UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Trabajo Práctico FINAL

IA 52 – PROCESAMIENTO DE IMÁGENES Y VISIÓN POR COMPUTADORA

INFORME

Modelo de detección de cartas españolas
Puntos del Envido

Carrera: Tecnicatura Universitaria en Inteligencia Artificial

Alumno: *Miguel Mussi*

Docentes: *J. P. Manson – A. Geary – A. Leon Cavallo – L. Brugé*

Ciclo: *2024*

TABLA DE CONTENIDOS

Sección A. Presentación ***3***

A.1. Introducción **3**

A.2. Pautas generales del TP **3**

Sección B. Ejercicio 1. Creación de un dataset colaborativo ***4***

B.1. Enunciado **4**

B.2. Resolución **4**

Dataset colaborativo 4

Dataset modificado 4

Sección C. Ejercicio 2. Dataset Personal ***7***

C.1. Enunciado **7**

C.2. Resolución **7**

Validación de anotaciones 7

Validación de formato Yolo 8

Partición en Train – Val – Test 9

Archivo YAML 9

Aumentación 9

Transformaciones 10

Balance de clases 12

Visualización en Fifty-One 14

Aumentación x2 14

Procesamiento en Photoshop 17

Sección D. Ejercicio 3. Entrenamiento de modelos ***18***

D.1. Consigna **18**

D.2. Resolución **18**

Kaggle 18

Yolo 19

Hiperparámetros 19

Métricas de los modelos 20

D.3. Modelo nano (yolov8n) **21**

Parámetros 21

Métricas 21

Gráficos 21

Inferencia en Test 22

Conclusiones 23

D.4. Modelo medium (yolov8m) **24**

Parámetros 24

Métricas 24

Gráficos 24

Inferencia en Test 25

Conclusiones 26

D.5. Modelo large (yolov8l) **26**

Parámetros 26

Métricas 26
Gráficos 27
Inferencia en Test 27
Conclusiones 28
D.6. Resumen comparativo 29
Métricas y tiempos de ejecución 29
Gráficos 29
Conclusiones finales 30
Sección E. Ejercicio 4. Evaluación 31
E.1. Consigna 31
E.2. Aclaraciones adicionales 31
E.3. Resolución 31
Optimización de la inferencia 31
Inferencia 32
Guardado de imágenes 32
Escritura del archivo json 32
Sección F. Plantillas para presentar el trabajo 34
F.1. Consigna 34

Sección A. Presentación

A.1. 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 envido 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.

A.2. 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.

Sección B. Ejercicio 1. Creación de un dataset colaborativo

B.1. Enunciado

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.

B.2. Resolución

Dataset colaborativo

En primera instancia tenemos la creación de un dataset colaborativo entre todos los alumnos que se encuentran cursando la materia, en el que cada uno debe realizar su aporte según indican las consignas.

Comienzo por mencionar que mis imágenes correspondientes para este dataset colaborativo se encuentran en el siguiente link hacia una carpeta en Google Drive: [Imágenes]

Se aclara que el dataset que se va a utilizar en los siguientes ejercicios es un dataset modificado que parte de este dataset colaborativo. El motivo de esta modificación tiene que ver con irregularidades detectadas y falta de criterio unificado en cuanto a la técnica utilizada para los bounding boxes.

El proceso que llevamos a cabo en un grupo conformado por tres personas, consiste en tomar todas las imágenes originales que se compartieron para la creación del dataset y asegurarnos de utilizar la misma técnica para los bounding boxes en todas las imágenes y además asegurar que todos los bounding boxes estén en el mismo formato en su respectivos txt.

Dataset modificado

Para la creación de este nuevo dataset modificado utilizamos la herramienta de anotación de imágenes llamada **Labelimg**. Esta herramienta se puede descargar y utilizar siguiendo las instrucciones dadas en su repositorio de GitHub: [Repositorio-labelimg]. Este repositorio indica como instalarlo en los distintos sistemas operativos o entornos, en mi caso, utilice la opción indicada para Windows + Anaconda.

Los tres usamos la misma herramienta ya que esta, al guardar los labels en formato txt, los guarda con un formato estándar de YOLO con 5 valores para cada bounding boxes.

A continuación, una imagen de como se ve en el txt:

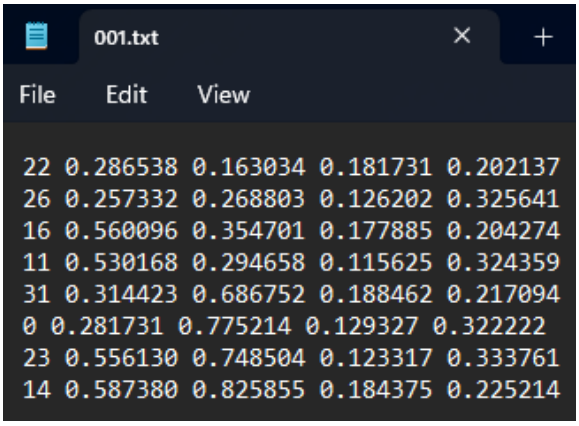


Imagen 1. Label de ejemplo en formato yolo escrito en un .txt

Como podemos ver en este txt, correspondiente a la primera imagen, tiene 8 filas donde cada fila representa un bounding box diferentes y, por lo tanto, una carta diferente. El primer valor (22, 26, 16, …) representa la clase a la que pertenece dicho bounding box, mientras que los cuatros valores consecuentes representan:

- Centro con respecto a x
- Centro con respecto a y
- Ancho
- Largo

Las clases están mapeadas de forma numérica pero cada una de ellas representa una carta diferente dentro de la baraja española. A continuación, se muestra que número representa cada carta:

• 0 – 1 Oro • 1 – 1 Copa • 2 – 1 Espada • 3 – 1 Basto • 4 – 2 Oro • 5 – 2 Copa • 6 – 2 Espada • 7 – 2 Basto • 8 – 3 Oro • 9 – 3 Copa • 10 – 3 Espada • 11 – 3 Basto • 12 – 4 Oro	• 13 – 4 Copa • 14 – 4 Espada • 15 – 4 Basto • 16 – 5 Oro • 17 – 5 Copa • 18 – 5 Espada • 19 – 5 Basto • 20 – 6 Oro • 21 – 6 Copa • 22 – 6 Espada • 23 – 6 Basto • 24 – 7 Oro • 25 – 7 Copa	• 26 – 7 Espada • 27 – 7 Basto • 28 – 8 Oro • 29 – 8 Copa • 30 – 8 Espada • 31 – 8 Basto • 32 – 9 Oro • 33 – 9 Copa • 34 – 9 Espada • 35 – 9 Basto • 36 – 10 Oro • 37 – 10 Copa • 38 – 10 Espada	• 39 – 10 Basto • 40 – 11 Oro • 41 – 11 Copa • 42 – 11 Espada • 43 – 11 Basto • 44 – 12 Oro • 45 – 12 Copa • 46 – 12 Espada • 47 – 12 Basto • 48 – Comodín

Como podemos ver en la imagen 1 anterior, en las 8 filas tenemos la clase 22, 26, 16, 11, 31, 0, 23 y 14. Si mapeamos cada clase con su respectiva carta nos queda que la imagen tiene las siguientes cartas:

• 2 = 6 de Espada • 26 = 7 de Espada	• 16 = 5 de Oro • 11 = 3 de Basto	• 31 = 8 de Basto • 0 = 1 de Oro	• 23 = 6 de Basto • 14 = 4 de Espada

Vamos a ver la imagen correspondiente a ver si, efectivamente, posee estas 8 cartas:

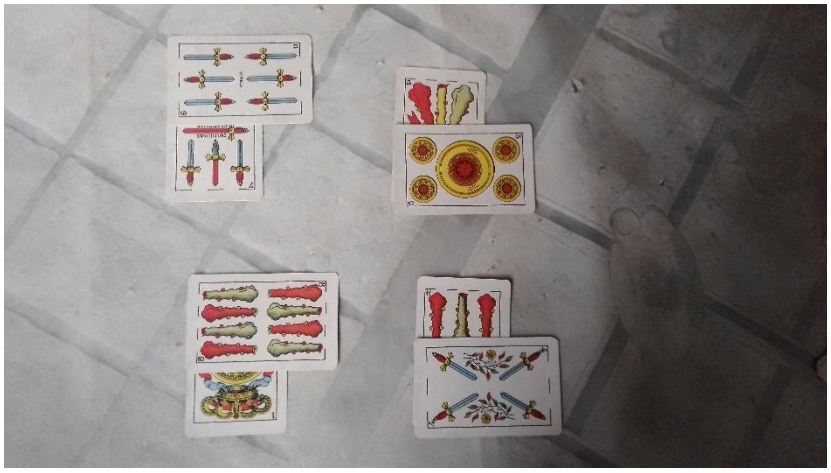
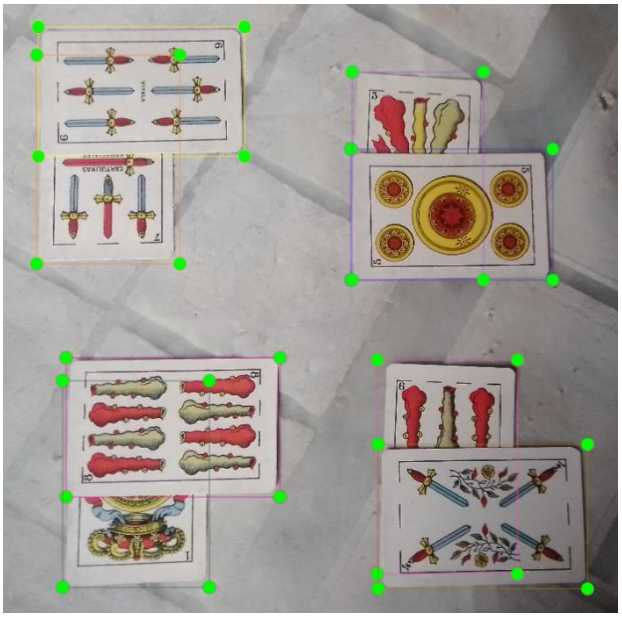


Imagen 2. Imagen de ejemplo de 8 cartas de la baraja española

Como podemos ver, las cartas que se encuentran en las imágenes corresponden con los respectivos labels en su txt correspondiente. Por último, para cerrar el tema de la creación del dataset colaborativo modificado, la técnica utilizada para la creación de los bounding boxes fue la de Carta Completa, es decir, se hizo un bounding box teniendo en cuenta el posible tamaño de la carta. ¿Por qué posible? Porque en ciertas imágenes, como la que se utilizó de ejemplo (Ver imagen 2) ciertas cartas tienen partes de las mismas tapadas, por ejemplo, el 1 de Oro de la imagen tiene su mitad tapada por otra carta, en este caso, el 8 de Basto.

Mostramos una imagen para ver cómo se ven los bounding boxes utilizando esta técnica sobre la misma imagen de ejemplo:



***I**magen 3. Imagen de ejemplo de la técnica de carta completa*

Por último, se menciona que el dataset modificado consta de unas ***1426 imágenes*** se encuentra en el siguiente link para su descarga: [dataset-colaborativo-modificado]

Sección C. Ejercicio 2. Dataset Personal

C.1. Enunciado

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.

C.2. Resolución

En este ejercicio ya tenemos una parte hecha, la "Unificación de las anotaciones en un único formato YOLO". Esto se hizo junto en la modificación del dataset comunitario para tener todas las anotaciones en un formato de YOLO con 5 valores. Para ver el formato más específicamente, repasar el "*Ejercicio 1. Creación de un dataset colaborativo"*. Por lo que vamos a centrarnos en las demás tareas.

Validación de anotaciones

Como se mencionó antes, el dataset propio importado contiene un total de 1426 imágenes. Como primera etapa del análisis se verificará:

- Que ningún punto de los bbox quede fuera de los límites de la imagen.
- Que ningún archivo .txt contenga una clase que no corresponda a una carta.

En caso de ocurrencia de alguna de estas situaciones, se moverá la imagen y el archivo txt a una carpeta llamada "label_errors".

Luego se imprimen los datos de las imágenes movidas de carpeta para analizar el motivo y se registra la cantidad descartada para evaluar su impacto porcentual en el dataset.

En general, el motivo por el que fueron movidas es por un pequeño valor que figura fuera de los límites de la imagen original.

```
------------------------
Imagen: '178.jpg' movida
Primer etiqueta en conflicto:
x1=0.7233335
x2=1.0000005
y1=0.661
y2=0.94675
clase=39(10B)
------------------------
------------------------
Imagen: '599.jpeg' movida
Primer etiqueta en conflicto:
x1=0.55
x2=0.70625
y1=0.7511115
y2=1.0000005
clase=25(7C)
------------------------
------------------------
Imagen: '906.png' movida
Primer etiqueta en conflicto:
x1=0.45208349999999997
x2=1.0000005
y1=0.35546849999999997
y2=0.7562495
clase=23(6B)
------------------------
------------------------
Imagen: '413.jpg' movida
Primer etiqueta en conflicto:
x1=0.6038415
x2=1.0000005
y1=0.4897055
y2=0.7193625
clase=42(11E)
------------------------
------------------------
Imagen: '450.jpg' movida
Primer etiqueta en conflicto:
x1=0.37499950000000004
x2=0.6616665
y1=0.6333335
y2=1.0000005
clase=43(11B)
------------------------
```

```
------------------------
Imagen: '250.jpg' movida
Primer etiqueta en conflicto:
x1=0.14700000000000002
x2=0.67125
y1=0.5403335
y2=1.0000005
clase=19(5B)
------------------------
------------------------
Imagen: '1190.png' movida
Primer etiqueta en conflicto:
x1=0.7939455
x2=1.0000005
y1=0.3872545
y2=0.5610295000000001
clase=28(8O)
------------------------
------------------------
Imagen: '170.jpg' movida
Primer etiqueta en conflicto:
x1=0.5983335
x2=1.0000005
y1=0.20275
y2=0.44425000000000003
clase=14(4E)
------------------------
------------------------
Imagen: '393.jpg' movida
Primer etiqueta en conflicto:
x1=0.1745095
x2=0.47965650000000004
y1=0.5846355
y2=1.0000005
clase=24(7O)
------------------------
------------------------
Imagen: '1062.png' movida
Primer etiqueta en conflicto:
x1=0.322
x2=0.61025
y1=0.7613335
y2=1.0000005
clase=17(5C)
------------------------
```

```
[9]  1 print(f'Cantidad de imagenes movidas: {int(len(os.listdir(ruta_label_errors))/2)}')
0s
Cantidad de imagenes movidas: 60
```

La cantidad de imágenes filtradas por este proceso de validación es de 60 sobre un total de 1426, lo que constituye un 4.20% que consideramos de bajo impacto en el dataset como para ser reprocesado y recuperado.

Validación de formato Yolo

En cuanto a esta parte de la consigna no es necesario hacerla ya que las anotaciones de todas las imágenes se volvieron a realizar bajo ese mismo criterio, por lo tanto, todas ellas contienen su respectiva anotación en versión de 5 valores por línea, como se mencionó en un apartado anterior.

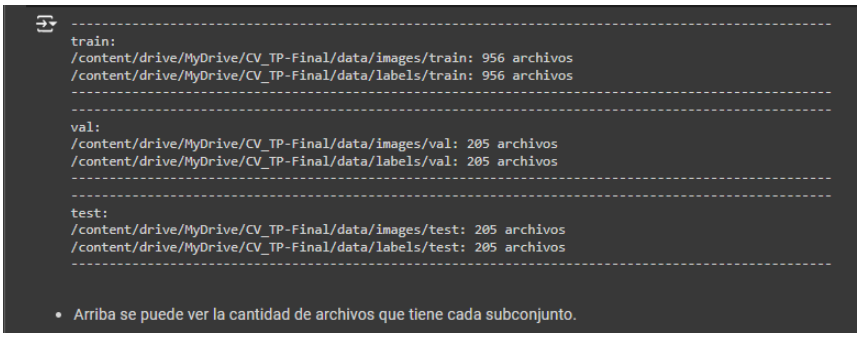
Partición en Train – Val – Test

En esta sección se realiza la correspondiente división del dataset personal en tres conjuntos: entrenamiento (train), validación (val) y prueba (test). En cada uno de ellos se reasignan de manera aleatoria las imágenes y sus labels correspondientes en una proporción 70% - 15% - 15%, respectivamente.

El split "***train***" se va a utilizar para entrenar el modelo y va a ser el conjunto donde se va a aplicar la aumentación de imágenes más adelante. El split "***val***" lo va a utilizar YOLO para validar las métricas después de cada época de entrenamiento. Y el split ***"test"*** se va a utilizar en la última instancia para realizar inferencia sobre esas imágenes.

Recordemos que el dataset tenía 1426 imágenes a las que se le deben descontar 60 que fueron descartadas por el proceso de validación de sus anotaciones. Por lo que el conjunto tiene, ahora, 1366 imágenes en total.

Se imprime una comprobación de las cantidades de archivos en cada carpeta.

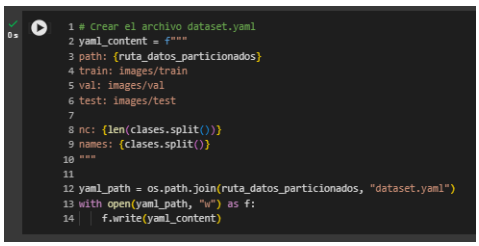


Resumen del Split (Total 1366 imágenes):

- Train: 0.70 o 70% 956 imágenes
- Val: 0.15 o 15% 205 imágenes
- Test: 0.15 o 15% 205 imágenes

Archivo YAML

Con la división de los conjuntos ya realizada, se procede a generar el archivo "dataset.yaml" correspondiente.



Aumentación

Este proceso de aumentación de datos o "data augmentation" es una técnica que se realiza en el conjunto de entrenamiento y consiste en aumentar la cantidad de imágenes por algún método sintético para mejorar el desempeño de los modelos durante su entrenamiento. Nos referimos como "métodos sintéticos" de generación de imágenes al proceso de generar copias de las mismas con alguna transformación aplicada sobre ellas. Recordemos que para la misma finalidad también podrían capturarse más imágenes.

Esta técnica no sólo consiste en generar copias de las imágenes una determinada cantidad de veces (no funcionaría dado que le estaríamos dando al modelo información que ya vio, y el modelo aprende a partir de información nueva), data aumentation consiste en aplicar distintas técnicas que pueden meter ruido gaussiano, ajustar el brillo o el contraste, aumentar el desenfoque de movimiento, convertir a escala de grises, girar o voltear la imagen, etc.

Antes de aplicar las transformaciones se realiza un cálculo de las áreas de los bboxes del conjunto de train, de modo de el mínimo valor de área encontrado será el definido como *"min_area"* para definir el pipeline de las transformaciones.

```
# Vemos algunas medidas de resumen
tabla_areas.describe()
```

	area_bb
count	5781.000000
mean	0.077948
std	0.048267
min	0.008092
25%	0.045542
50%	0.067598
75%	0.097179
max	0.463541

Transformaciones

Para seleccionar las transformaciones a aplicar y los parámetros de cada una de ellas se requirió de la ayuda de la documentación oficial de Albumentations y se experimentó cada una de ellas en el [sitio oficial].

```
# Creamos el pipeline de transformaciones
transform = A.Compose([
  # Transormaciones a nivel pixel
A.OneOf([ # Cuando usamos OneOf ponemos una p de OneOf de que se entre en este bloque y las p de cada uno de los elementos del interior en 1 asi el que se elige se ejecuta si o si
    A.CLAHE(always_apply=False, p=1.0, clip_limit=(1, 45), tile_grid_size=(47, 52)), # Aplica ecualizacion de histograma adaptativo limitado a la imagen de entrada
    A.RandomBrightnessContrast(always_apply=False, p=1.0, brightness_limit=(-0.2, 0.48), contrast_limit=(-0.08, 0.51), brightness_by_max=True) # Ajusta aleatoriamente el brillo y el contraste
], p=0.5),
A.OneOf([
    A.Blur(always_apply=False, p=1.0, blur_limit=(3, 75)), # Desenfoca imagen de entrada utilizando kernel de tamaño aleatorio
    A.Defocus(always_apply=False, p=1.0, radius=(3, 3), alias_blur=(0.0, 0.0)), # Aplica desenfoque
    A.GaussNoise(always_apply=False, p=1.0, var_limit=(271.28, 480.5), per_channel=True, mean=45.39), # Aplica ruido Gaussiano a la imagen de entrada
    A.ISONoise(always_apply=False, p=1.0, intensity=(0.6, 1.06), color_shift=(0.21, 0.33)), # Aplica ruido de sensor de camara
    A.MotionBlur(always_apply=False, p=1.0, blur_limit=(9, 19), allow_shifted=True), # Aplica desenfoque de movimiento
    A.Sharpen(always_apply=False, p=1.0, alpha=(0.55, 0.86), lightness=(0.32, 2.13)) # Enfoca la imagen
], p=0.4),
A.OneOf([
    A.ChannelDropout(always_apply=False, p=1.0, channel_drop_range=(1, 1), fill_value=117), # Elimina canales aleatoriamente en la imagen de entrada
    A.ChannelShuffle(always_apply=False, p=1.0), # Mezcla aleatoriamente los canales de la imagen de entrada
    A.Downscale(always_apply=False, p=1.0, scale_min=0.31, scale_max=0.39, interpolation=cv2.INTER_LINEAR), # Disminuye la calidad de la imagen
    A.Emboss(always_apply=False, p=1.0, alpha=(0.27, 0.46), strength=(2.55, 3.97)), # Realza la imagen de entrada y superpone el resultado con la imagen original
    A.MultiplicativeNoise(always_apply=False, p=1.0, multiplier=(1.66, 2.39), per_channel=True, elementwise=True), # Multiplica los pixeles por un numero aleatorio
    A.ToGray(always_apply=False, p=1.0) # Pasa la imagen a blanco y negro
], p=0.3),
A.OneOf([
    A.PixelDropout(always_apply=False, p=1.0, dropout_prob=0.12, per_channel=0, drop_value=0), # Establece algunos pixeles a 0
    A.Spatter(always_apply=False, p=1.0, mean=(0.2, 0.5), std=(0.1, 0.2), gauss_sigma=(0.1, 0.3), intensity=(0.1, 0.2), cutout_threshold=(0.1, 0.2), mode=['rain'], color={'rain': [238, 238, 175], 'mud': [20, 42, 63]}) # Aplica transformación de salpicaduras
], p=0.4),
  # Transformaciones a nivel espacial
A.OneOf([
    A.Flip(always_apply=False, p=0.1), # Da vuelta la imagen vertical u horizontalmente u ambas
    A.Rotate(always_apply=False, p=1.0, limit=(0, 360), interpolation=3, border_mode=0, value=(0, 0, 0), mask_value=NONE, rotate_method='largest_box', crop_border=False) # Rota la imagen de entrada
], p=0.7)
], bbox_params=A.BboxParams(format='yolo', min_area=area_minima.to_numpy()[0]))
```

Ejemplos de las transformaciones aplicadas aleatoriamente:

- **CLAHE**. Aplica ecualización de histograma adaptativo limitado a la imagen de entrada.
- **RandomBrightnessContrast**. Ajusta aleatoriamente el brillo y el contraste.
- **Blur**. Desenfoca imagen de entrada utilizando kernel de tamaño aleatorio.
- **Defocus**. Aplica desenfoque a la imagen de entrada.
- **GaussNoise**. Aplica ruido Gaussiano a la imagen de entrada.
- **ISONoise**. Aplica ruido de sensor de cámara.
- **MotionBlur**. Aplica desenfoque de movimiento.
- **Sharpen**. Enfoca la imagen.

- **ChannelDropout**. Elimina canales aleatoriamente en la imagen de entrada.
- **ChannelShuffle**. Mezcla aleatoriamente los canales de la imagen de entrada.
- **Downscale**. Disminuye la calidad de la imagen.
- **Emboss**. Realza la imagen de entrada y superpone el resultado con la imagen original.
- **MultiplicativeNoise**. Multiplica los pixeles por un número aleatorio.
- **ToGray**. Convierte a escala de grises.
- **PixelDropout**. Establece los valores de algunos pixeles a 0.
- **Spatter**. Aplica transformación de salpicaduras.
- **Flip**. Voltea la imagen vertical u horizontalmente, u ambas.
- **Rotate**. Aplica una rotación aleatoria a la imagen de entrada

En primera instancia se realizó una aumentación por cada instancia de entrenamiento. Como se mencionará más adelante, con este nuevo conjunto se realizó una etapa de entrenamiento del modelo nano de Yolo. Las métricas obtenidas, en comparación con otras opiniones de compañeros (que experimentaban con mayor cantidad de aumentaciones), hizo que se decidiera por aplicar un nuevo ciclo de aumentaciones más adelante.

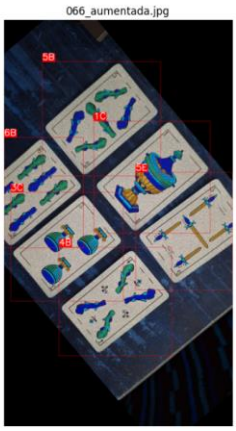
El conjunto de train tenía 956 imágenes antes del data-augmentation, se imprime una comprobación de la cantidad final al terminar:

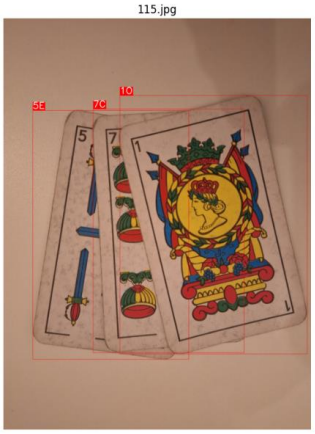
```
[32]  1 # Lista con los nombres de las instancias de entrenamiento
      2 print('-----'*7)
      3 print('Train:')
      4 print('-----'*7)
      5 print(f'Cantidad de imagenes: {len(os.listdir(ruta_img_train))}')
      6 print(f'Cantidad de labels: {len(os.listdir(ruta_labels_train))}')
      7 print('-----'*7)

----------------------------------
Train:
----------------------------------
Cantidad de imagenes: 1912
Cantidad de labels: 1912
----------------------------------
```

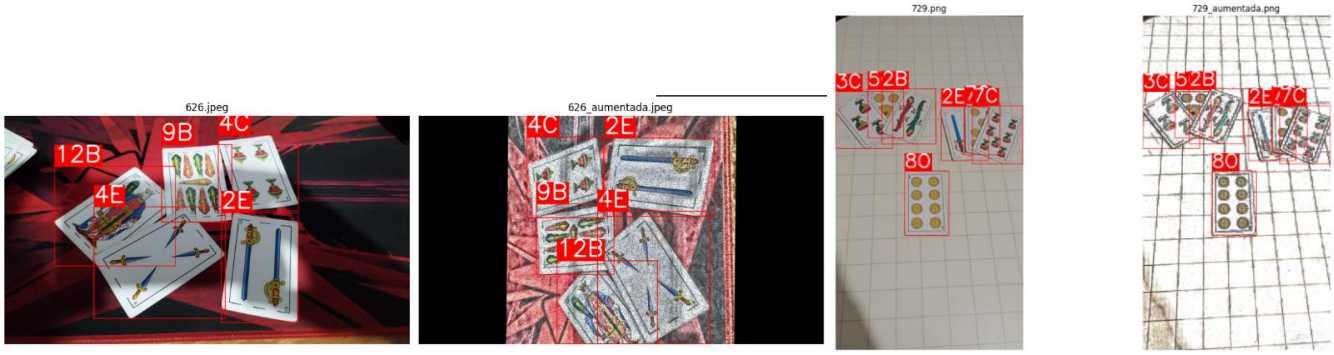
- Antes teniamos 956 imagenes y labels de train, ahora tenemos 1912, es decir, el doble.

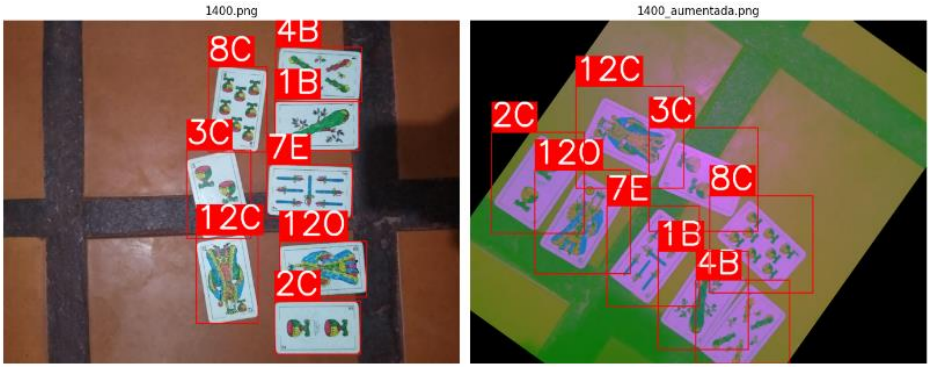
Finalmente, con los bbox de las imágenes originales y las aumentadas, se procede a realizar una cantidad generosa de impresiones de muestra para verificación e inspección visual del proceso.

Ejemplos de aumentación









En líneas generales, la aumentación parece haberse hecho respetando bastante bien los bboxes originales. En algunos casos el bbox quizás no quedo tan ajustado, pero aún así continúa envolviendo correctamente a la carta.

Balance de clases

Para realizar esta tarea es necesario, ahora que tenemos todo el dataset ampliado (original+aumentado) recorrer las anotaciones e ir registrando las apariciones de las clases para poder graficar la composición

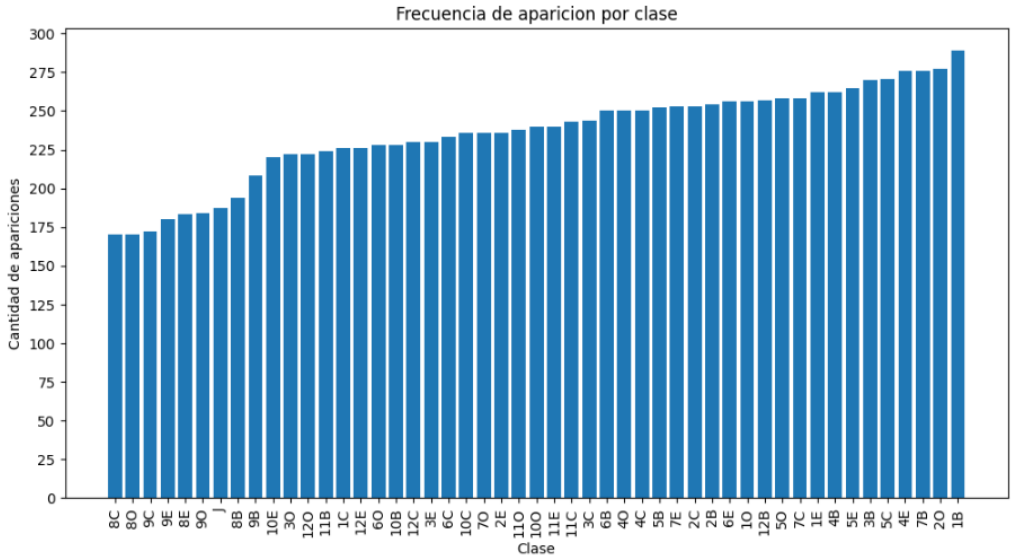
	clase	frecuencia_absoluta	frecuencia_relativa	orden_original
29	8C	170	1.47	29
28	8O	170	1.47	28
33	9C	172	1.49	33
34	9E	180	1.56	34
30	8E	183	1.59	30
32	9O	184	1.59	32
48	J	187	1.62	48
31	8B	194	1.68	31
35	9B	208	1.80	35
38	10E	220	1.91	38
8	3O	222	1.92	8
44	12O	222	1.92	44
43	11B	224	1.94	43
1	1C	226	1.96	1
46	12E	226	1.96	46
20	6O	228	1.97	20

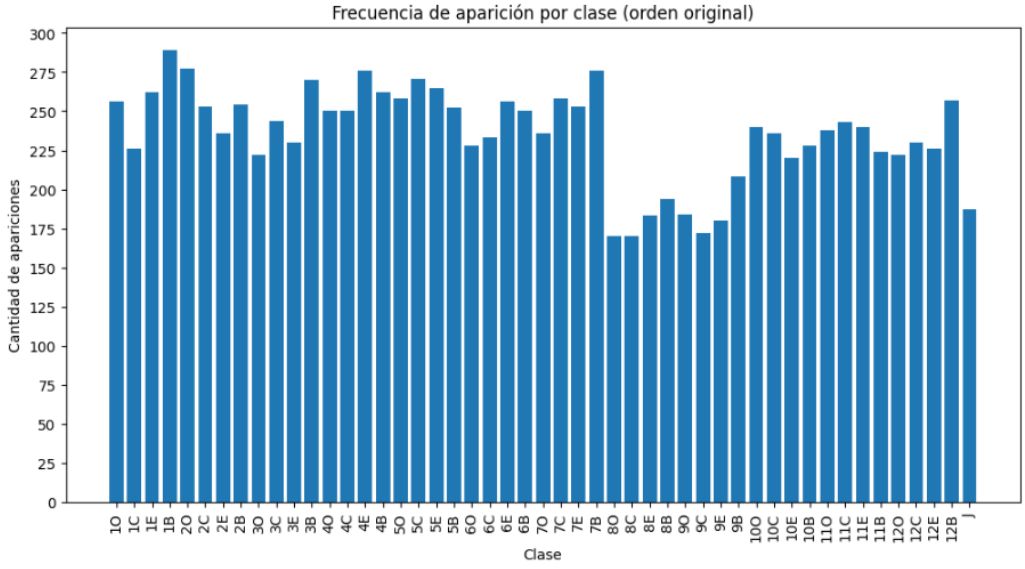
	frecuencia_absoluta	frecuencia_relativa
count	49.000000	49.000000
mean	235.612245	2.040408
std	30.265365	0.262472
min	170.000000	1.470000
25%	224.000000	1.940000
50%	240.000000	2.080000
75%	256.000000	2.220000
max	289.000000	2.500000

```
[39]  1 print(f"Cantidad de cartas en total: {tabla_conteo_clases['frecuencia_absoluta'].sum()}")
0s
Cantidad de cartas en total: 11545
```

Observando las medidas resumen de las frecuencias relativas vemos que todas las clases con tienen en proporción entre el 1.39% y 2.38% del total. En términos absolutos estamos hablando entre 160 y 274 apariciones por clase.

Se realizan gráficos de barras para evaluar la frecuencia de aparición por clase, uno ordenado por frecuencia (de menor a mayor) y el otro ordenado como el mazo original. Se adjuntan los resultados.





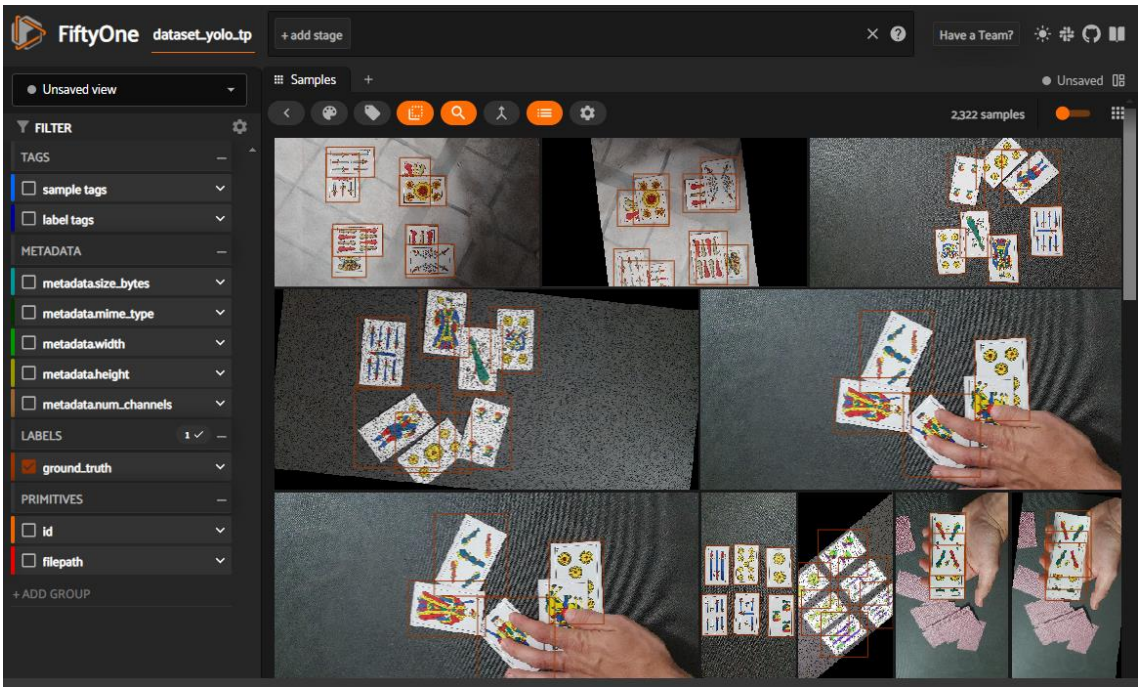
- A simple vista lo anterior no se aprecia un gran desbalance de los datos.
- La carta de mayor aparición es el 1 de basto y la menos frecuente el 8 de copa.

La reducción significativa que se aprecia en algunas cartas corresponde a los cuatro 8, los cuatro 9 y el joker o comodín. Esto tiene sentido, y es lógico que suceda, porque el trabajo apunta a una consigna relacionada al juego del truco, en el que esas nueve cartas no se emplean. Por otro lado, también existen versiones comerciales de mazos de 50 como de 40 cartas. Puede suceder que algunas personas tengan disponibles en sus casas versiones de estos últimos.

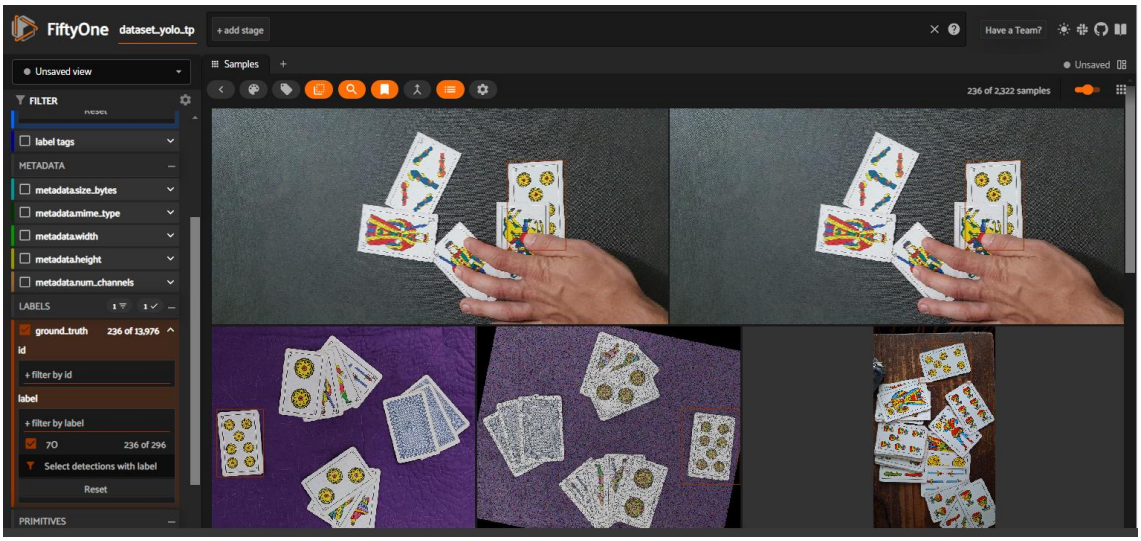
Se decide no tomar ninguna medida respecto al balance del conjunto de train, primero por la dificultad al considerar que cada imagen tiene varias cartas en su interior, distribuídas aleatoriamente y con superposición (lo que dificulta su recorte); y segundo, por lo que se mencionó, el modelo será empleado trabajará sobre los puntos del envido en el juego del truco, en el que mayoritariamente las cartas corresponden al mazo de 40 y sólo excepcionalmente se incluirán algunos 8/9 para poner a prueba el modelo.

Visualización en Fifty-One

En esta sección se hace uso de la librería ***FiftyOne*** para poder visualizar y analizar las imágenes y los bbox de los conjuntos del dataset.



Ejemplo de filtrado del 7 de oros



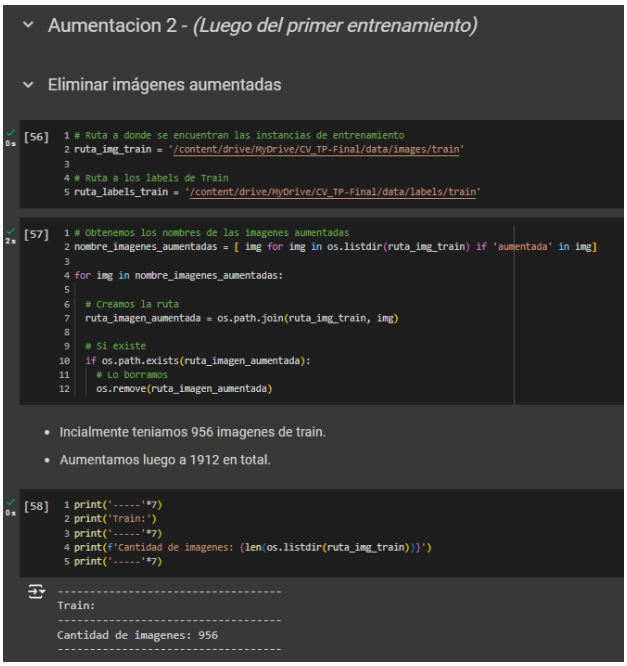
Aumentación x2

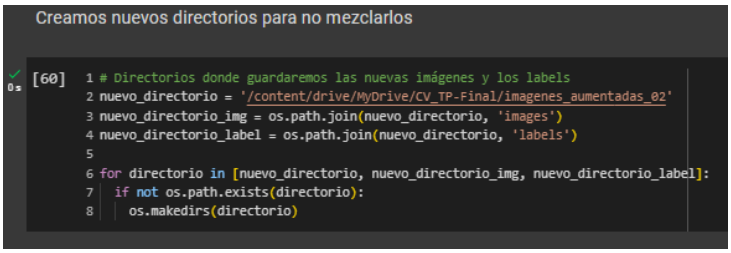
A esta altura del trabajo ya se intentó entrenar el primer modelo (Yolov8n) en una notebook de Kaggle haciendo una aumentación de 1 imagen por cada imagen del dataset de train. Se obtienen métricas decentes, pero se intentará mejorarlas. Lo que se intentará para mejorar las métricas es realizar una nueva aumentación por cada imagen (es decir por cada imagen original, nos quedarán 2 imágenes aumentadas).

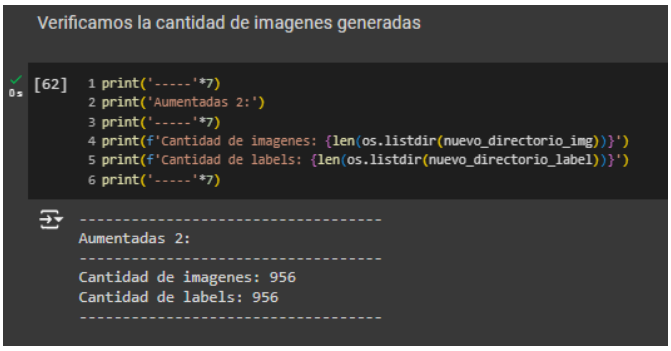
Debido a que trabajamos con Drive y Colab para esta etapa y Drive nos proporciona un almacenamiento limitado, vamos a tener que eliminar las primeras aumentaciones para generar espacio, generar las segundas y juntar todo en nuestro pc local para subirlo luego a un dataset Kaggle y entrenar desde una notebook Kaggle.

En principio sólo se eliminarán las imágenes aumentadas de la primera aumentación y los txt con los bb se dejarán con el objetivo de luego sumar los nuevos y poder hacer un nuevo análisis del balance de clases (ya que sólo necesitamos los txt).

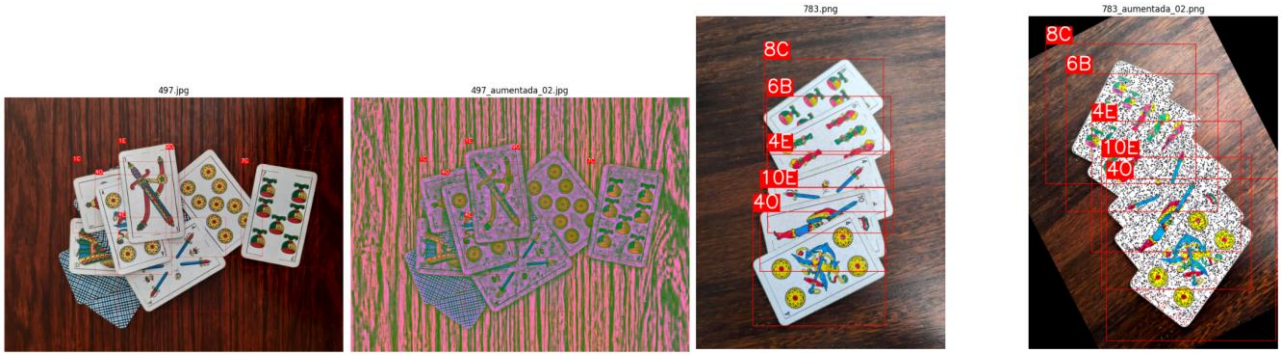
Las nuevas imágenes se descargarán manualmente y se juntara todo en las carpetas correspondientes para subir a Kaggle.

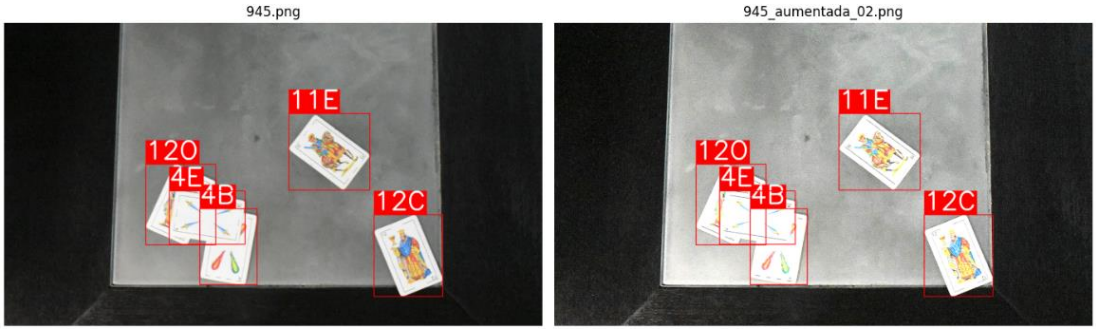






Ejemplos de segunda aumentación





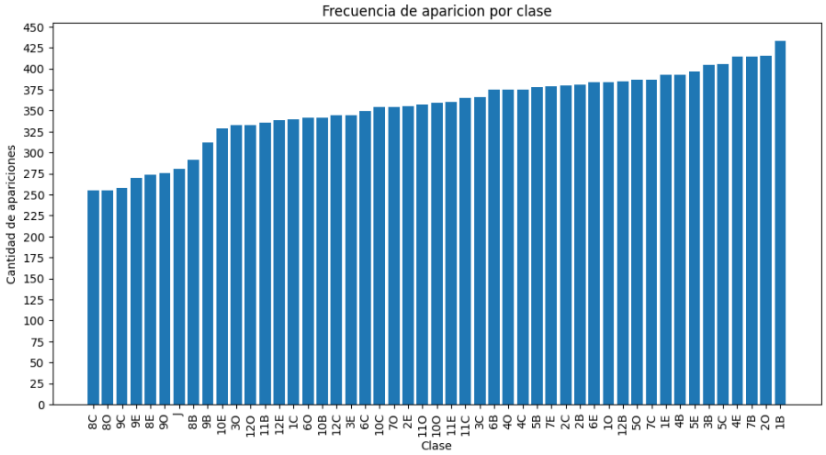
Verificación del balance de las clases

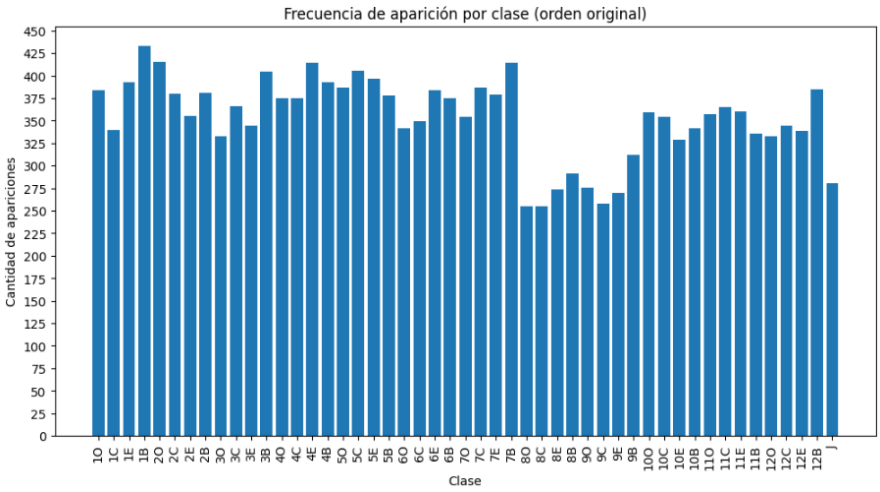
	clase	frecuencia_absoluta	frecuencia_relativa	orden_original
29	8C	255	1.47	29
28	8O	255	1.47	28
33	9C	258	1.49	33
34	9E	270	1.56	34
30	8E	274	1.58	30
32	9O	276	1.59	32
48	J	281	1.62	48
31	8B	291	1.68	31
35	9B	312	1.80	35
38	10E	329	1.90	38
8	3O	333	1.92	8
44	12O	333	1.92	44

	frecuencia_absoluta	frecuencia_relativa	orden_original
count	49.000000	49.000000	49.00000
mean	353.367347	2.040408	24.00000
std	45.343455	0.262583	14.28869
min	255.000000	1.470000	0.00000
25%	336.000000	1.940000	12.00000
50%	359.000000	2.070000	24.00000
75%	384.000000	2.220000	36.00000
max	433.000000	2.500000	48.00000

```
1 print(f"Cantidad de cartas en total: {tabla_conteo_clases['frecuencia_absoluta'].sum()}")
```

```
Cantidad de cartas en total: 17315
```



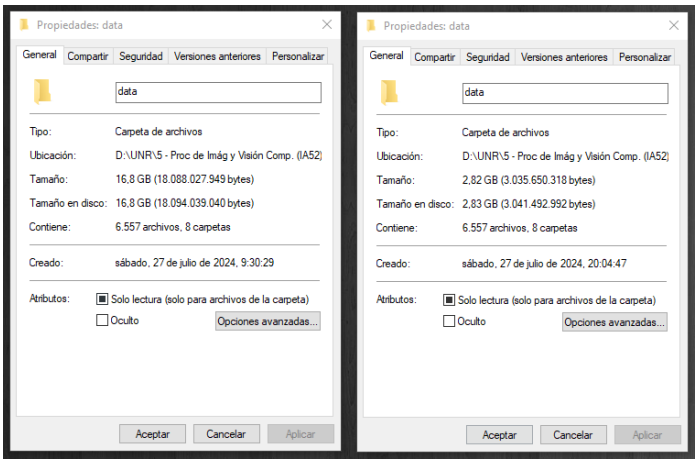


Se mantiene la relación anterior en términos relativos por lo que el dataset continúa bastante balanceado. Con este nuevo dataset vamos a probar entrenar un modelo pre-entrenado con más parámetros y evaluar si mejora su performance.

Procesamiento en Photoshop

Como última etapa del procesamiento del dataset se observa que el dataset aumentado tiene más de los 15 Gb, lo que excede el almacenamiento que ofrece Google Drive para sus cuentas gratuitas. Se menciona que algunas de las imágenes del dataset superan los 10Mb, lo que puede ser optimizado significativamente. Por ese motivo, se decidió procesar todo el conjunto de imágenes en un lote y aplicarle una tarea automatizada en Adobe Photoshop que consiste en comprimirlas y guardar como jpg (algunas ya estaban en este formato y otras en png). No se modificó tamaño ni relación de aspecto.

De esta manera, el dataset personal aumentado pasó de casi 17 Gb a 2.8 Gb lo que no sólo mejorará los tiempos de carga y descarga de carpetas compartidas, sino también redundará en una mejora en los tiempos de entrenamiento e inferencia de los modelos.



Sección D. Ejercicio 3. Entrenamiento de modelos

D.1. Consigna

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.

D.2. Resolución

En este ejercicio necesitamos entrenar el modelo o varios para conseguir las mejores métricas para luego concretar la inferencia con imágenes extras. Lo primero que se debe recordar es que todos los entrenamientos fueron llevados a cabo en la plataforma Kaggle con GPU y se probaron un total de 3 modelos distintos de YOLO, cada uno sobre la base de un modelo pre-entrenado distinto, donde la diferencia de tamaño entre ellos radica en la cantidad de parámetros.

Kaggle

Para el entrenamiento de los modelos con el dataset personal de imágenes de cartas españolas, y ante la falta de una GPU correctamente configurada para trabajar de manera local, se optó por realizarlos en la plataforma Kaggle (kaggle.com) en lugar de Google Colab (que se usó para el procesamiento de las imágenes y creación del dataset). La decisión se basa en la disponibilidad que ofrece esta plataforma de:

- Dos Nvidia Tesla T4 GPUs para entrenar en paralelo (4 CPU cores - 29Gb de RAM).
- Almacenamiento privado de 20 Gb para notebooks
- 30 horas semanales de GPU para administrar como se quiera
- Trabajo ininterrumpido y sin desconexiones (hasta 12 hs con CPU/GPU).

Algunos detalles a considerar. En primer lugar, en esta plataforma se debe importar el dataset comprimido como un archivo zip y luego Kaggle lo despliega para usar durante los entrenamientos. Otro aspecto esta relacionado al archivo yaml correspondiente al dataset, hay que redefinir la variable path debido a que en el archivo original estaban relacionada a la ubicación en Drive.

Yolo

YOLO (You Only Look Once), es un popular modelo de detección de objetos y segmentación de imágenes, fue desarrollado por Joseph Redmon y Ali Farhadi en la Universidad de Washington. Lanzado en 2015, YOLO ganó popularidad rápidamente por su gran velocidad y precisión. En esta oportunidad, se trabajó con modelos de la versión 8:

Model	Name	Size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	Nano	640	37.3	80.4	0.99	**3.2**	8.7
YOLOv8s	Small	640	44.9	128.4	1.20	**11.2**	28.6
YOLOv8m	Medium	640	50.2	234.7	1.83	**25.9**	78.9
YOLOv8l	Large	640	52.9	375.2	2.39	**43.7**	165.2
YOLOv8x	Extra	640	53.9	479.1	3.53	**68.2**	257.8

Tabla comparativa de los modelos aplicados a la tarea de DETECCIÓN sobre el dataset COCO

A modelos más pesados, más tiempo de entrenamiento por época y se espera que las métricas sean mejores.

Para este análisis se comenzó trabajando de “menos a más”, entrenando en primera instancia al modelo **nano** y con el dataset que tenía sólo un ciclo de aumentación. Analizando las métricas resultantes es que se decidió realizar una nueva aumentación para definir al que sería el dataset definitivo. Por esta razón es que no se registraron las métricas de ese primer entrenamiento, dado que se lo consideró una prueba en borrador. Una vez entrenado el modelo nano, se probó con el **médium** y luego con el **large**. Las métricas resultantes de este último, en comparación con las métricas obtenidas por compañeros que probaron el modelo extra (las nuestras con el modelo large resultaban superiores), hacen que se considere suficiente el entrenamiento, no probar con otros modelos y limitamos el análisis a estos tres.

Hiperparámetros

Se adjunta un detalle de los hiperparámetros analizados en los entrenamientos.

- **epochs**: la cantidad de épocas de entrenamiento. Se probó con 100 y resultó suficiente, siempre se interrumpió por earlystopping unas pocas épocas antes.
- **Imgsz**: es el tamaño a la que van a redimensionar las imágenes para que tengan todas el mismo tamaño y mismo nivel de detalle, mientras más alto sea el número mayor calidad y también mayor cantidad de memoria necesaria y, por lo tanto, más tiempo de entrenamiento. Su valor para todos los modelos es: 640.
- **batch**: Tamaño del batch, como tal este término hace referencia el tamaño del conjunto de entrenamiento que se va a utilizar en simultaneo durante cada iteración del entrenamiento. Su valor para todos los modelos es: 16.
- **device**: este hiperparámetro sirve para especificar si se va a utilizar la CPU o la/las GPU (en caso de tener varias). En este caso, el valor 0 representa que se va a utilizar la GPU en la posición 0. Por un descuido, los dos primeros modelos se entrenaron con una sola GPU, y el modelo large se entrenó con dos.

- **optimizer**: es el optimizador que se va utilizar durante el entrenamiento para ir mejorando las métricas a medida que el entrenamiento avance. Su valor para todos los modelos es: Adam.
- **cos_lr**: este hiperparámetro es un programador que ajusta la tasa de aprendizaje (LR) utilizando una curva de coseno a lo largo de las épocas de entrenamiento. Su valor es True para todos los modelos, lo que significa que el modelo irá ajustando su LR a lo largo de todo el entrenamiento.
- **lr0**: este es el learning-rate o tasa de aprendizaje (LR) con la que va a iniciar el modelo. El valor para todos los modelos va a ser: 0.001.
- **lrf**: este es el learning-rate final, es decir, que utilizando el cos_lr en True, va a ir bajando el LR desde es inicial (lr0) hasta el final si es necesario. Esto es capaz vez que el modelo lo considere. Su valor para todos los modelos es: 0.00001.
- **seed**: esta opción permite la reproducibilidad a la hora de entrenar, ya que sino esta es aleatoria en cada entrenamiento. Por lo que, si se mantienen todas las mismas condiciones a la hora de entrenar, los resultados serán los mismos. Su valor para todos los modelos es: 1234.
- **patience**: este valor es la cantidad de épocas que el modelo va a utilizar como EarlyStopping. Esto quiere decir que, si después de pasada esta cantidad de épocas las métricas no mejoran, el entrenamiento se detiene automáticamente. Su valor para todos los modelos es: 10.
- **project**: se define el nombre del proyecto.
- **name**: se define el nombre de la sesión de entrenamiento del modelo

Métricas de los modelos

hora vamos a ir modelo a modelo viendo las distintas métricas, gráficas y comparaciones entre las imágenes reales y las predicciones que hizo cada modelo. Las métricas que vamos a evaluar van a ser:

- **mean Average Precision at IoU=0.5 (mAP-50)**: Esta métrica calcula el promedio de las precisiones medias en todas las clases, considerando solo las predicciones que tiene un solapamiento (IoU = Insersection over Union) con el valor de 0.5 o más. Esta medida evalúa la calidad de las detecciones en tareas de detección de objetos donde las predicciones deben solaparse en un 50% con el objeto real para ser considerada correcta.
- **mean Average Precision at IoU=0.5:0.95 (mAP50-95)**: Esta métrica es una extensión de mAP50 y calcula el promedio de las precisiones medias en todas las clases a través de varios umbrale IoU que van desde 0.5 a 0.95 con un incremento de 0.05. Ofrece una visión más completa del rendimiento del modelo en una gama más amplia de umbrales de solapamientos, proporcionando una evaluación más rigurosa de la capacidad del modelo para realizar detecciones precisas.
- **Recall**: La tasa de recuperación (recall) mide la proporción de verdaderos positivos que fueron correctamente identificados por el modelo en relación con el total de verdaderos positivos más falsos negativos. Indica qué porcentaje de los objetos reales fueron detectados por el modelo. Un recall alto significa que el modelo detecta una gran parte de los objetos presentes en la imagen.
- **Box(Precision)**: Esta métrica evalúa la precisión de las predicciones de las cajas delimitadoras (BB) del modelo. La precisión de las cajas mide la proporción de predicciones de cajas que son verdaderos positivos en comparación con todas las predicciones hechas. Indica la exactitud de las cajas delimitadoras generadas por el modelo respecto a las cajas reales.

D.3. Modelo nano (yolov8n)

Parámetros

Tabla con parámetros de entrenamiento.

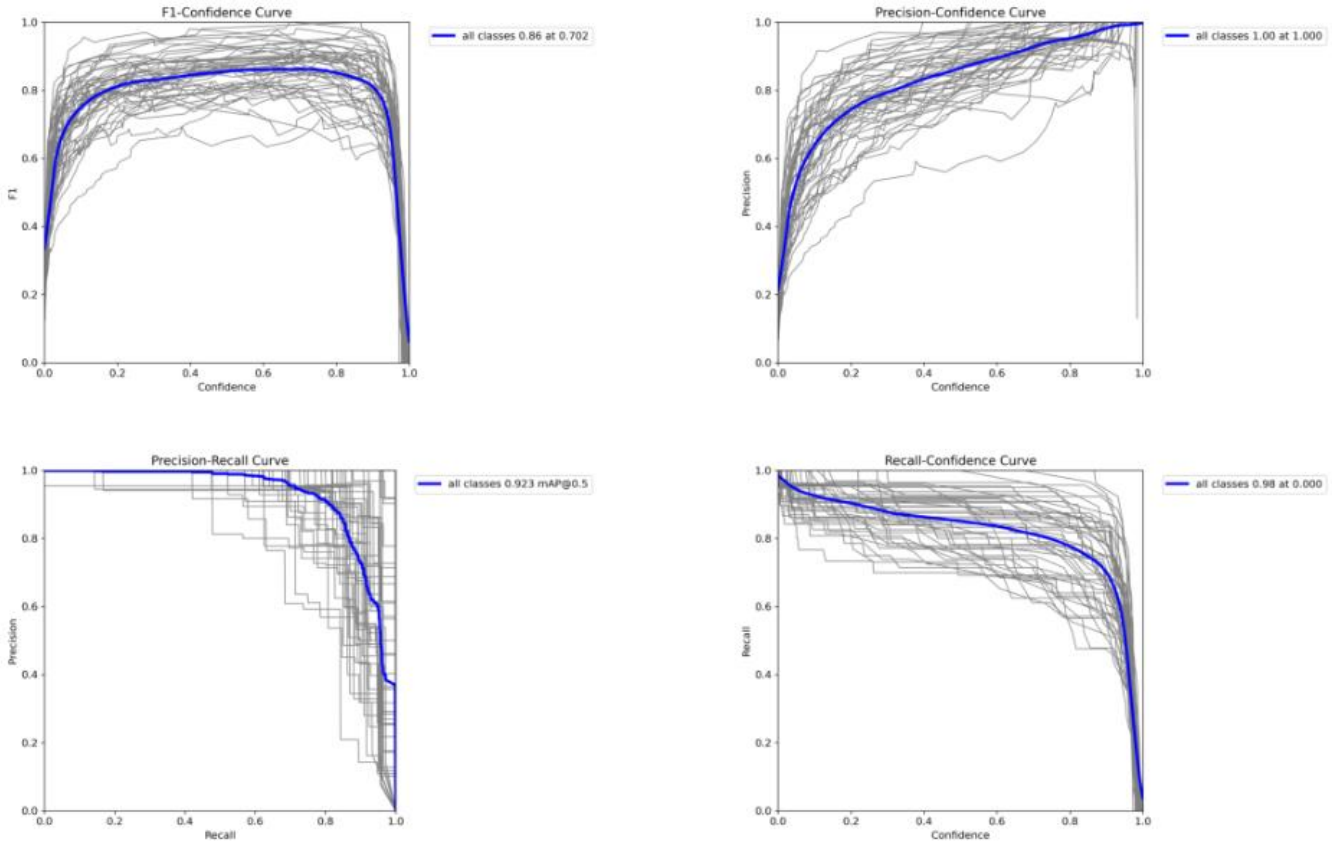
Model	Param	Layers	Epocs (EarlyStop)	GPU (Train)	Time (m:s/epoc)	Time (total)	Imgsz	Batch	Lr0	Lrf
Nano	3 M	225	100 (96)	1	1:25	2h 20’	640	16	0,001	0,00001

Métricas

Tabla comparativa de velocidades y métricas.

Split	Images	Instances	Preproc (ms)	Inferencia (ms)	Loss (ms)	Postpr. (ms)	Precision (Box)	Recall	mAP50	mAP50-95
Val	205	1226	0,3	3,1	0,0	2,4	0,93	0,82	0,922	0,874
Test	205	1205	0,6	16,2	0,0	4,2	0,92	0,818	0,923	0,873

Gráficos



- **F1-Confidence Curve (Curva F1-Confianza)**: Muestra la relación entre la confianza (confidence) de las predicciones del modelo y el F1-score. El F1-score es una medida que considera tanto la precisión como el recall del modelo. Un alto F1-score indica un buen equilibrio entre precisión y recall. En este gráfico, podemos observar cómo varía el F1-score a medida que se ajusta el umbral de confianza del modelo. La curva azul gruesa

representa el rendimiento general del modelo en todas las clases, mientras que las líneas grises representan las diferentes clases individualmente.

La leyenda: "all classes 0.86 at 0.702" significa que el valor F1-score promedio para todas las clases es 0.86 cuando el umbral de confianza es 0.702. Esto indica que el modelo tiene un buen equilibrio entre precisión y recall a ese nivel de confianza.

- **Precision-Confidence Curve (Curva Precisión-Confianza)**: Muestra la relación entre la confianza de las predicciones y la precisión del modelo. La precisión indica la proporción de verdaderos positivos entre todas las instancias clasificadas como positivas. Un alto valor de precisión significa que el modelo tiene una baja tasa de falsos positivos. Este gráfico muestra cómo varía la precisión cuando se cambia el umbral de confianza.

  La leyenda: "all classes 1.00 at 1.000" significa que la precisión promedio para todas las clases es 1.00 cuando el umbral de confianza es 1.000. Esto significa que, cuando el modelo está completamente seguro de sus predicciones (confianza = 1.000), todas las predicciones positivas son correctas (sin falsos positivos).

- **Precision-Recall Curve (Curva Precisión-Recuperación)**: muestra la relación entre la precisión y el recall. El recall mide la proporción de verdaderos positivos entre todas las instancias que son realmente positivas. Este gráfico muestra cómo cambia la precisión en función del recall.

  La leyenda: "all classes 0.923 mAP0.5" significa que el valor mAP promedio para todas las clases es 0.9234 cuando se utiliza un umbral de IoU de 0.5.

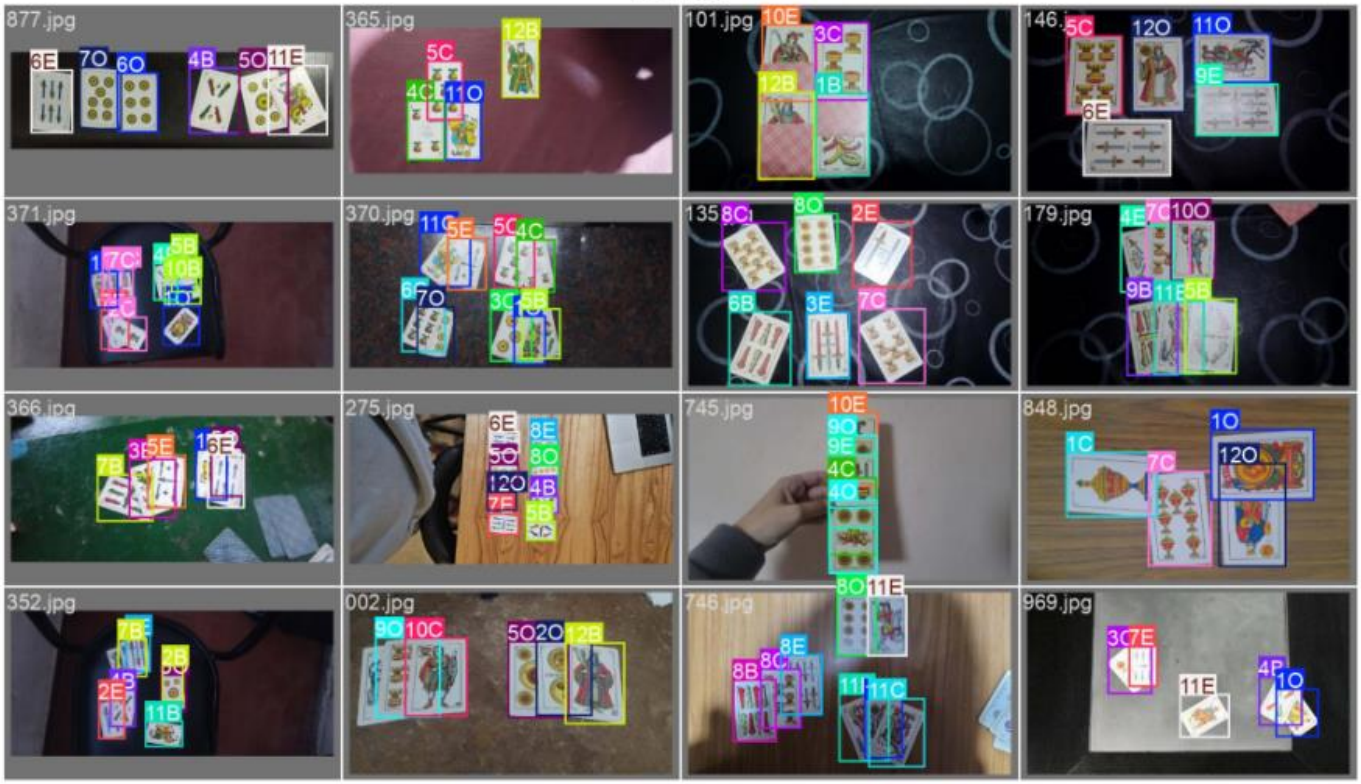
- **Recall-Confidence Curve (Curva Recuperación-Confianza)**: muestra la relación entre la confianza de las predicciones y el recall del modelo. Este gráfico muestra cómo varía el recall a medida que se ajusta el umbral de confianza. Un alto recall significa que el modelo tiene una baja tasa de falsos negativos.

  La leyenda: "all classes 0.98 at 0.000" significa que el recall promedio para todas las clases es 0.98 cuando el umbral de confianza es 0.000. Esto indica que el modelo es capaz de identificar la mayoría de las instancias positivas (bajo número de falsos negativos) incluso con el umbral de confianza más bajo.

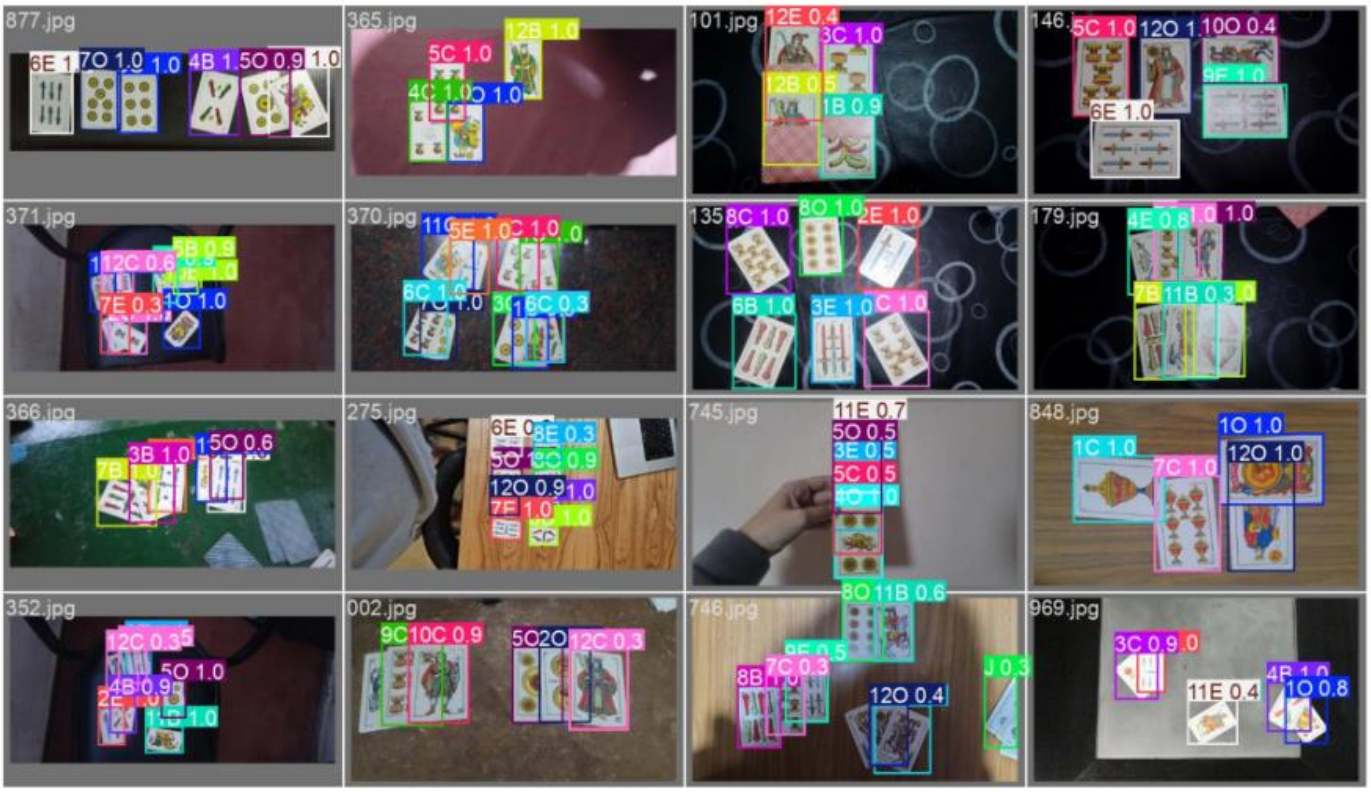
Inferencia en Test

Analizamos la comparativa entre un set de imágenes utilizadas para test (reales) contra las predicciones que hizo el modelo sobre las mismas (predicciones).

Reales



Predicciones



Conclusiones

A partir de todo lo expuesto vemos que el primer modelo presenta resultados muy buenos, más que nada considerando el tamaño y las velocidades. De todos modos, vamos a intentar mejorar esos resultados entrenando un modelo un poco más pesado y así tener como comparativa de rendimientos.

D.4. Modelo medium (yolov8m)

Parámetros

Tabla con parámetros de entrenamiento.

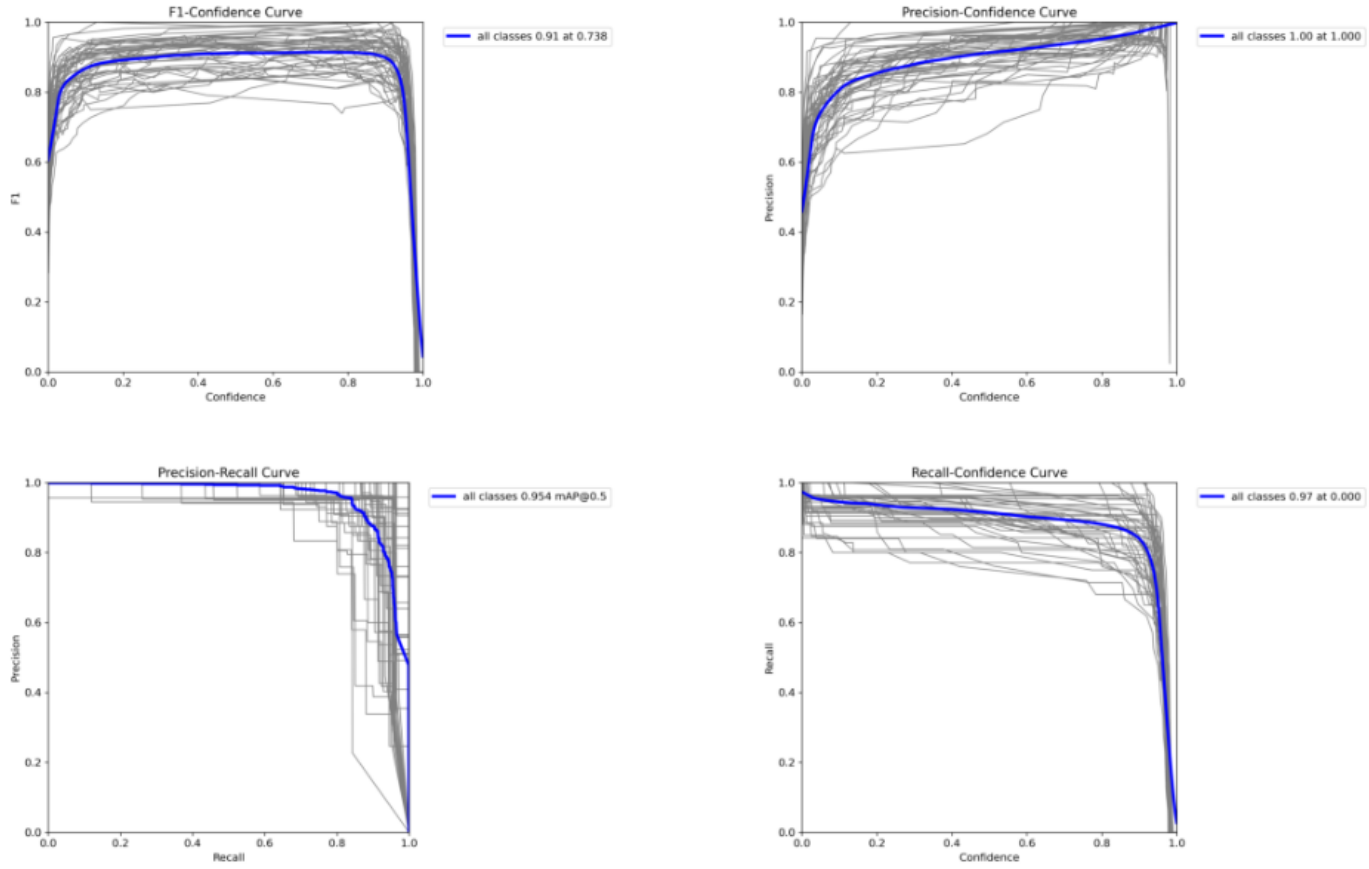
Model	Param	Layers	Epocs (EarlyStop)	GPU (Train)	Time (m:s/epoc)	Time (total)	Imgsz	Batch	Lr0	Lrf
Medium	25,8 M	295	100 (99)	1	1:35	2h 45'	640	16	0,001	0,00001

Métricas

Tabla comparativa de velocidades y métricas.

Split	Images	Instances	Preproc (ms)	Inferencia (ms)	Loss (ms)	Postpr. (ms)	Precision (Box)	Recall	mAP50	mAP50-95
Val	205	1226	0,2	9	0,0	11	0,949	0,885	0,956	0,921
Test	205	1205	0,2	22	0,0	4,4	0,944	0,889	0,954	0,919

Gráficos



- **F1-Confidence Curve (Curva F1-Confianza)**: La leyenda: "all classes 0.91 at 0.738" significa que el valor F1-score promedio para todas las clases es 0.91 cuando el umbral de confianza es 0.738. Ya en esta instancia podemos ver una mejoría con respecto al primer modelo, ya que, el valor F1-score promedio aumentó con un mayor umbral de confianza.

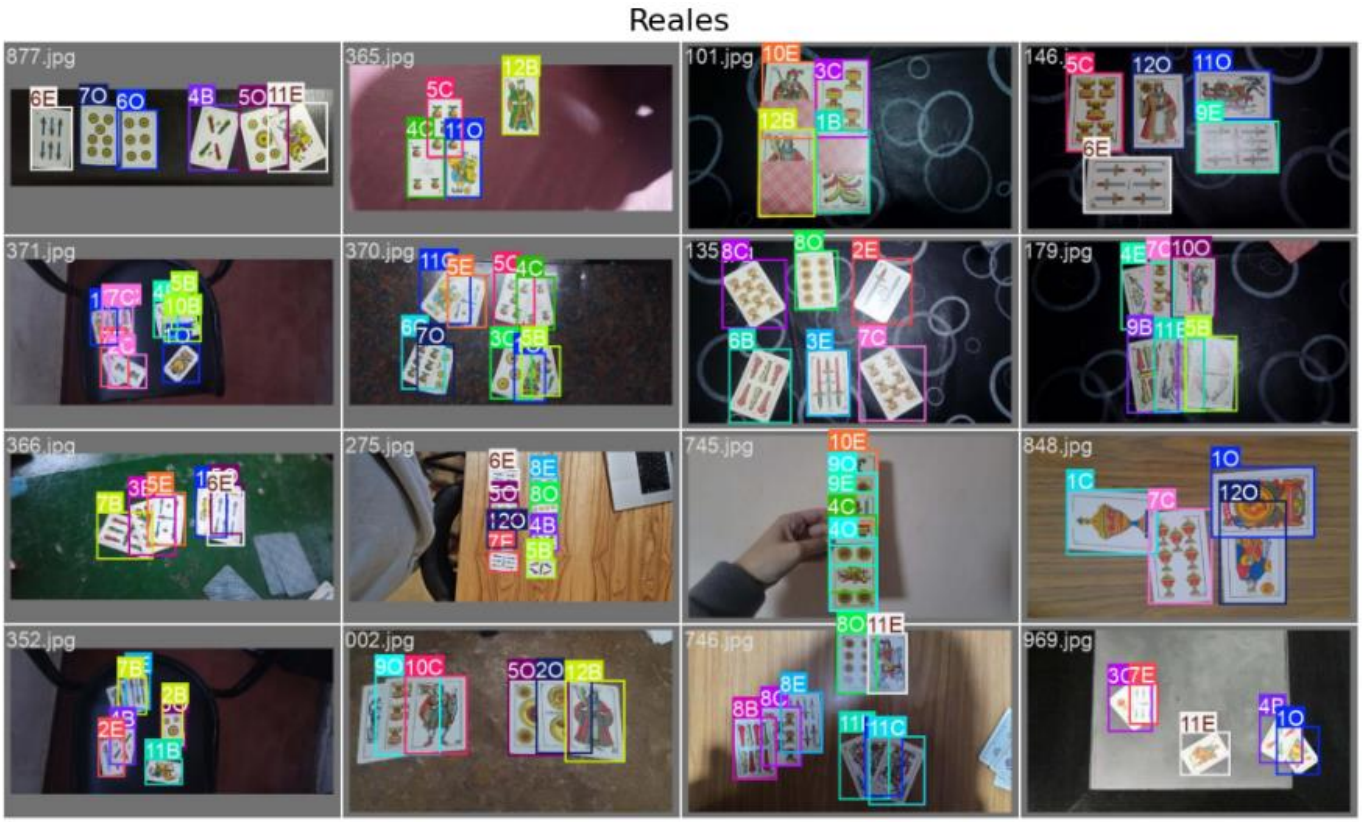
- **Precision-Confidence Curve (Curva Precisión-Confianza)**: La leyenda: "all classes 1.00 at 1.000" significa que la precisión promedio para todas las clases es 1.00 cuando el umbral de confianza es 1.000. Estas métricas se mantuvieron iguales, todas las predicciones positivas son correctas (sin falsos positivos).

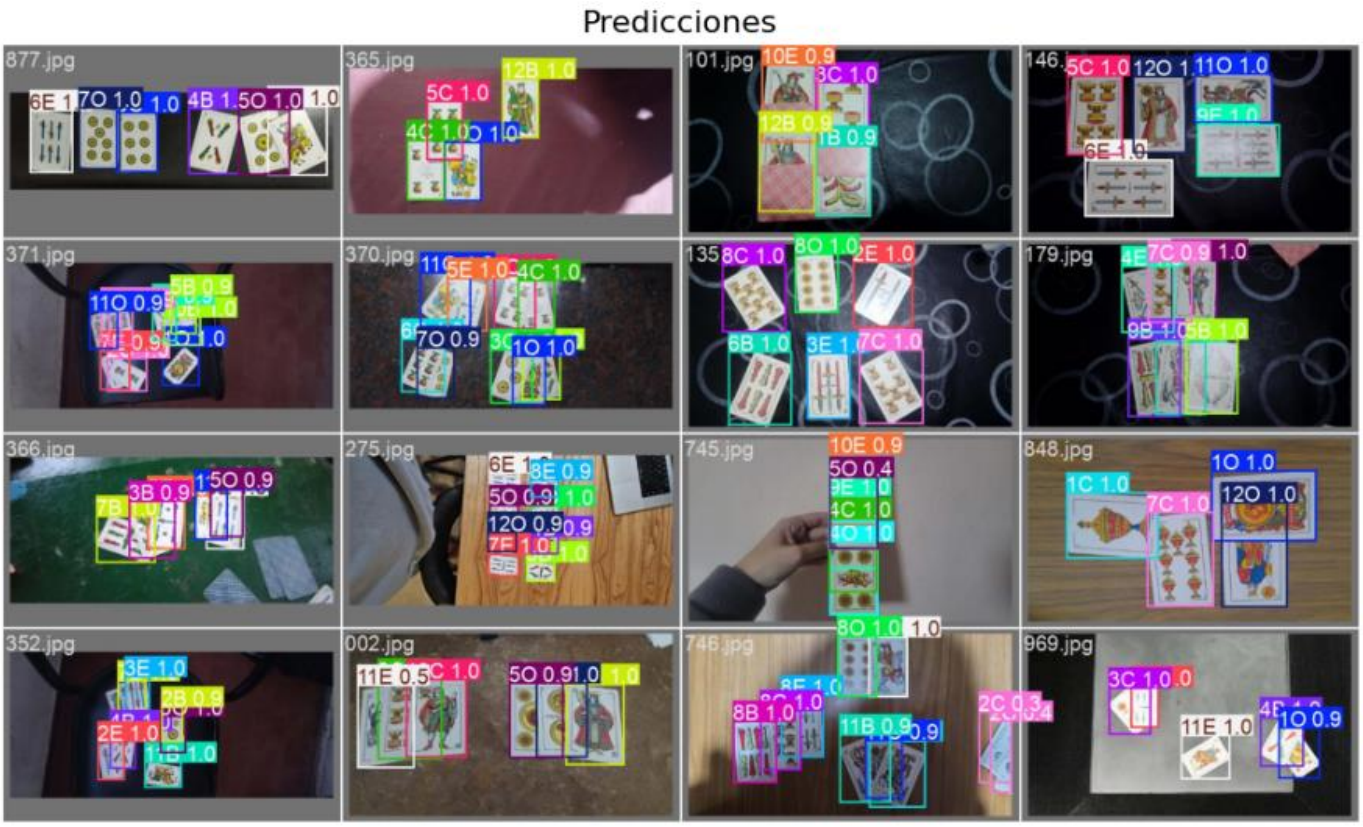
- **Precision-Recall Curve (Curva Precisión-Recuperación)**: La leyenda: "all classes 0.954 mAP0.5" significa que el valor mAP promedio para todas las clases es 0.954 cuando se utiliza un umbral de IoU de 0.5. También mejoró respecto al anterior.

- **Recall-Confidence Curve (Curva Recuperación-Confianza)**: La leyenda: "all classes 0.97 at 0.000" significa que el recall promedio para todas las clases es 0.97 cuando el umbral de confianza es 0.000. Esta métrica se mantuvo igual.

Inferencia en Test

Analizamos la comparativa entre un set de imágenes utilizadas para test (reales) contra las predicciones que hizo el modelo sobre las mismas (predicciones).





Conclusiones

Las métricas expuestas confirman que el modelo es superador respecto al anterior. Suponiendo que la tendencia se debe mantener, se decide experimentar con un nuevo salto de modelo hacia el inmediato seguidor.

D.5. Modelo large (yolov8l)

Parámetros

Tabla con parámetros de entrenamiento.

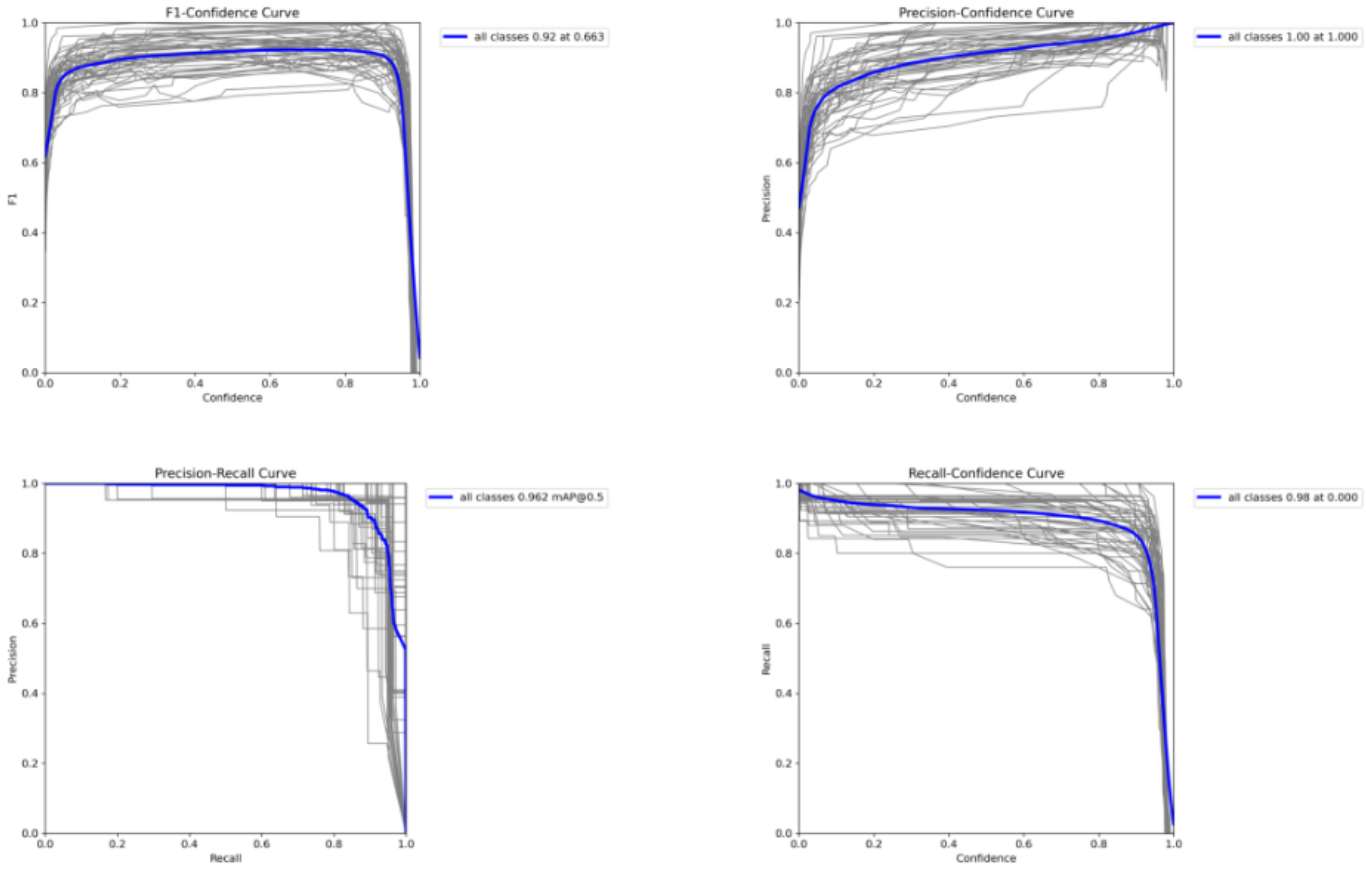
Model	Param	Layers	Epocs (EarlyStop)	GPU (Train)	Time (m:s/epoc)	Time (total)	Imgsz	Batch	Lr0	Lrf
Large	43,6 M	365	100 (100)	2	1:25	2h 36’	640	16	0,001	0,00001

Métricas

Tabla comparativa de velocidades y métricas.

Split	Images	Instances	Preproc (ms)	Inferencia (ms)	Loss (ms)	Postpr. (ms)	Precision (Box)	Recall	mAP50	mAP50-95
Val	205	1226	0,2	12,9	0,0	1,4	0,943	0,895	0,959	0,927
Test	205	1205	0,3	27,6	0,0	10,6	0,939	0,909	0,962	0,929

Gráficos



- **F1-Confidence Curve (Curva F1-Confianza)**: La leyenda: "all classes 0.92 at 0.663" significa que el valor F1-score promedio para todas las clases es 0.912 cuando el umbral de confianza es 0.663. El valor de F1-score mejoró muy levemente pero con un umbral de confianza algo menor que el médium.

- **Precision-Confidence Curve (Curva Precisión-Confianza)**: La leyenda: "all classes 1.00 at 1.000" significa que la precisión promedio para todas las clases es 1.00 cuando el umbral de confianza es 1.000. Estas métricas se mantuvieron iguales, todas las predicciones positivas son correctas (sin falsos positivos).

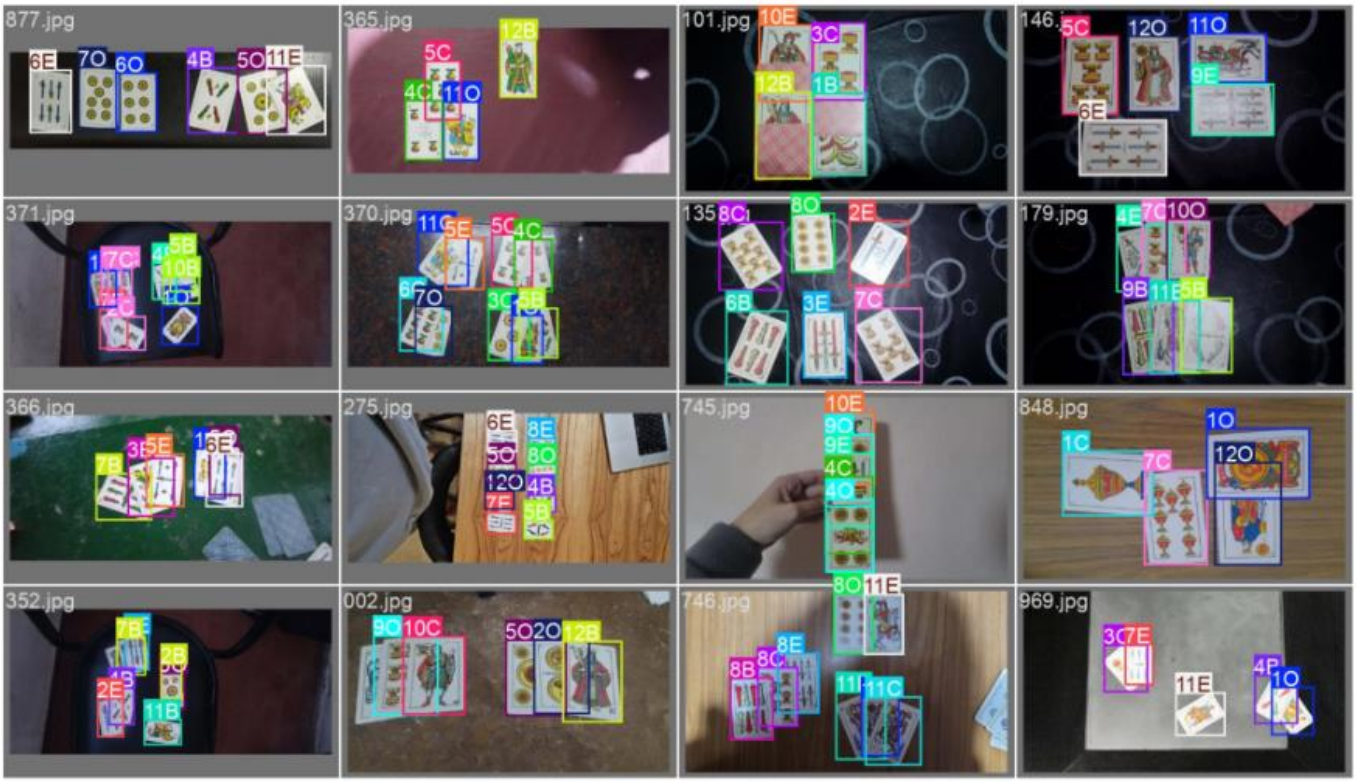
- **Precision-Recall Curve (Curva Precisión-Recuperación)**: La leyenda: "all classes 0.962 mAP0.5" significa que el valor mAP promedio para todas las clases es 0.962 cuando se utiliza un umbral de IoU de 0.5. Esta métrica volvió a mejorar respecto del modelo anterior.

- **Recall-Confidence Curve (Curva Recuperación-Confianza)**: La leyenda: "all classes 0.98 at 0.000" significa que el recall promedio para todas las clases es 0.98 cuando el umbral de confianza es 0.000. Esta métrica se mantuvo igual en los últimos modelos aunque podríamos considerarla casi en el límite.

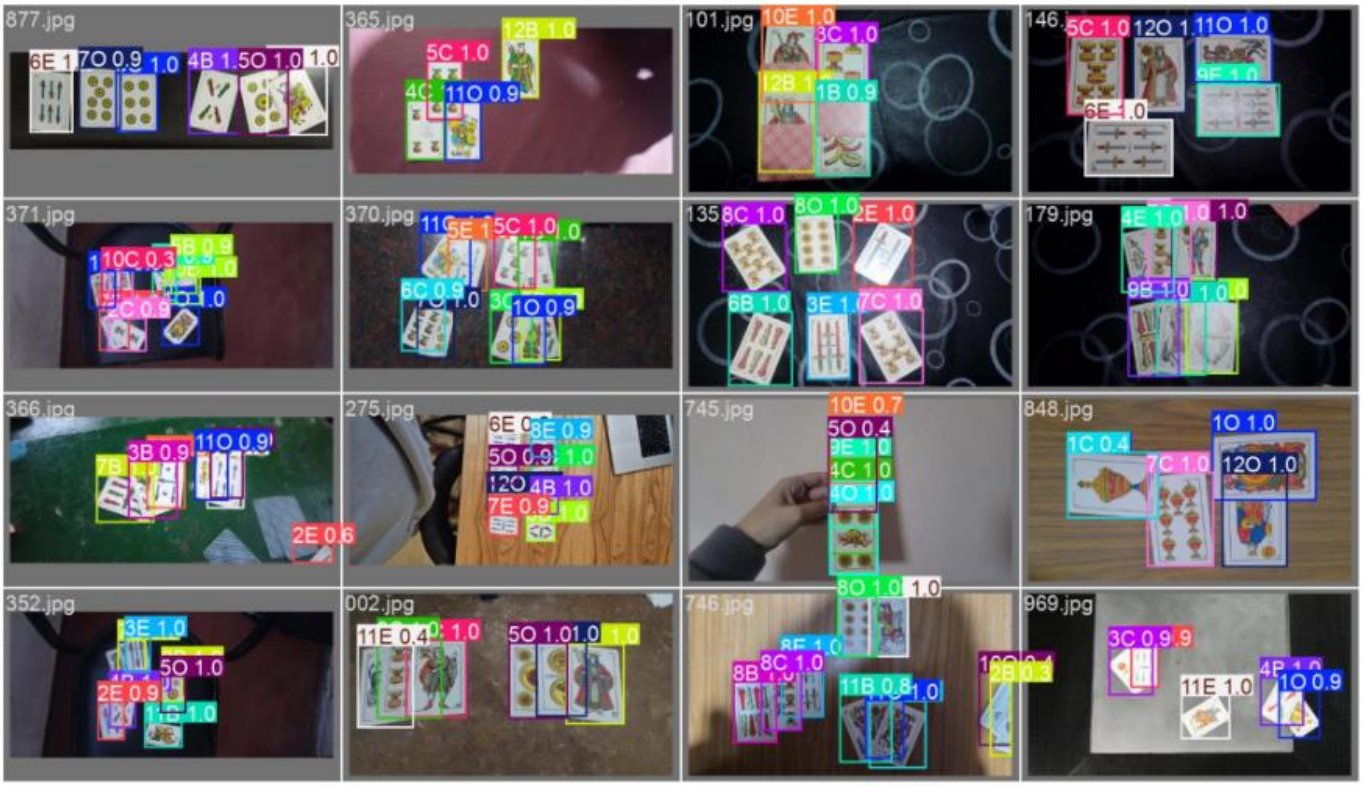
Inferencia en Test

Analizamos la comparativa entre un set de imágenes utilizadas para test (reales) contra las predicciones que hizo el modelo sobre las mismas (predicciones).

Reales



Predicciones



Conclusiones

El modelo large presenta resultados que sobresalen frente a los anteriores. Sus métricas siguen con la tendencia que se suponía que debería haber al pasar de un modelo a otro.

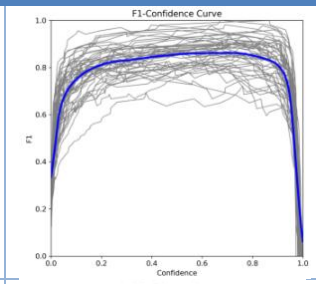
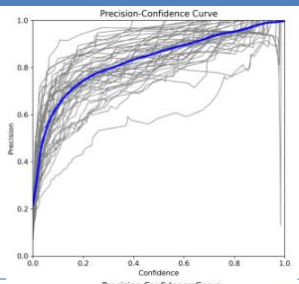
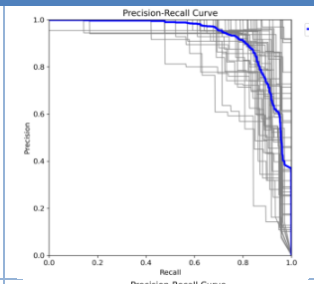
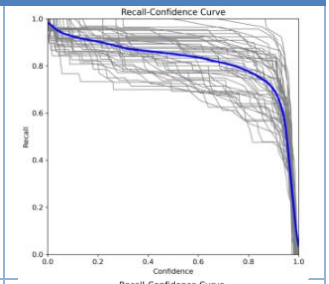
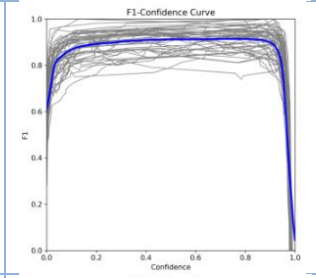
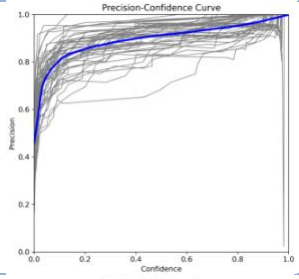
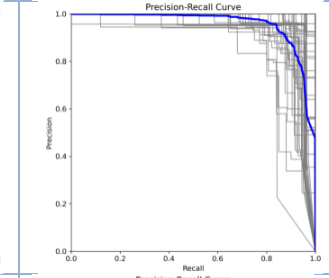
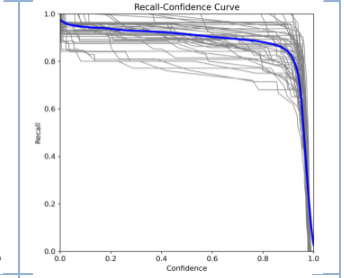
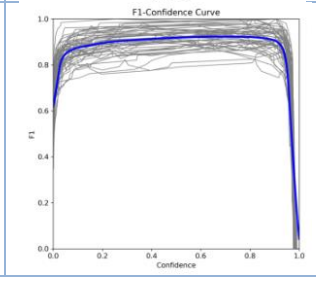
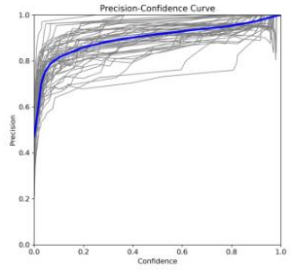
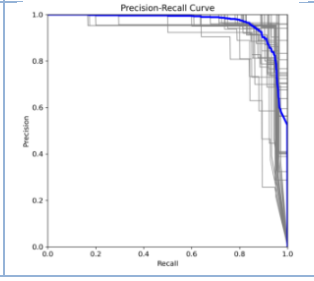
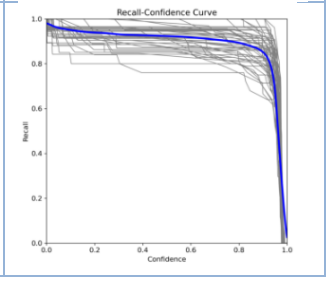
D.6. Resumen comparativo

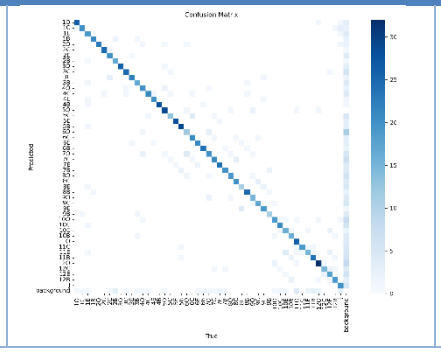
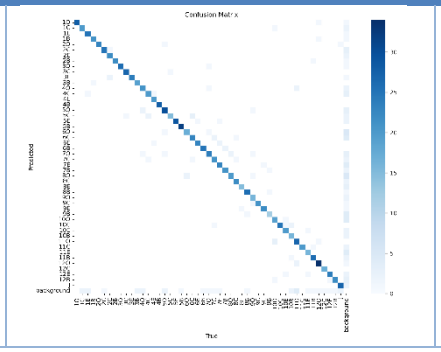
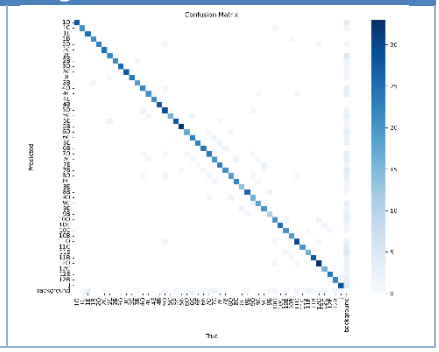
Métricas y tiempos de ejecución

Model	Param	Layers	Epocs (EarlyStop)	GPU (Train)	Time (m:s/epoc)	Time (total)	mAP50 (TRAIN)	mAP50-95 (TRAIN)	mAP50 (VAL)	mAP50-95 (VAL)
Nano	3 M	225	100 (96)	1	**1:25**	**2h 20'**	0,922	0,874	0,923	0,873
Medium	25.8 M	295	100 (99)	1	1:35	2h 45'	0,956	0,921	0,954	0,919
Large	43.6 M	365	100 (100)	2	**1:25**	2h 36'	**0,959**	**0,927**	**0,962**	**0,929**

Model	Images (TEST)	Instances (TEST)	Preproc (ms)	Inferencia (ms)	Loss (ms)	Postpr. (ms)	Total (ms)
Nano	205	1226	0,6	**16,2**	0,0	**4,2**	**21**
Medium	205	1205	0,2	22	0,0	4,4	26,6
Large	205	1205	0,3	27,6	0,0	10,6	38,5

Gráficos

Model	F1-Confidence	Precision-Confidence	Precision-Recall	Recall-Confidence
Nano				
Medium				
Large				

Model	Nano	Medium	Large
Matriz de Confusión			

Conclusiones finales

Analizando detenidamente todas las características de los modelos y sus diferencias en cuanto a performance, podemos afirmar que el modelo ideal para esta actividad es el ***yolov8l*** (large). El modelo large presenta resultados que sobresalen frente a los anteriores y que son muy buenas para la terea que debe desempeñar. Sus métricas siguen con la tendencia que se suponía que debería haber al pasar de un modelo a otro.

No obstante, al modelo **large** podría criticársele que es el peor en la característica temporal. Tal como lo dice la documentación, mayor cantidad de parámetros, un modelo más grande y mayor tiempo de inferencia. Si bien los tiempos de entrenamiento no muestran una desventaja significativa entre ellos, los tiempos de inferencia son los que marcan una diferencia. De todos modos, en este caso, al ser empleado en un tp y no tener que ejecutarlo en tiempo real, no tener especificaciones de hardware donde va a correr, etc. elegimos el modelo más pesado (que es el que mejores resultados dio). En caso de tener otros requisitos que cumplir, podría no ser necesariamente el modelo elegido y quizá resultara una buena opción un modelo más chico con mayor velocidad de inferencia, aunque se sacrifique un par de puntos de métricas.

Otro aspecto interesante a destacar es lo observado en las matrices de confusión de cada modelo. Tal como puede observarse de la comparación, no parece haber una reducción en el rendimiento de la detección y clasificación de las cartas que eran minoritarias en las imágenes del dataset: los 8’s, los 9’s y los comodines. Por lo que se puede concluir que el balance de las clases (que mostraba una pequeña variación en las cartas mencionadas) no condiciona negativamente al rendimiento de los modelos.

Sección E. Ejercicio 4. Evaluación

E.1. Consigna

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.

E.2. Aclaraciones adicionales

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

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.

E.3. Resolución

El notebook provisto tiene una sección para configurar los path a los directorios personalizados. Como primera tarea se deben definir correctamente esas variables para la correcta ejecución de las funciones que permitirán evaluar el trabajo por parte de los docentes de la cátedra. Se adjunta captura de las mismas.

```python
# Nombre del alumno
student_name = 'miguel_mussi'

# Ruta al archivo de pesos
model_path = '/content/drive/MyDrive/UNR/5 - Proc de Imág y Visión Comp. (IA52)/CV_TP_Final/model/weights/best.pt'
model_path_tensor = '/content/drive/MyDrive/UNR/5 - Proc de Imág y Visión Comp. (IA52)/CV_TP_Final/model/weights/best.onnx'

# Ruta al directorio que contiene las imagenes
imgs_dir = '/content/drive/MyDrive/UNR/5 - Proc de Imág y Visión Comp. (IA52)/CV_TP_Final/data/eval/images/val'

# Ruta al directorio de destino de las detecciones
base_dir = '/content/drive/MyDrive/UNR/5 - Proc de Imág y Visión Comp. (IA52)/CV_TP_Final/data/out'
dets_dir = os.path.join(base_dir, student_name)

# Reestablecimiento del directorio de destino (eliminacion)
if os.path.exists(dets_dir):
    shutil.rmtree(dets_dir)
os.makedirs(dets_dir)
```

Optimización de la inferencia

En la sección correspondiente a la carga del modelo pre-entrenado Yolo se agrega la instrucción adicional. Esta corresponde a la exportación del modelo como un tensor para utilizarlo posteriormente con TensorRT y GPU. Para esta finalidad se emplea Onnx, intermediario utilizado para convertir los pesos de modelos de aprendizaje automático entrenados de un formato a otro. Se espera que su implementación mejore los tiempos de inferencia. Por lo que se exploran configuraciones alternativas de hiperparámetros.

```python
# Cargamos el modelo
model = YOLO(model_path)

# Lo exportamos como un tensor para utilizar tensorRT con la GPU
model.export(format='onnx', imgsz=640, dynamic=True)
```

Inferencia

Alineado con la recomendación de la consigna, se somete al script encargado de realizar la inferencia a tres tipos de metodologías de aplicación:

- Inferencia con CPU
- Inferencia con GPU (sin TensoRT)
- Inferencia con GPU (con TensorRT)

Se definen adecuadamente las variables del cálculo del tiempo en la ejecución de cada uno de las opciones para poder mostrar la comparación entre ellos.

```
# Seteo de tiempos de ejecución
execution_time_cpu = 0
execution_time_gpu = 0
execution_time_rt = 0
```

```
[14] # Comparación de los tiempos de ejecución
     print("Comparación de Tiempos de Ejecución:")
     print(f"Tiempo de ejecución en CPU: {execution_time_cpu:.3f} segundos")
     print(f"Tiempo de ejecución en GPU (sin TensorRT): {execution_time_gpu:.3f} segundos")
     print(f"Tiempo de ejecución en GPU (con TensorRT): {execution_time_rt:.3f} segundos")

     Comparación de Tiempos de Ejecución:
     Tiempo de ejecución en CPU: 53.014 segundos
     Tiempo de ejecución en GPU (sin TensorRT): 13.060 segundos
     Tiempo de ejecución en GPU (con TensorRT): 0.000 segundos
```

Los resultados son los siguientes:

Inferencia	Cantidad de imágenes	CPU	GPU (sin TensoRT)	GPU (con TensoRT)
Tiempo de ejecución (s)	32	51,52	37,14	25,13

Guardado de imágenes

Luego de la inferencia (con la metodología deseada) se puede descomentar la línea correspondiente en el siguiente código y se van a guardar las detecciones en el directorio seteado.

```
# Cargamos el modelo
# model = YOLO(model_path)              # Con CPU
model = YOLO(model_path).to(device)     # Con GPU (sin TensorRT)
# model = YOLO(model_path_tensor)       # Con GPU (con TensorRT)

# Ejecutamos la inferencia
results = model(imgs_dir, stream=True)

# Directorio donde van a ir las deteciones
detection_dir = f'{base_dir}/detections/'

# Creamos el directorio si no existe
os.makedirs(detection_dir, exist_ok=True)

# Iteramos sobre los resultados
for result in results:
    # Obtenemos el nombre del archivo de imagen
    img_filename = os.path.basename(result.path)

    # Guardamos la detección
    result.save(filename=f'{detection_dir}/{img_filename}')
```

Escritura del archivo json

Finalmente se ejecuta la celda aportada por la cátedra para generar el archivo envido.json que será utilizado por los docentes de la cátedra para efectuar la evaluación.

```
with open(os.path.join(dets_dir, 'envido.json'), 'w') as jf:
    json.dump(card_js_file, jf, indent=4)
```

Visualización de un fragmento del archivo json:

envido.json


```
{
    "IMG_20240630_173711753_HDR.jpg": {
        "total_cards": 2,
        "cards": {
            "O": [
                "4"
            ],
            "C": [
                "11"
            ],
            "E": [],
            "B": []
        },
        "points": 0,
        "figure": "N/A"
    },
    "IMG_20240630_173737533.jpg": {
        "total_cards": 3,
        "cards": {
            "O": [],
            "C": [
                "1",
                "5"
            ],
            "E": [],
            "B": [
                "2"
            ]
        },
        "points": 26,
        "figure": "C"
    },
    "IMG_20240630_173758691.jpg": {
        "total_cards": 2,
        "cards": {
            "O": [],
            "C": [
                "1"
            ],
            "E": [],
            "B": [
                "2"
            ]
        },
        "points": 0,
        "figure": "N/A"
    },
```


envido.json


```
    "IMG_20240630_173828513_HDR.jpg": {
        "total_cards": 4,
        "cards": {
            "O": [],
            "C": [],
            "E": [],
            "B": []
        },
        "points": 0,
        "figure": "N/A"
    },
    "IMG_20240630_173836085.jpg": {
        "total_cards": 2,
        "cards": {
            "O": [],
            "C": [],
            "E": [
                "7"
            ],
            "B": [
                "6"
            ]
        },
        "points": 0,
        "figure": "N/A"
    },
    "IMG_20240630_173859456.jpg": {
        "total_cards": 1,
        "cards": {
            "O": [],
            "C": [],
            "E": [],
            "B": []
        },
        "points": 0,
        "figure": "N/A"
    },
    "IMG_20240630_173918579.jpg": {
        "total_cards": 2,
        "cards": {
            "O": [
                "8",
                "9"
            ],
            "C": [],
            "E": [],
            "B": []
        },
        "points": 0,
        "figure": "N/A"
```

Sección F. Plantillas para presentar el trabajo

F.1. Consigna

Dado que ejecutaremos el código durante la defensa del trabajo práctico, les facilitamos una estructura de directorios, con un dataset de prueba anotado, y un Colab Notebook en el que deberán incrustar su código para realizar inferencia sobre las imágenes como para cantar envido. Las salidas de estas dos actividades se deben almacenar en diferentes archivos (uno por cada imagen) en formato YOLOv5 para las detecciones de objeto, y uno en formato json con las cartas y los puntos cantados de acuerdo a esas cartas.

Les dejamos el link a zip con la estructura de trabajo, otro link a un video que explica cómo utilizar la plantilla para generar los resultados en el formato correcto.