Apuntes Semana 4 - 13/3/2025

Kenneth Chacón Rivas 2020054320

***Abstract*— Estos apuntes corresponden a la primer clase de la Semana 15 del curso de Inteligencia Artificial, impartida el día 03 de junio de 2025. En la cual se explicaron los detalles de la Tarea 3 y el Proyecto 3. En la tarea se debe construir un asistente conversacional basado en los apuntes del curso, usando técnicas de recuperación y generación de información. En el proyecto se trabaja con imágenes de mariposas aplicando autoencoders, agrupamiento, reducción de dimensionalidad y transferencia de conocimiento para clasificarlas. Además, se repasaron conceptos clave para facilitar el desarrollo de ambos trabajos.**

I. Introducción

En esta clase se dedicó el tiempo a explicar dos entregables importantes: la Tarea 3 y el Proyecto 3. Primero se describió paso a paso lo que debe hacerse en la tarea, incluyendo el uso de documentos PDF como fuente de conocimiento, el almacenamiento en bases vectoriales y la implementación de herramientas para responder preguntas mediante un agente conversacional. También se comentó sobre la interfaz, el uso de herramientas como Streamlit, y los criterios de evaluación.

Luego se presentó el Proyecto 3, enfocado en la clasificación de imágenes de mariposas aplicando autoencoders y técnicas complementarias como reducción de dimensionalidad, agrupamiento y transferencia de conocimiento. Se explicó cómo entrenar modelos con o sin etiquetas, agregar ruido a los datos, visualizar representaciones latentes y reutilizar el encoder para mejorar el rendimiento de clasificadores.

Además, se repasaron conceptos clave como PCA, t-SNE, K-means, quantization, y las diferencias entre modelos entrenados desde cero y aquellos que reutilizan conocimiento previo. Estos temas sirvieron como base para entender mejor cómo abordar los desafíos del proyecto y la tarea, y para facilitar su desarrollo de manera ordenada y efectiva.

II. Tarea 3: Asistente Conversacional con RAG y Agentes

Esta tarea, correspondiente al **6.66%** de la nota final, consiste en desarrollar un asistente conversacional capaz de responder preguntas sobre los apuntes del curso de Inteligencia Artificial. El sistema debe combinar una arquitectura de tipo RAG (Retrieval-Augmented Generation) con un agente que utiliza herramientas (*tools*) gestionadas por un modelo de lenguaje (LLM).

Paso a paso del proceso

1) **Recolección de documentos:** Reunir todos los apuntes del curso en formato PDF. Estos documentos constituirán la base de conocimiento del asistente.
2) **Preprocesamiento y extracción de metadatos:** Extraer información clave de cada documento, como:
   - Nombre del archivo.
   - Fecha.
   - Autor.
   - Semana o sesión correspondiente.

   Estos metadatos permiten que el asistente pueda responder a consultas como "¿Quién elaboró el apunte de la semana 8?"
3) **Generación de embeddings:** Dividir los textos en fragmentos (*chunks*) y generar vectores de embeddings con el modelo `text-embedding-3-small` de OpenAI. Para conservar el contexto, se recomienda emplear solapamiento (*overlap*) entre los fragmentos.
4) **Almacenamiento en base de datos vectorial:** Almacenar los vectores generados en FAISS de forma local. Esto produce un archivo `.pkl` que puede reutilizarse en futuras ejecuciones.
5) **Implementación de herramientas:**
   - `racktool.py`: accede a la base FAISS para recuperar fragmentos relevantes.
   - `internettool.py`: permite búsquedas web si se requiere información adicional.
6) **Definición del prompt del agente:** Crear un prompt claro para el modelo LLM (sugerido: `gpt-3.5-turbo-0125`), en el que se especifique:
   - Que las respuestas deben basarse en los apuntes.
   - Que `racktool` debe utilizarse para consultas relacionadas con los documentos.
   - Que `internettool` solo debe usarse si se solicita explícitamente.
7) **Orquestación del agente:** El agente debe seleccionar dinámicamente la herramienta más adecuada según el tipo de consulta. También debe mantener el contexto de la conversación dentro de una misma sesión.
8) **Desarrollo de la interfaz:** Crear una interfaz web funcional utilizando herramientas como `streamlit_agent`, de forma que la interacción con el asistente sea clara e intuitiva.
9) **Evaluación de funcionalidad:** Verificar que el sistema pueda responder con precisión a preguntas sobre:
   - Autores de los apuntes.
   - Fechas y semanas específicas.
   - Temas y contenidos tratados.
   - Resúmenes y observaciones dentro de los documentos.

Consideraciones importantes

- Las búsquedas en Internet deben realizarse únicamente si el usuario lo solicita explícitamente.

- El asistente debe recuperar información relevante siempre que esté presente en los documentos. No debe emitir respuestas erróneas ni indicar ausencia de información si esta existe.
- El modelo sugerido tiene un costo de uso bajo. En caso de limitaciones económicas, se recomienda buscar soluciones con alternativas open source o contactar al docente responsable.

Fecha de entrega: Jueves 12 de junio de 2025, 11:59 p.m.

A continuación se muestra un diagrama que resume visualmente el funcionamiento del sistema RAG implementado en esta tarea, desde la indexación de documentos hasta la generación de respuestas por parte del modelo de lenguaje:

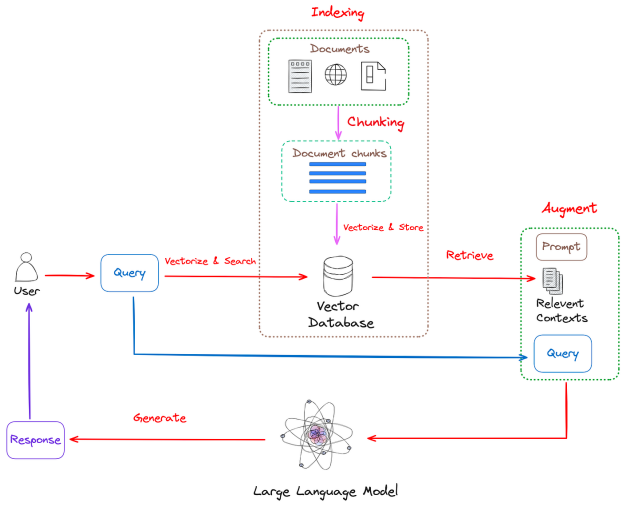

Fig. 1. Diagrama del flujo del sistema RAG.

III. Tercer Proyecto: Clasificación de Mariposas mediante Transfer Learning y Autoencoders

Este proyecto, con un valor del **25%** de la nota final y entrega programada para el **24 de junio de 2025**, tiene como objetivo aplicar técnicas de aprendizaje no supervisado y transferencia de conocimiento para construir clasificadores multiclase que identifiquen especies de mariposas. La estrategia principal consiste en entrenar autoencoders que aprendan representaciones latentes robustas a partir de imágenes, incluso sin etiquetas, y utilizar esas representaciones como base para clasificadores.

Contexto general

Se utiliza el conjunto de datos *Butterflies - 100 Species*, que contiene aproximadamente 130 imágenes por clase. Para este proyecto se establecen las siguientes consideraciones:

- Utilizar únicamente las 30 clases con mayor cantidad de imágenes, descartando las clases con pocos ejemplos.
- Reducir la resolución de las imágenes hasta `128x128x3` para facilitar y acelerar el entrenamiento.
- Reasignar aproximadamente 20 imágenes del conjunto de entrenamiento al conjunto de prueba, dado que el conjunto de prueba original es limitado.
- Utilizar Google Colab si no se cuenta con acceso a GPU local.

Objetivos principales

El proyecto busca evaluar dos hipótesis experimentales:

- Entrenar un autoencoder con imágenes no etiquetadas puede mejorar el rendimiento de un clasificador posterior cuando se dispone de una cantidad limitada de datos etiquetados.
- Incorporar ruido en las imágenes (por ejemplo, ruido tipo *Salt & Pepper*) puede favorecer la generación de representaciones latentes más robustas, facilitando la organización de los datos o la formación de agrupamientos coherentes.

Para abordar estos objetivos se plantean dos experimentos complementarios.

Herramientas requeridas

- **PyTorch Lightning:** Se debe estructurar el modelo mediante las clases `LightningModule` y `LightningDataModule`, implementando los métodos `training_step`, `validation_step`, `test_step` y `configure_optimizers`. Es obligatorio utilizar el callback `EarlyStopping`.
- **Hydra:** Se utiliza para gestionar configuraciones desde archivos `.yaml`, permitiendo definir valores como tamaño de lote, número de épocas, tipo de arquitectura, entre otros.
- **Weights & Biases (wandb):** Permite llevar un seguimiento detallado de métricas de entrenamiento, visualización de imágenes reconstruidas, precisión, pérdida y más.

Estructura del código

- Todo el desarrollo debe centralizarse en un único `Notebook`, que sirva como punto de entrada principal al proyecto.
- Es posible usar módulos auxiliares como `model.py`, `train.py`, etc., pero todo debe ejecutarse claramente desde el notebook principal.
- Se debe garantizar la reproducibilidad entre ejecuciones, utilizando semillas aleatorias fijas y configuraciones bien documentadas.

Experimento 1: Transferencia de conocimiento

Este experimento evalúa si un encoder aprendido por un autoencoder puede mejorar el rendimiento de modelos clasificadores.

División de los datos:

- Primera configuración: 70% del conjunto sin etiquetas y 30% con etiquetas.
- Segunda configuración: 90% sin etiquetas y 10% con etiquetas.

Pasos a realizar:

1) Entrenar un autoencoder con arquitectura **U-Net**, empleando únicamente imágenes sin etiquetas. Se requiere que la arquitectura incluya *skip connections*.
2) Utilizar el encoder entrenado para construir tres clasificadores distintos:

- **Modelo A:** Entrenado desde cero, sin encoder preentrenado.
- **Modelo B1:** Utiliza el encoder del autoencoder, congelado (sin actualización de pesos).
- **Modelo B2:** Utiliza el encoder preentrenado, pero ajusta sus pesos durante el entrenamiento.

3) Evaluar el rendimiento de los modelos empleando métricas como precisión y pérdida.
4) Aplicar **quantization** para reducir el tamaño de los modelos, evaluando:
   - **Latencia:** tiempo requerido para realizar predicciones.
   - **Tamaño del modelo.**
   - **Precisión antes y después del proceso de cuantización.**

Experimento 2: Autoencoder con ruido (denoising)

Este experimento explora si añadir ruido mejora la capacidad del modelo para aprender representaciones internas más sólidas.

- Se añade ruido tipo **Salt & Pepper** a las imágenes de entrada.
- El autoencoder se entrena para reconstruir las versiones limpias de las imágenes.
- Se aplica `t-SNE` sobre las representaciones latentes para visualizar posibles agrupamientos por clase.
- Se ejecuta el algoritmo `K-means` sobre los vectores latentes para formar clusters.
- Se deben presentar al menos 10 imágenes de 5 clusters distintos, analizando si tienen coherencia visual o temática.

Entregables

- **Notebook principal:** Incluye todo el desarrollo del proyecto, resultados, visualizaciones y análisis.
- **Informe en formato artículo científico** elaborado en LaTeX, con secciones como: *Abstract*, *Introducción*, *Metodología*, *Resultados*, *Discusión*, *Conclusiones* y *Referencias*.
- **Código fuente**, archivos de configuración `.yaml` y pesos entrenados.

Fecha de entrega: Martes 24 de junio de 2025, 11:59 p.m.

Repaso de conceptos clave

Autoencoders

Un autoencoder es una red neuronal compuesta por dos partes: un encoder y un decoder. Su objetivo es comprimir una imagen en una representación latente (de menor dimensión) y luego reconstruirla intentando que se parezca lo más posible a la original. Aunque no requiere etiquetas para entrenarse, el autoencoder puede aprender patrones relevantes que representan bien los datos. Las representaciones latentes generadas pueden utilizarse como entrada para tareas como clasificación, detección de anomalías o agrupamiento (*clustering*).

Arquitectura U-Net con skip connections

U-Net es una arquitectura simétrica de autoencoder compuesta por una etapa de compresión (encoder) y una de reconstrucción (decoder). Su característica principal es el uso de *skip connections*, que conectan directamente capas correspondientes entre el encoder y el decoder. Estas conexiones permiten conservar detalles importantes que podrían perderse durante la compresión. Para este proyecto, el uso de U-Net es obligatorio con el objetivo de obtener reconstrucciones más precisas y con mayor calidad.

Denoising Autoencoders

Un denoising autoencoder se entrena con imágenes que han sido alteradas por ruido (como el ruido *Salt & Pepper*) y tiene como objetivo reconstruir la versión original sin ruido. Al enfrentarse a este reto, el modelo aprende a identificar patrones importantes del contenido real de las imágenes, y a ignorar variaciones irrelevantes. Esto conduce a representaciones latentes más robustas, útiles para técnicas de agrupamiento o reducción de dimensionalidad.

Visualización con t-SNE

t-SNE (*t-Distributed Stochastic Neighbor Embedding*) es una técnica no lineal de reducción de dimensionalidad que transforma vectores de alta dimensión en coordenadas de 2D o 3D, conservando la estructura local del espacio original. Su principal aplicación es la visualización de representaciones latentes para detectar agrupamientos o separaciones naturales entre clases. Se recomienda utilizar diferentes colores para distinguir clases reales o clusters generados, lo que facilita la interpretación visual de los resultados.

Clustering con K-means

K-means es un algoritmo de aprendizaje no supervisado que divide los datos en k grupos, asignando cada punto al cluster con el centroide más cercano. Al aplicarse a las representaciones latentes del autoencoder, permite explorar cómo se agrupan naturalmente las imágenes según sus características. Se requiere presentar al menos 10 imágenes representativas por cada uno de 5 clusters formados, y verificar si las agrupaciones tienen coherencia visual, especialmente considerando que se trabaja sin etiquetas.

Transferencia de conocimiento

Este proyecto considera el uso del encoder del autoencoder como base para modelos clasificadores. Se comparan tres estrategias:

- **Modelo A:** Entrenado completamente desde cero, sin uso de encoder preentrenado.
- **Modelo B1:** Utiliza un encoder preentrenado y lo mantiene congelado durante el entrenamiento.
- **Modelo B2:** Utiliza un encoder preentrenado pero lo ajusta mediante *fine-tuning*.

Estas variantes permiten analizar el efecto de aprovechar conocimiento previo y evaluar si la representación aprendida mejora el rendimiento del clasificador, especialmente en contextos con pocos datos etiquetados.

Quantization

La quantization es una técnica de compresión que consiste en reducir la precisión de los pesos del modelo, como pasar de representaciones en punto flotante de 32 bits a enteros de 8 bits. Esto reduce el tamaño del modelo, mejora la velocidad de inferencia y facilita su implementación en dispositivos con recursos limitados. Se debe comparar el desempeño de los clasificadores antes y después de la quantization, midiendo la precisión, el tamaño del modelo y la latencia.

Análisis de Componentes Principales (PCA)

PCA (Análisis de Componentes Principales) es una técnica estadística de reducción de dimensionalidad que transforma los datos originales en un nuevo sistema de coordenadas ortogonales, en el que cada componente principal captura una parte decreciente de la varianza total del conjunto. Se trata de un método lineal que permite conservar las direcciones de máxima variabilidad de los datos, facilitando tanto su análisis como su visualización.

En este proyecto, PCA se aplica a las representaciones latentes generadas por los autoencoders con varios propósitos:

- **Visualización en 2D o 3D:** proyectar las representaciones en un plano o en el espacio tridimensional para identificar si existe una separación natural entre especies, aun sin utilizar etiquetas durante el entrenamiento.
- **Reducción previa a clustering:** reducir la dimensionalidad de los vectores antes de aplicar K-means puede mejorar la calidad de los clusters, reducir el ruido y disminuir el tiempo de cómputo.
- **Comparación con t-SNE:** permite contrastar visualizaciones obtenidas por PCA y t-SNE. Mientras t-SNE es más potente para separar grupos localmente, PCA es más estable, reproducible y útil cuando se desea interpretar la estructura global de los datos.
- **Análisis de varianza explicada:** mediante un gráfico de varianza acumulada (scree plot), se puede determinar cuántos componentes principales conservar, eligiendo aquellos que explican un porcentaje significativo de la variabilidad (por ejemplo, el 95%).
- **Interpretabilidad:** como los componentes principales son combinaciones lineales de las variables originales, se puede analizar su composición para entender qué dimensiones o características contribuyen más a la variabilidad de los datos.

PCA también sirve como validación indirecta del modelo: si las imágenes se agrupan correctamente en el espacio proyectado sin necesidad de etiquetas, eso indica que el autoencoder ha aprendido una representación significativa de los datos.

Indicaciones técnicas

- Utilizar semillas aleatorias fijas para asegurar la reproducibilidad de los experimentos.
- Mostrar visualizaciones de t-SNE y PCA con distinción clara de clases o agrupamientos.
- Incluir reconstrucciones de imágenes, curvas de entrenamiento, ejemplos visuales de clusters y resultados de predicción.
- Registrar el proceso completo en Weights & Biases (wandb), incluyendo métricas, visualizaciones y configuraciones.
- Guardar los pesos entrenados, los archivos `.yaml` de configuración y los resultados finales como respaldo.

IV. Conclusión

La clase permitió aclarar los objetivos, pasos y enfoques necesarios para desarrollar correctamente la Tarea 3 y el Proyecto 3 del curso. En la tarea, se explicó cómo construir un asistente conversacional utilizando los apuntes del semestre como base de conocimiento. En el proyecto, se presentó una estrategia para clasificar imágenes de mariposas utilizando autoencoders, reducción de dimensionalidad, agrupamiento y transferencia de conocimiento.

También se presentaron recomendaciones útiles para facilitar el desarrollo, como dividir el trabajo en partes manejables, mantener un código organizado, visualizar los resultados de forma clara y registrar correctamente los experimentos. Estos lineamientos permiten abordar los entregables con mayor claridad y efectividad.