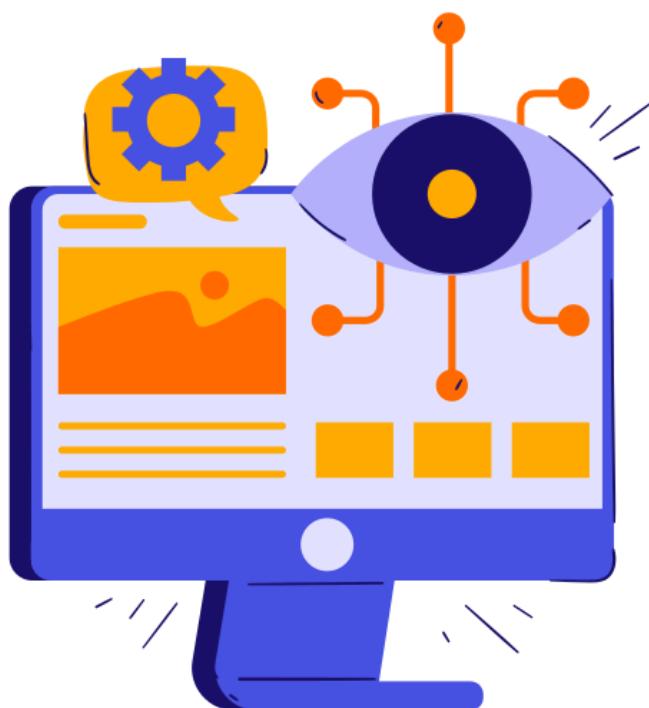




**UNR** Universidad  
Nacional de Rosario

# Trabajo práctico Computer Vision

## Tecnicatura Universitaria en Inteligencia Artificial



### Docentes

- MANSON, Juan Pablo
- LEON CAVALLO, Andrea
- GEARY, Alan
- BRUGÉ, Lucas

### Autor

- Raffaeli Taniel



# Índice

<b>Índice</b>	<b>2</b>
<b>Consignas</b>	<b>3</b>
Introducción	3
Pautas generales del TP	3
Ejercicio 1. Creación de un dataset colaborativo	4
Ejercicio 2. Dataset personal	4
Ejercicio 3. Entrenamiento de modelos	5
Ejercicio 4. Evaluación	6
Aclaraciones adicionales	6
<b>Resolución de los ejercicios</b>	<b>7</b>
Explicación del flujo de datos	7
Nota importante:	8
Realizar primer procesamiento	8
Crear dataset y aumentar datos	9
Entrenamiento del modelo	10
Archivos de utilidad	10
Ejercicio 1	11
Dataset colaborativo	11
Ejercicio 2	11
Dataset personal	11
Ejercicio 3	15
Entrenamiento del modelo	15
Evaluación del modelo	16
<b>Conclusiones generales</b>	<b>22</b>

# Consignas

## Introducción

El objetivo es que el alumno elabore un software de su autoría, que basado en modelos de reconocimientos de objetos y programación, sea capaz de calcular los puntos del envío del juego de truco a partir del reconocimiento de cartas españolas. Para ello deberá recorrer al menos una vez el ciclo de vida completo de un entrenamiento novel, desde el análisis del juego y la problemática, el diseño de la captura de datos, la captura de datos, anotación, selección, aumentación, y elaboración de los diferentes conjuntos de datos. Deberá iterar sobre las rutinas de entrenamientos de el o los modelos seleccionados las veces que fuera necesario, fundamentando las decisiones para iterar, alterar (agregar, quitar, aumentar, etc) los datos, la conformación de los conjuntos de datos, los ajustes a los hiperparámetros y cualquier decisión que justifique cada iteración basándose en los resultados obtenidos.

Además de los resultados obtenidos, se evaluará la presentación del informe, los argumentos y la información presentada en él, la brevedad y la defensa oral en la defensa.

## Pautas generales del TP

Se deberá entregar un link a la carpeta drive que deberá ser privada y compartida a los docentes, la misma deberá contener:

- Informe en pdf que describa el proceso del trabajo realizado, con carátula, introducción, proceso de evaluación de datos.
- Uno o varios archivos de Jupyter con la creación del dataset, el proyecto de entrenamiento, con la ejecución ya realizada, incluyendo las métricas correspondientes.
- El archivo con los pesos del entrenamiento.
- Los archivos del dataset aumentado.
- Otro archivo de Jupyter con el código que aplica la inferencia, deberán usar la estructura de archivos que se encuentra en la última sección de este enunciado.

La fecha límite de entrega de los archivos del proyecto será el domingo 28 de julio a las 23.59, aunque si prefieren presentar en la semana anterior podrán adelantar

tiempos. Posterior a la entrega, se acordarán horarios para la instancia de defensa del mismo.

## Ejercicio 1. Creación de un dataset colaborativo

Cada alumno realizará un aporte individual de al menos 30 imágenes reales de un mazo de cartas españolas, como un aporte a dataset colaborativo:

- Cada imagen debe contener entre 5 y 8 cartas.
- Las imágenes deberán estar en formato jpg o png.
- Deben anotarse las imágenes en formato Yolo.
- Para el nombre de los archivos, se utilizará el formato LEGAJO\_Nombre\_Y\_Apellido\_XX con extensiones (jpg/png y txt, según el caso). XX será un número correlativo de imagen, y el Legajo sin barras o guiones.
- Los archivos se deben subir a un repositorio Drive. Se espera que en dicha carpeta, no haya subcarpetas, solamente los pares de archivos imagen/anotación.
- El acceso será público, para que cualquier alumno pueda usar las imágenes.
- Las imágenes deben ser reales, sin aumentaciones.
- La técnica de bounding box (carta completa o carta parcial) se acordará entre los alumnos.

## Ejercicio 2. Dataset personal

A partir de las imágenes y anotaciones generadas en el Ejercicio 1, el alumno deberá generar su propio conjunto de datos, aumentarlo, y generar las particiones (entrenamiento, validación, prueba) para trabajar en los entrenamientos.

Realizar un script que:

- Valide las anotaciones, verificando que ningún bounding box exceda los límites de la imagen. En caso de hallar anomalías, separar los pares de archivos en una carpeta llamada “label\_errors”.
- Unifique las anotaciones en un formato Yolo único (ya que pueden estar en versiones de 5 valores o de 9 valores por línea, según la versión de Yolo), para lo cual se deben reescribir los .txt en caso que sea necesario. De cualquier manera, las evaluaciones sobre las detecciones que realice el modelo se evaluarán a partir del formato YoloV5 propuesto, donde debe existir un archivo .txt por cada archivo de

image con el mismo nombre, donde cada línea del archivo corresponda a la detección de un objeto. Cada línea, deberá contener los siguientes datos: id\_clase, x\_centro, y\_centro, ancho, alto, certeza.

Explicar:

- Cantidades de imágenes de cada split.
- Los tipos de aumentación utilizados y por qué aplicó determinadas herramientas.
- La composición del dataset en términos de balanceo de clases. Graficar.
- Utilizar la herramienta FiftyOne para visualizar el nuevo dataset generado.

## Ejercicio 3. Entrenamiento de modelos

Se debe generar un notebook que dé cuenta de todos los pasos realizados para entrenar el modelo, tales como la instalación de librerías en el entorno (si fuera necesario), copia o descarga de datos o las indicaciones que permitan repetir el entrenamiento todas las veces que sea necesario.

El notebook también debe mostrar en sus salidas los registros (logs) de entrenamiento que habitualmente los frameworks o las mejores prácticas suelen sugerir, tales como métricas de pérdida general (loss), pérdida de clase (class loss), pérdida de caja delimitadora (bbox loss), precisión, recall, etc. La lista es sólo a modo de ejemplo. Incluir también las gráficas de la evolución del entrenamiento según las epochs, proporcionadas por el propio framework de entrenamiento.

Al final del entrenamiento es requerido contar con las métricas de mAP[0.5:0.95] y mAP50, precisión, recall y F1 Score por clase sobre el conjunto de validación, así como algunas imágenes (no menos de 10) del mismo conjunto con las detecciones del modelo impresas sobre ellas. El objetivo es poder visualizar las detecciones para relacionarlo con el análisis de las métricas y justificar la necesidad de modificar datos y/o hiperparámetros antes de realizar un nuevo entrenamiento del modelo.

Todas las iteraciones de mejora (re-entrenamiento) deben contar con las debidas métricas sobre el total del conjunto de validación para conformar una tabla que formará parte del informe final.

Se recomienda emplear algún método de optimización del modelo para la inferencia (como TensorRT) y dejar registro de los tiempos antes y después de la optimización. Estas métricas se deberán incluir en el informe final. Se valorará la ejecución de la inferencia en diferentes dispositivos, así como en CPU y GPU.

## Ejercicio 4. Evaluación

Se debe ejecutar el notebook de evaluación provisto con el fin de evaluar el modelo sobre el conjunto de prueba y obtener las mediciones para verificar los requisitos de funcionamiento que permiten la evaluación al momento del examen. Se espera que al modificar la ruta de almacenamiento del dataset, se ejecuten todas las funciones que nos permitirán evaluar el trabajo.

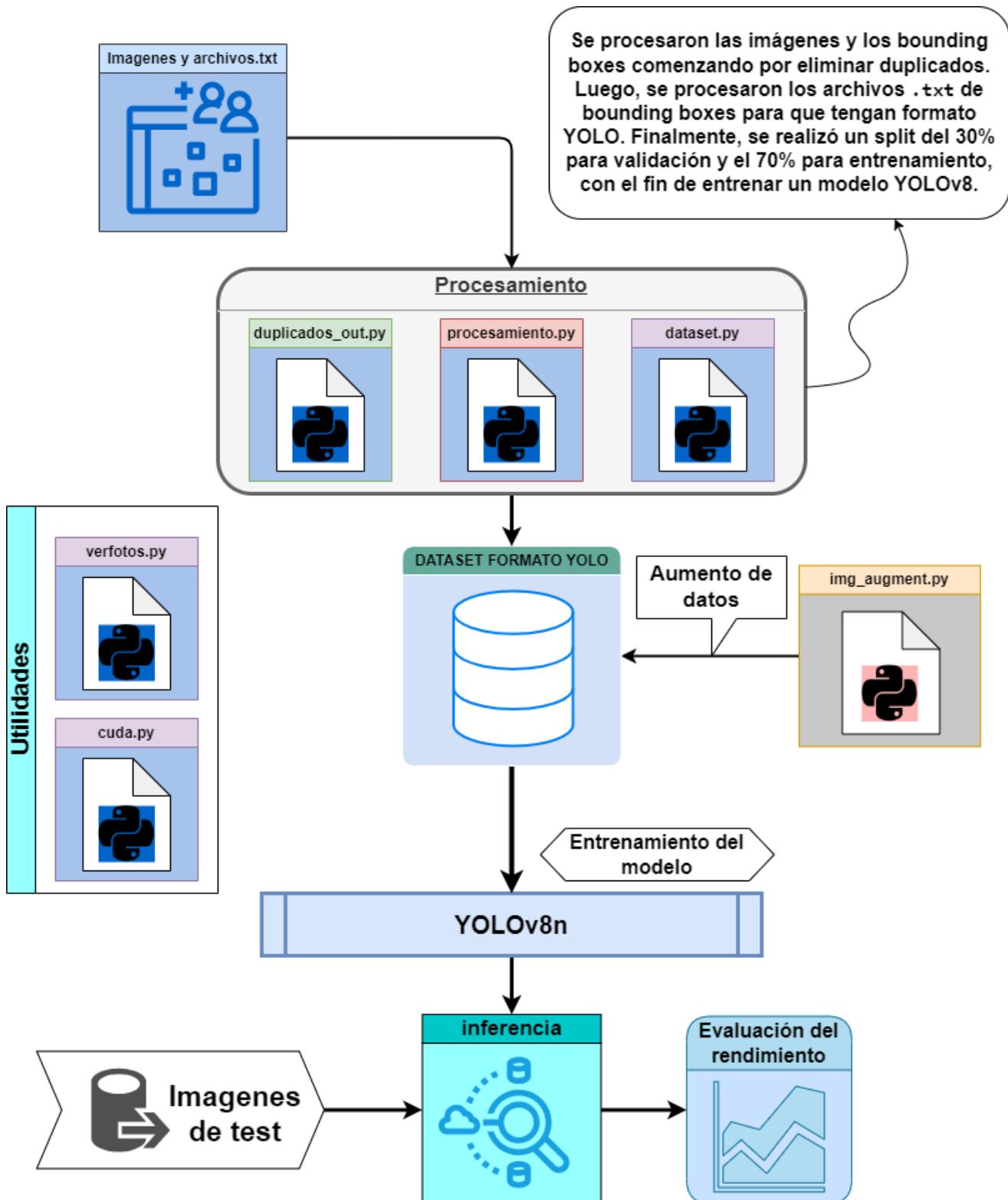
### Aclaraciones adicionales

Tener en cuenta para el cálculo del envido que 10, 11 y 12 valen 0 cuando se cantan solos.

# Resolución de los ejercicios

## Explicación del flujo de datos

Gráfico del flujo de datos



## Nota importante:

Para la entrega, se solicitaba que el código se presentara en notebooks de Colab. Aunque el desarrollo del trabajo práctico se realizó en archivos .py, también he incluido notebooks de Colab que replican las mismas funcionalidades que los archivos de Python. A continuación, se presenta una lista de los notebooks de Colab junto con los archivos .py correspondientes.

- **procesamiento.ipynb**: contiene (duplicados\_out.py y procesamiento.py)
- **dataset.ipynb**: contiene (dataset.py y img\_augment.py)
- **main\_yolo.ipynb**: contiene (main\_yolo.py y best\_model.py)
- **00.run\_envido.ipynb**

## Realizar primer procesamiento

### Eliminación de duplicados ([duplicados\\_out.py](#))

La función **hash\_archivo** calcula el hash MD5 del contenido de un archivo. Luego, el código recorre todos los archivos en el directorio, calculando el hash de cada archivo con las extensiones especificadas (jpg, jpeg, png, txt). Si el hash de un archivo ya existe en el diccionario hashes, se considera un duplicado y se elimina. Si el hash no existe, se agrega al diccionario.

### Procesamiento de archivos de anotaciones e imágenes ([procesamiento.py](#))

La función **process\_annotations** recorre los archivos de anotaciones, buscando las correspondientes imágenes, y convierte las anotaciones al formato YOLO si es necesario. También elimina los archivos de anotaciones si no se encuentra la imagen correspondiente.

La función **process\_annotations** también utiliza:

- La función **is\_yolo\_format** verifica si una línea de anotación está en el formato YOLO adecuado.
- La función **convert\_to\_yolo\_format** convierte las anotaciones que no están en el formato YOLO a dicho formato, calculando los valores del centro, ancho y alto de las bounding boxes a partir de las coordenadas de las esquinas.

La función **validate\_images\_and\_annotations** verifica que cada imagen tenga su archivo de anotación y viceversa, eliminando cualquier archivo huérfano que no tenga su correspondiente par.

## Crear dataset y aumentar datos

### Creación del dataset ([dataset.py](#))

Este código crea un dataset estructurado para YOLO a partir de imágenes y anotaciones existentes. Define rutas para los datos y la configuración del dataset, incluyendo la lista de clases y el número de clases.

Luego, crea las carpetas necesarias para almacenar las imágenes y anotaciones divididas en conjuntos de entrenamiento y validación. Guarda la configuración en un archivo YAML y los nombres de las clases en un archivo de texto.

Recoge todos los archivos de anotaciones y los divide en conjuntos de entrenamiento y validación, moviendo las anotaciones correspondientes a las carpetas respectivas. Finalmente, mueve las imágenes correspondientes a cada conjunto de anotaciones a sus carpetas designadas.

### Realizar aumento de datos con Albumentations ([img\\_augment.py](#))

Este código amplía un dataset YOLO aplicando técnicas de aumentación de datos a las imágenes y sus correspondientes anotaciones. Primero, define las rutas de entrada y salida para las imágenes y las etiquetas.

Luego, crea las carpetas necesarias si no existen. Define una serie de transformaciones de aumentación utilizando la librería albumentations, como ajuste de brillo y contraste, rotación, desenfoque, sombras aleatorias, cambios en la saturación, y desplazamiento, escalado y rotación.

Recorre las imágenes en la carpeta de entrada, leyendo cada imagen y sus anotaciones. Para cada imagen, verifica la validez de las anotaciones y aplica las transformaciones definidas, generando múltiples imágenes aumentadas y sus correspondientes archivos de anotación en formato YOLO.

Las imágenes y anotaciones aumentadas se guardan en las carpetas de salida especificadas. Si una imagen o sus anotaciones son inválidas, la imagen se omite y se registra en una lista de imágenes omitidas. Finalmente, se imprime un mensaje indicando la finalización del proceso y las imágenes que fueron omitidas.

## Entrenamiento del modelo

### Entrenamiento del modelo YOLOv8n ([main\\_yolo.py](#))

Este código configura y entrena un modelo YOLOv8 utilizando PyTorch y la librería Ultralytics. Primero, verifica si CUDA está disponible y muestra el nombre de la GPU en uso.

Luego, define los directorios donde se encuentran las imágenes de entrenamiento y validación, así como la ruta al archivo de configuración data.yaml.

Crea una instancia del modelo YOLOv8 preentrenado y lo entrena usando los datos especificados en data.yaml durante 2 épocas, con un tamaño de lote de 32 y un tamaño de imagen de 640 píxeles.

### Poner a prueba el modelo entrenado ([best\\_model.py](#))

Se cargan los pesos de un modelo YOLOv8 previamente entrenado desde la ruta especificada y lo utilizan para realizar predicciones en imágenes de prueba.

## Archivos de utilidad

### Utilización de GPU ([cuda.py](#))

Este script verifica la disponibilidad de CUDA y muestra la versión de CUDA, la versión de PyTorch, y el nombre de la GPU si está disponible. Además, comprueba y muestra la versión de la biblioteca Ultralytics YOLO.

### Visualizar fotos ([verfotos.py](#))

Este script toma imágenes y sus archivos de etiquetas correspondientes, dibuja los bounding boxes en las imágenes redimensionadas, y luego crea un collage de estas imágenes.

- La función **draw\_bounding\_boxes** dibuja rectángulos y etiquetas de clase en las imágenes.
- La función **create\_collage** organiza varias imágenes en un collage.
- Finalmente, **visualize\_images\_with\_bboxes** selecciona imágenes de un directorio, dibuja los bounding boxes en ellas, y muestra las imágenes en collages de 30 imágenes por lote.

## Ejercicio 1

### Dataset colaborativo

Se realizó un aporte individual por parte de cada alumno, donde se entregaron al menos 30 imágenes reales de un mazo de cartas españolas, como contribución a un dataset colaborativo.

Cada imagen contenía entre 5 y 8 cartas y estaba en formato jpg. Las imágenes fueron anotadas en formato YOLO. Los archivos fueron subidos a un repositorio Drive que no contenía subcarpetas, solo los pares de archivos de imagen y anotación.

El acceso al repositorio fue público, permitiendo a cualquier alumno usar las imágenes. Las imágenes fueron sin aumentaciones, y la técnica de bounding box (carta completa o carta parcial) fue acordada entre los alumnos.

## Ejercicio 2

### Dataset personal

#### Split del dataset

Debido a la cantidad de datos disponibles y a una eficiente aumentación de datos utilizando herramientas avanzadas, se consideró adecuado realizar un split del dataset en un 70% de los datos para entrenamiento (train) y un 30% para validación (validation). Esta división permite un amplio conjunto de datos para entrenar el modelo y otro significativo para validar su desempeño. Además, se determinó que, en este caso, es posible obtener nuevas imágenes para realizar las pruebas (test) del modelo de manera fácil y efectiva, garantizando así una evaluación completa y precisa de su rendimiento en datos no vistos previamente.

#### Aumento de datos

Para realizar la aumentación de datos se utilizó la biblioteca de Python Albumentations. Esta realiza aumentaciones de datos en tareas de visión por computadora. Su objetivo es crear variaciones de las imágenes originales mediante la aplicación de transformaciones aleatorias, lo que ayuda a mejorar la robustez y generalización de los modelos de aprendizaje automático.

### Código utilizado

```
# Define augmentations
transform = A.Compose([
    A.RandomBrightnessContrast(p=1),
    A.RandomRotate90(p=1),
    A.Blur(p=1, blur_limit=(3, 7)),
    A.RandomShadow(p=1),
    A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=1),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.1, rotate_limit=15, p=1)
], bbox_params=A.BboxParams(format='yolo', label_fields=['class_labels']))
```

Transformaciones definidas en el código y lo que hacen

La probabilidad p=1 significa que esta transformación se aplicará siempre.

1. **A.RandomBrightnessContrast(p=1)**: Ajusta aleatoriamente el brillo y el contraste de la imagen.
2. **A.RandomRotate90(p=1)**: Rotea la imagen aleatoriamente en incrementos de 90 grados.
3. **A.Blur(p=1, blur\_limit=(3, 7))**: Aplica un desenfoque a la imagen con un límite de desenfoque entre 3 y 7 píxeles.
4. **A.RandomShadow(p=1)**: Añade una sombra aleatoria a la imagen.
5. **A.HueSaturationValue(hue\_shift\_limit=20,sat\_shift\_limit=30, al\_shift\_limit=20, p=1)**: Ajusta aleatoriamente el matiz (hue), la saturación y el valor de la imagen dentro de los límites especificados.
6. **A.ShiftScaleRotate(shift\_limit=0.1, scale\_limit=0.1, rotate\_limit=15, p=1)**: Desplaza, escala y rota la imagen aleatoriamente. El desplazamiento y la escala se limitan al 10% de la imagen original, y la rotación se limita a 15 grados.

Las anotaciones de las imágenes también se ajustan en consecuencia utilizando A.BboxParams, que asegura que las cajas delimitadoras (bounding boxes) se actualicen para que se mantengan precisas después de aplicar las transformaciones.

## Resultado de las aumentaciones de datos

Imagen Original



Modificaciones de la imagen original

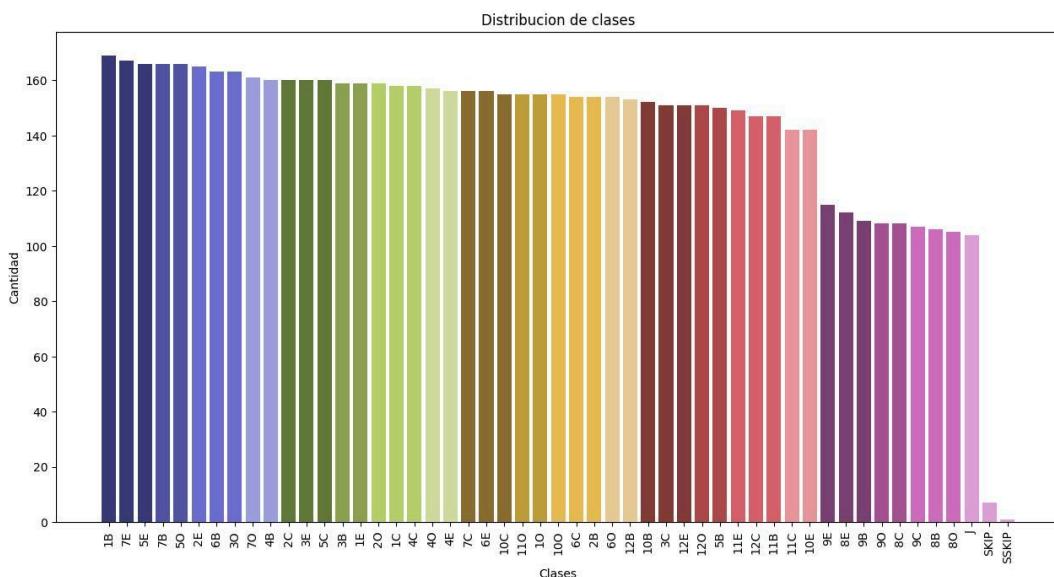


## Visualización del balance de clases

En cuanto a la composición de clases del dataset, las clases de cartas utilizadas en el juego del truco están muy bien equilibradas. Las únicas clases ausentes son el 9, el 8 y el J, que no se utilizan en el juego. Dado que el dataset incluye todas las cartas relevantes para el truco y no contiene cartas innecesarias, se considera que está sumamente balanceado.

Por lo tanto, no es necesario aplicar técnicas de balanceo de clases adicionales para la resolución de este problema. Esta distribución equilibrada asegura que el modelo pueda aprender de manera efectiva sin sesgos hacia clases sobre representadas o sub representadas.

Gráfico del balance de clases



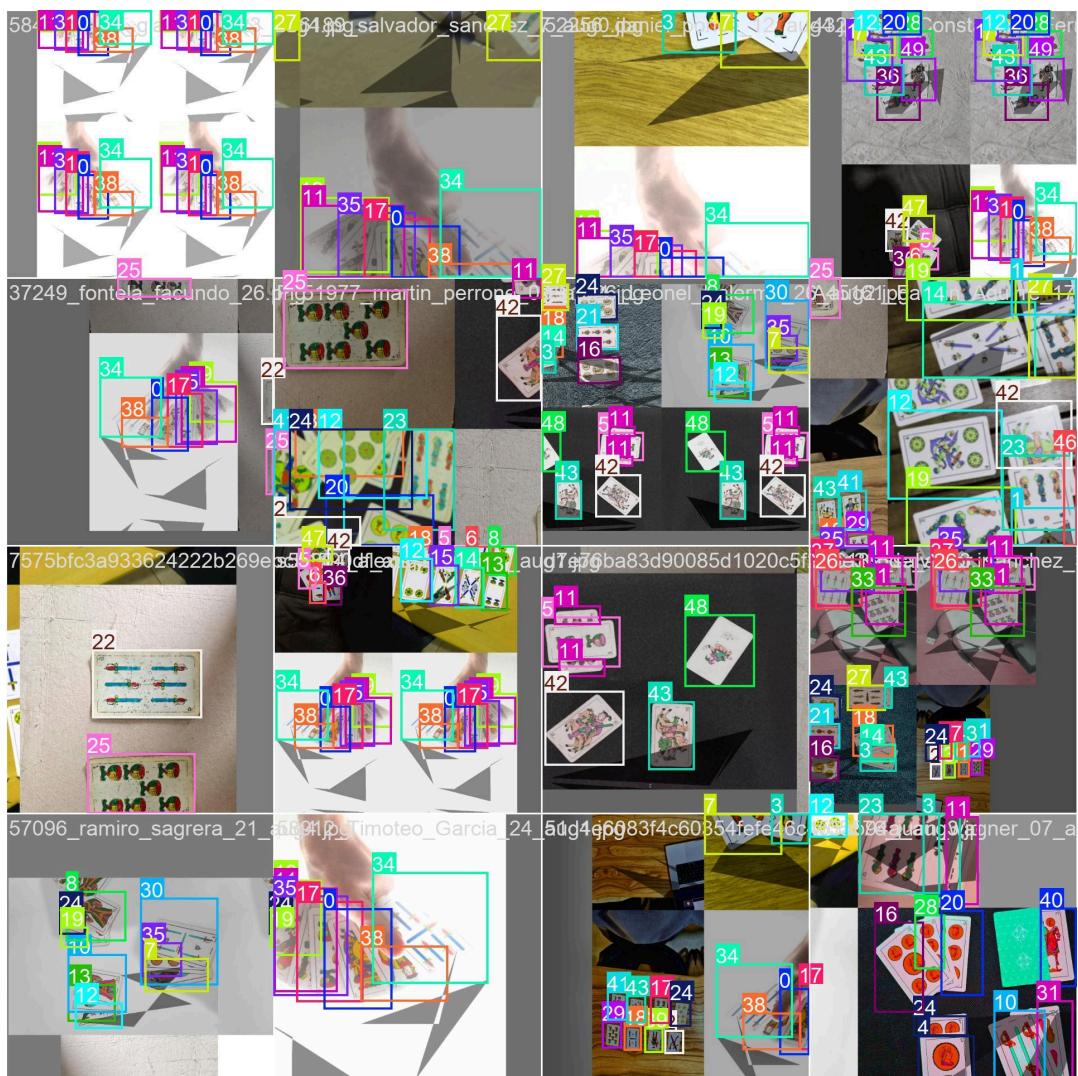
## Ejercicio 3

### Entrenamiento del modelo

El modelo YOLOv8n se entrenó durante un total de 235 épocas. Inicialmente, se había estimado que el entrenamiento se extendería hasta 500 épocas, pero el modelo detuvo su entrenamiento anticipadamente debido a la falta de mejora en las métricas durante las últimas 100 épocas, lo cual es un indicador de convergencia o estancamiento.

El entrenamiento se llevó a cabo utilizando un dataset compuesto por más de 5000 ejemplos en el conjunto de entrenamiento, incluyendo técnicas de aumento de datos para mejorar la robustez y generalización del modelo.

Batch de entrenamiento del modelo



## Evaluación del modelo

Métricas de evaluación del rendimiento del modelo

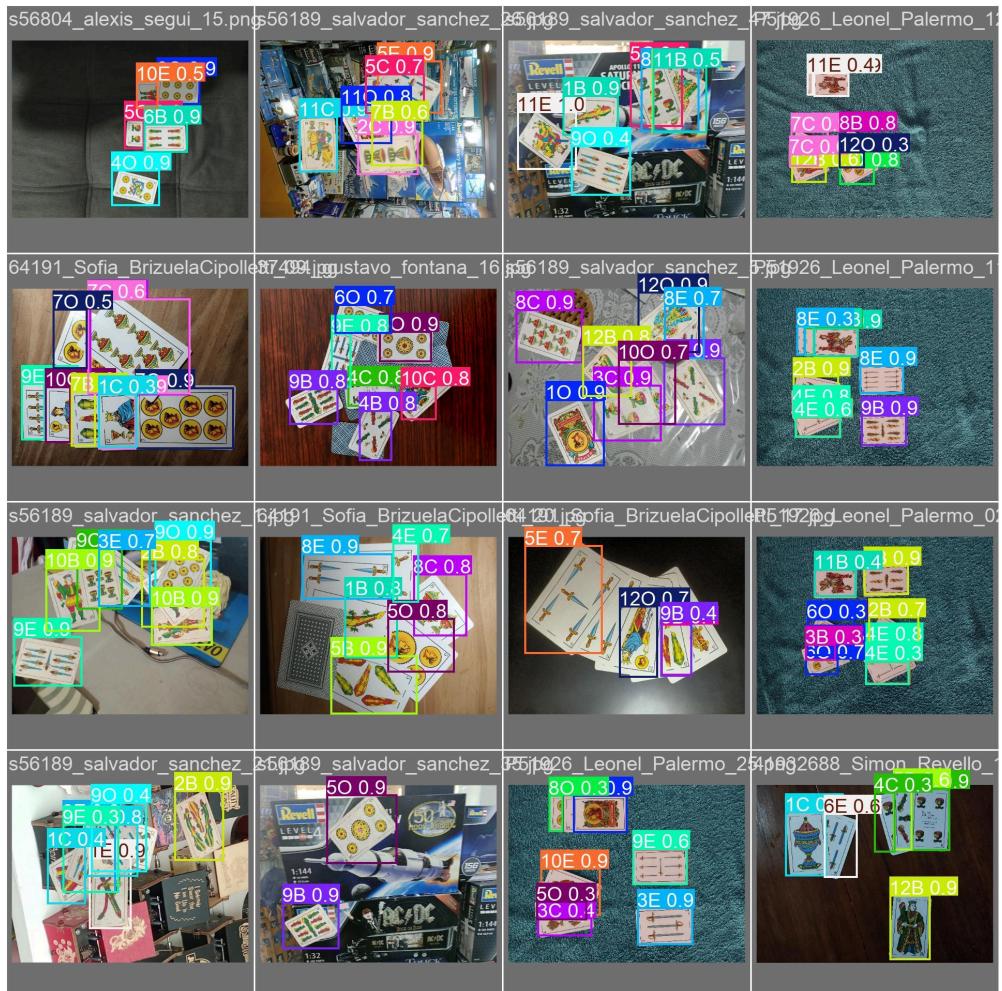
1. **box\_loss:** Mide el error en la predicción de las coordenadas de los bounding boxes. Cuanto más bajo sea este valor, mejor estará el modelo en predecir las posiciones de los objetos.
2. **cls\_loss:** Indica el error en la clasificación de las etiquetas de los objetos dentro de los bounding boxes. Un valor más bajo significa que el modelo está clasificando los objetos con mayor precisión.
3. **dfl\_loss:** Se refiere a la pérdida del "distribution focal loss", una técnica avanzada utilizada para manejar el desequilibrio en la clasificación. Un valor bajo indica que el modelo está manejando bien el desequilibrio de clases y es efectivo en la clasificación.
4. **precision(B):** La precisión mide la proporción de verdaderos positivos entre todos los objetos que el modelo ha identificado como positivos. Un valor alto indica que, de todos los objetos detectados, la mayoría son correctos. En el contexto de la métrica (B), se refiere a la precisión en la clase base.
5. **recall(B):** El recall mide la proporción de verdaderos positivos entre todos los objetos que deberían haber sido identificados. Un valor alto indica que el modelo está detectando la mayoría de los objetos presentes en las imágenes. En (B) se refiere al recall en la clase base.
6. **mAP50(B):** El "mean Average Precision" a un umbral de IoU (Intersection over Union) de 0.5, mide la precisión promedio en la detección de objetos. Un valor más alto indica mejor rendimiento general del modelo. En (B) se refiere al mAP en la clase base.
7. **mAP50-95(B):** Es una versión más estricta del mAP que promedia la precisión a diferentes umbrales de IoU (de 0.5 a 0.95). Ofrece una evaluación más detallada del rendimiento del modelo. Un valor más alto indica mejor rendimiento en una gama más amplia de umbrales.

### Diferencias de métricas entre primera y última época

epoch	precision	recall	mAP50	mAP50-95	val box_loss	val cls_loss	val dfl_loss
<b>1</b>	0.09703	0.16578	0.03031	0.02072	1.3675	3.4965	1.4679
<b>235</b>	0.83104	0.7205 2	0.78092	0.56484	1.1179	0.8942	1.4245

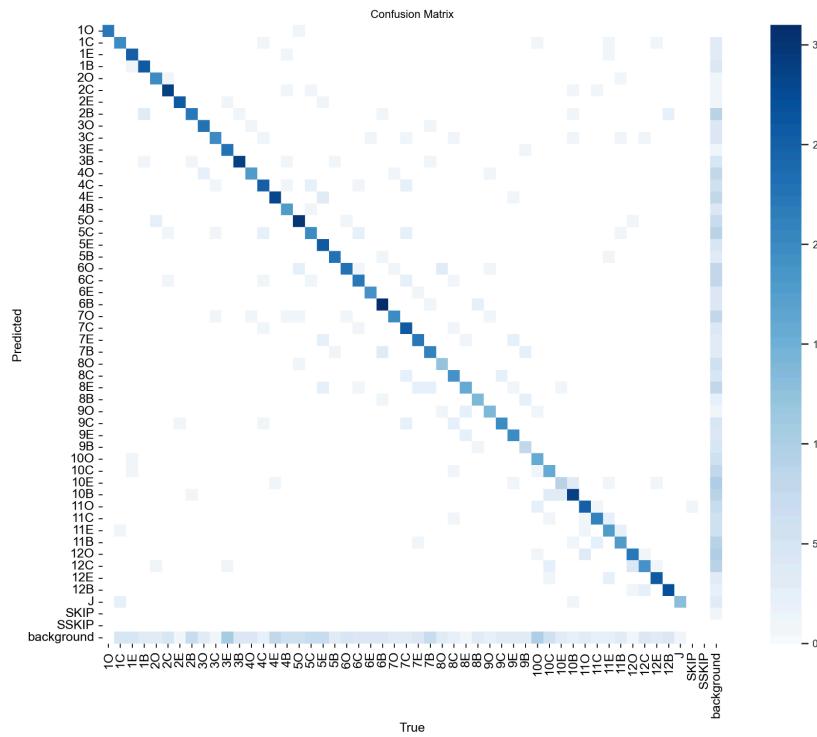
A lo largo del entrenamiento, el modelo ha mejorado significativamente en términos de precisión y recall, indicando una mejor capacidad para detectar y clasificar correctamente los objetos. Las métricas de mAP también han mejorado notablemente, reflejando una precisión general superior en varias condiciones de solapamiento. Las pérdidas en validación han disminuido, mostrando mejoras en la precisión de localización y clasificación.

### Batch de validacion del modelo



En el gráfico de la matriz de confusión, observamos que el modelo logra detectar la mayoría de las clases correspondientes de forma satisfactoria. Esto se evidencia por la alta concentración de valores en la diagonal de la matriz, lo cual indica que las predicciones coinciden en gran medida con las etiquetas verdaderas. Cada celda en la diagonal representa los casos en los que el modelo predijo correctamente la clase del objeto.

Gráfico de matriz de confusión



### Interpretación de los gráficos de curvas

#### **F1\_curve (Curva F1):**

Muestra la relación entre la precisión (precision) y la exhaustividad (recall) en diferentes umbrales de decisión. Un F1 score más cercano a 1 en diferentes umbrales sugiere un mejor rendimiento del modelo.

#### **P\_curve (Curva de Precisión):**

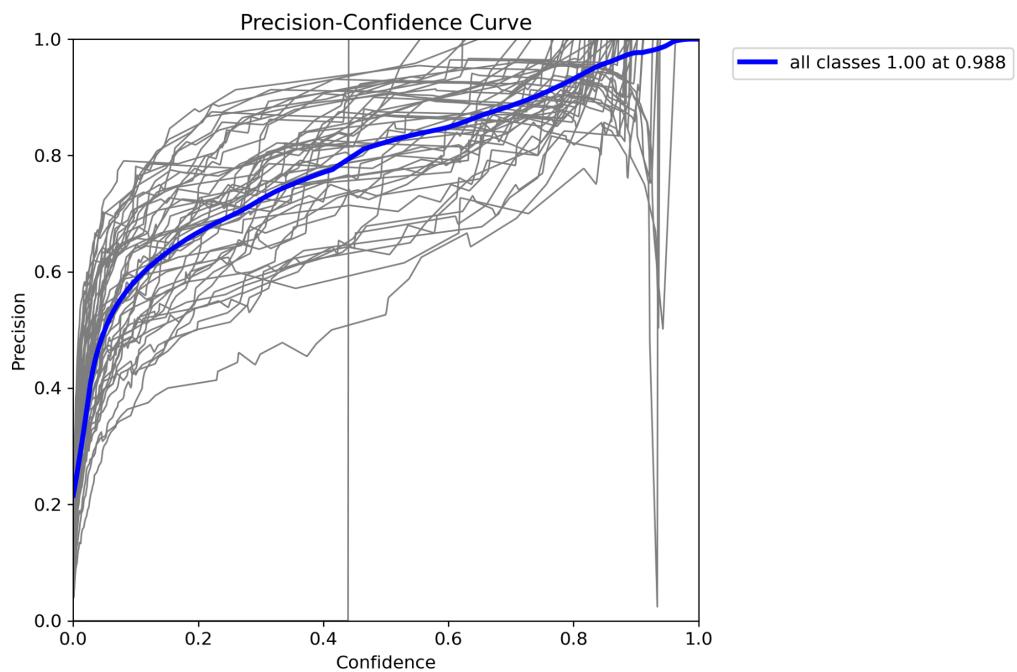
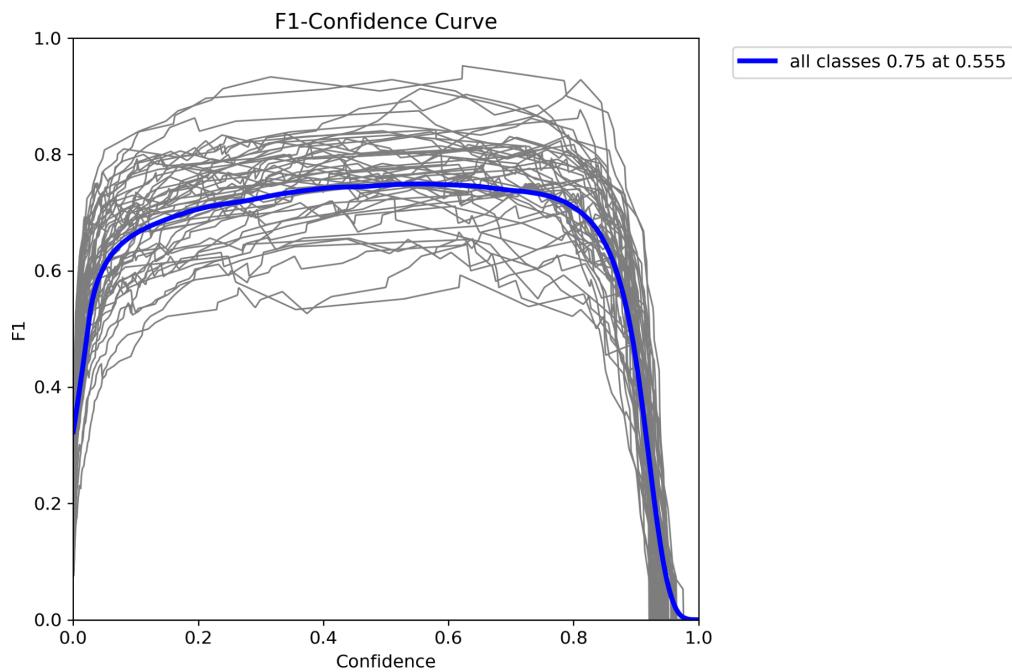
Indica la precisión del modelo en diferentes umbrales de decisión. La curva debe mantenerse lo más cerca posible de 1 en todos los umbrales.

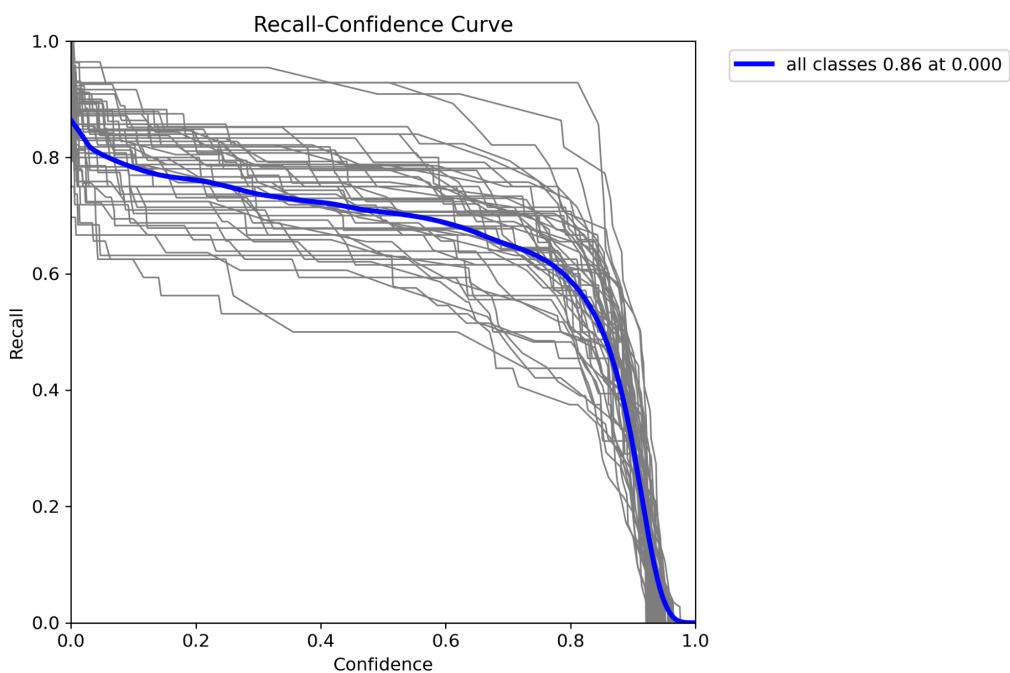
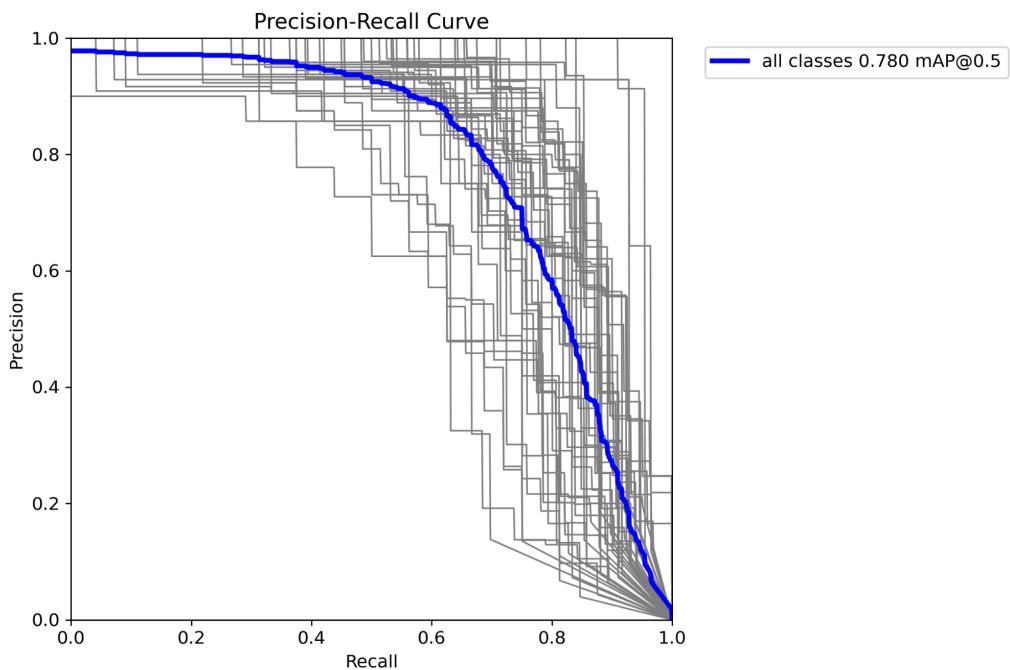
#### **PR\_curve (Curva de Precisión-Recall):**

Muestra la relación entre la precisión y el recall a través de diferentes umbrales. Una curva PR alta y extendida hacia la esquina superior derecha indica un buen equilibrio entre precisión y recall en varios umbrales.

**R\_curve (Curva de Recall):**

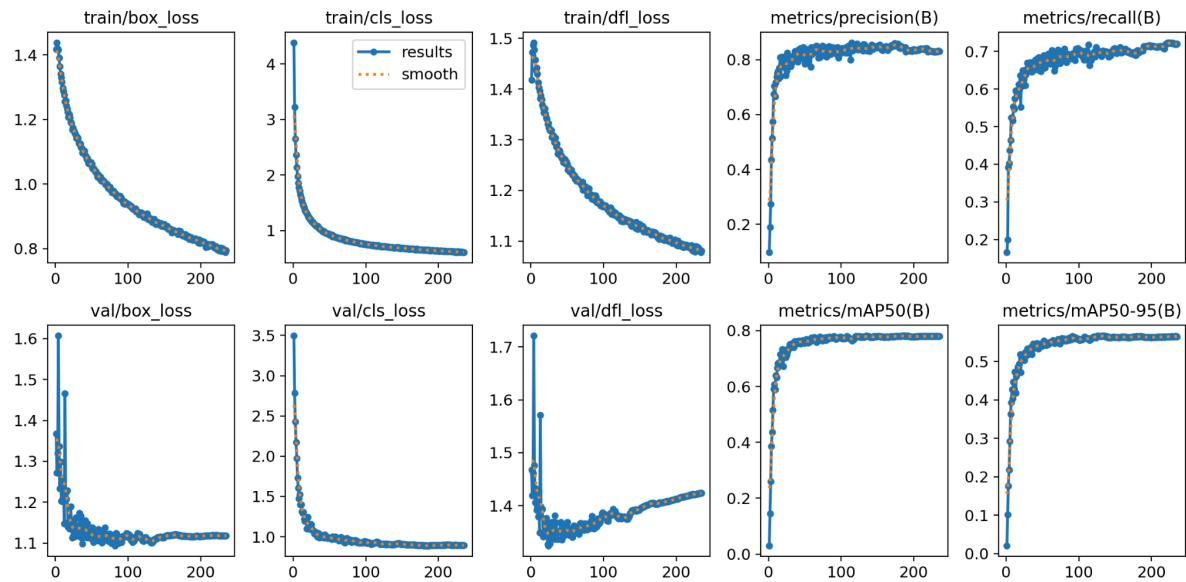
Muestra el recall del modelo en diferentes umbrales de decisión. Una curva que se mantiene alta (cerca de 1) es deseable.





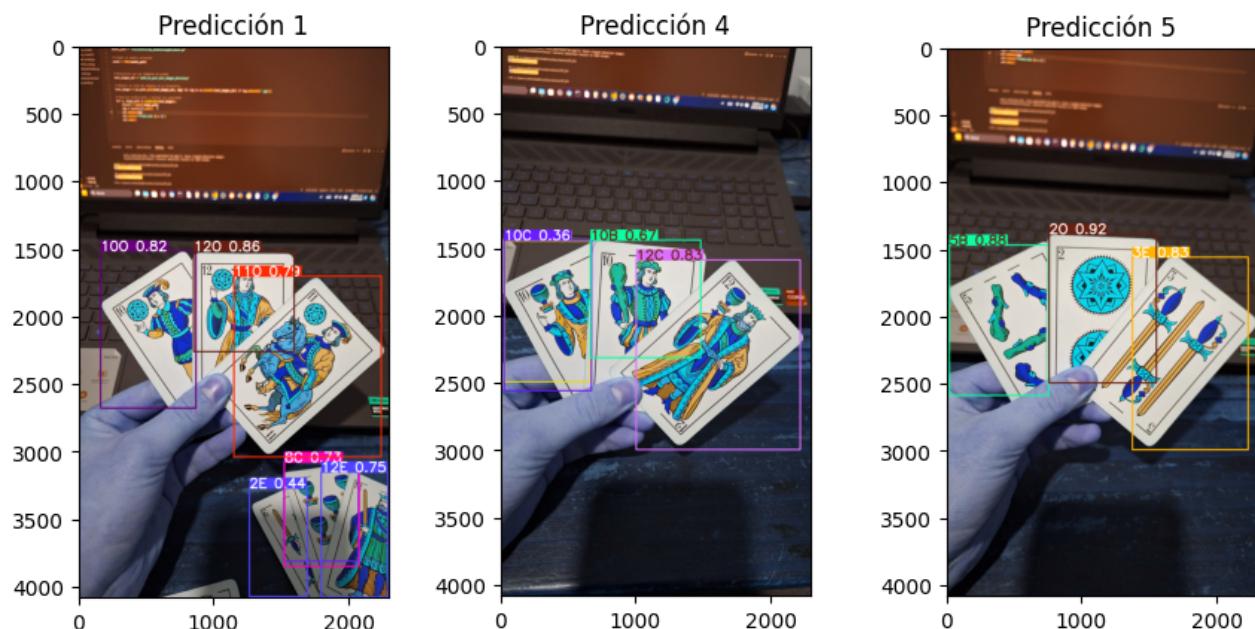
Luego de analizar las gráficas con las curvas de las métricas, se concluyó que el modelo funciona de forma correcta y tiene robustez. Las curvas de F1, precisión (P), precisión-recall (PR) y recall (R) muestran un rendimiento adecuado del modelo en diferentes umbrales de decisión.

### Gráfica de resultados



Estos gráficos nos muestran el desempeño del modelo durante el proceso de entrenamiento. Al analizarlos, podemos observar cómo las métricas de rendimiento del modelo, como la precisión, el recall y el mAP, evolucionan inicialmente con una mejora constante en estas métricas, sin embargo, a medida que el entrenamiento progresó, llegamos a un punto en el que las métricas comienzan a estabilizarse y no mejoran significativamente. Este estancamiento sugiere que el modelo ha alcanzado un nivel de optimización en el que los beneficios adicionales de continuar el entrenamiento pueden ser casi nulos.

### Prueba de test del modelo



Después de realizar una prueba con imágenes que no se encontraban en el dataset de entrenamiento, se ha observado que el modelo mantiene un rendimiento muy bueno en términos de inferencia. Las predicciones realizadas en estas imágenes nuevas demuestran que el modelo tiene una capacidad sólida para generalizar datos no vistos. Aunque, también se identifican ciertos errores en la clasificación, lo que indica áreas potenciales para mejoras. A pesar de estos errores, el modelo sigue ofreciendo resultados efectivos en general.

## Conclusiones generales

El análisis de las métricas y curvas de rendimiento del modelo YOLOv8 revela que el modelo ha alcanzado un desempeño robusto y confiable. Las métricas muestran mejoras significativas a lo largo del proceso de entrenamiento.

Además, las curvas de F1, precisión (P), precisión-recall (PR) y recall (R) indican que el modelo mantiene un equilibrio adecuado entre precisión y recall, lo que es crucial para asegurar un bajo número de falsos positivos y falsos negativos.

Podría ser posible mejorar el modelo si se entrenara con un dataset más grande, ya que un mayor volumen de datos generalmente contribuye a una mejor generalización y precisión. Sin embargo, es importante considerar que el costo computacional de entrenar con un dataset de mayor tamaño es considerablemente elevado.

Este incremento en los recursos necesarios incluye más tiempo de entrenamiento, mayor consumo de memoria y potencia de procesamiento. A pesar de estas limitaciones, el rendimiento actual del modelo para la tarea específica de detección de cartas es bastante satisfactorio.

El modelo ha demostrado una robustez y precisión que cumplen con los requisitos de la aplicación, lo que sugiere que, aunque hay margen para mejorar, el equilibrio entre costo y beneficio ya es bastante favorable. Por lo tanto, en el contexto actual, el modelo proporciona una solución eficaz sin necesidad de una inversión adicional significativa en recursos computacionales.