiles.

t-SNE: es una técnica (y también un artículo científico) que explica en profundidad este proceso de representación en espacios de menor dimensión.

Ocultar partes de la imagen

En esta técnica, se analiza la imagen ocultando secciones específicas para observar en qué parte se enfoca la red neuronal. Por ejemplo, se puede superponer un cuadro sobre la imagen para identificar qué región está siendo procesada por la red.

Este método también permite detectar si la red se está enfocando en regiones irrelevantes o inesperadas para la tarea de clasificación o reconocimiento.

Mapas de activación

Los mapas de activación permiten visualizar los vectores donde la red neuronal realiza mayor activación. Estos se representan como un "*mapa de calor*" que resalta las neuronas más activas durante el procesamiento.

Esto ayuda a comprender en qué zonas de la imagen se está enfocando la red y puede proporcionar evidencia visual sobre la interpretación que hace el modelo. Por ejemplo, al clasificar una imagen de un pulmón, un buen mapa de activación debería resaltar las zonas correspondientes a posibles lesiones o anomalías.

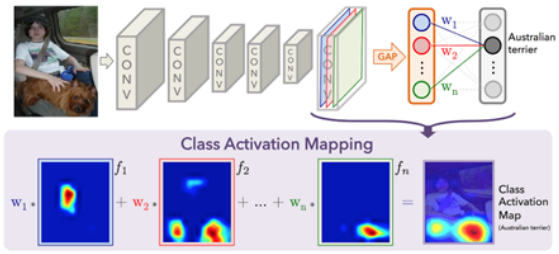

FIGURE 9. Mapas de activación.

Autoencoder

Los *autoencoders* se basan en la autoconstrucción de una imagen. Es decir, se parte de una imagen, se le aplican convoluciones y se genera un vector de características. A partir de este vector, se reconstruye la imagen original.

Unsupervised Learning

Normalmente, en la mayoría de la literatura, este enfoque se reconoce como *no supervisado*, aunque en realidad sí existe cierto grado de supervisión. Sin embargo, se le clasifica como no supervisado porque:

- No utiliza etiquetas.
- Se compara el resultado contra las mismas entradas X.

Componentes de un Autoencoder

Un *autoencoder* es una arquitectura diseñada para reconstruir los datos de entrada después de ser codificados en un vector o espacio latente. Se compone de tres partes principales:

- **Encoder**: Extrae las características de la imagen.
- **Latent Space**: Espacio donde el vector de características es distribuido.
- **Decoder**: A partir del *latent space*, se reconstruye la imagen original.

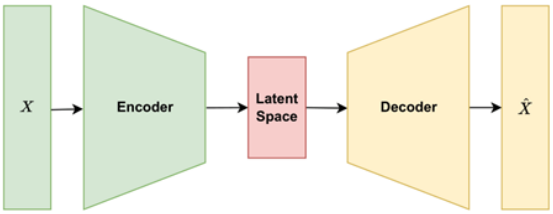

FIGURE 10. Autoencoder.

Tareas que podemos resolver con Autoencoders

Reducción de dimensionalidad

- Permite representar la entrada en un vector más compacto.
- Ofrece una representación más poderosa en comparación con PCA.
- Permite reconstruir la información con menor pérdida.

Detección de anomalías

- Se entrena para la reconstrucción de datos de una tarea tomando en cuenta su estado positivo, por ejemplo:
  - Transferencias bancarias correctas.
  - Registros de alta fidelidad.
- Aprende la representación latente de casos positivos.
- Si una transferencia es suficientemente distinta a la normal, el *autoencoder* (en particular su *encoder*) generará un vector latente deficiente.
- Su función de pérdida será muy alta al momento de la reconstrucción.

Procesamiento de imágenes

El *autoencoder* también puede utilizarse para el procesamiento de imágenes. Por ejemplo, puede entrenarse para que el modelo aprenda a eliminar ruido de las imágenes, entre otras tareas, como:

- Comprensión de imágenes
- Reducción de ruido
- *Super resolution*

Super Resolution

- Crear imágenes de alta resolución a partir de imágenes de baja resolución.
- Reconocimiento facial en imágenes de baja resolución.

3: En el Tec Digital se encuentra un *notebook* con el nombre **CNN-GPU**, el cual puede ser un buen punto de partida para el Proyecto 2.