



Facultad de Ciencias Exactas, Ingeniería y Agrimensura Trabajo Práctico Final

Tecnicatura Universitaria en Inteligencia Artificial

Computer Vision

Alumno:

- Garcia, Timoteo (G-5891/2)

Objetivo del trabajo

Poner a prueba conocimientos adquiridos a lo largo de la materia para así desarrollar un software 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.

Resolución:

Para la resolución de este proyecto lo que hice fue separarlo en etapas. Primero que nada extraer las imágenes que serán la fuente de datos de mi modelo de detección de objetos, armar la estructura de datos en mi Google Drive, para luego preprocesar esa imágenes, normalizarlas, realizar una limpieza de datos de las mismas, verificar que los bounding boxes estén correctos, y quedarme con las imágenes finales con las que iba a armar mi dataset de entrenamiento de mi modelo de Deep Learning. Una vez seleccionadas las imágenes, lo que hice es generarme dos subgrupos, Train y Validation, donde con el primero entrene mi modelo detector de objetos y con el segundo valido que tan bien aprendió mi modelo.

Del total de imágenes pre-seleccionadas, que eran 813 Imágenes, las separe en 80-20 para train-val, es decir, en train inicialmente me quedaron 662

imágenes con sus etiquetas, y en val 151. Inicialmente entrene un modelo Yolov8n, con las 662 imágenes, pero me dio un resultado bastante defectuoso, por lo que decidí realizar aumentación de datos y aumentar la cantidad de imágenes que tengo. A cada imágenes le hice 5 transformaciones por lo que lleve el dataset de entrenamiento de 662 imágenes a 3972, y tome la decisión de que era una suma suficiente de imágenes para entrenar mi modelo final.

Mi modelo final elegido fue un Yolov8n, entrenado a 100 epochas, en batchs de 64. Esto significa que el modelo ha pasado por todos los ejemplos en el conjunto de datos de entrenamiento 100 veces, y que el modelo actualiza sus pesos después de ver 64 ejemplos.

Para entrenar el modelo utilice Google Colab PRO, debí pagar 20 USD para poder acceder a GPU T4 y High Ram, que entreno mi modelo en menos de 10 minutos, ya que con la GPU que brindaba Google esporádicamente me demoraba mas de 24 horas y nunca terminaba.

Luego el modelo entrenado lo descargue localmente, y en VsCode arme la estructura del software “Cantador de envido”. Para esto lo que hice fue detectar 3 cartas como mínimo, y analizar el “palo” y el “valor” de la misma en el envido, y generar la lógica para cantarlo. Por otro lado, utilice una librería de Python, que explicare luego, que canta los puntos del envido con una voz.

Pasemos al código

Dependencias utilizadas

```
import gdown
import glob, shutil, os
import yaml
import random
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import shutil
import torch
import albumentations as A
from albumentations.pytorch import ToTensorV2
from tqdm import tqdm
```

Descargo las imágenes en mi Google Drive

Para descargar las imágenes lo que hice fue utilizar un archivo.zip que un compañero de la TUIA brindo con las imágenes y sus respectivas etiquetas, comprimidas en un mismo archivo. Por lo que descargue desde un Drive Compartido, ese archivo y lo comencé a trabajar en mi Drive.

A su vez realice un esquema de directorios para facilitar el manejo y procesamiento de las imágenes, donde se divide en 3, uno donde caen todas las imágenes recién descargadas (downloaded), otras donde caen luego de ser preprocesadas y normalizadas (bbox_check_folder), y luego donde quedan listas para entrenar el modelo (yolo-format)

```
datasets = [
    "https://drive.google.com/file/d/1u7eRhAuarrWJNNMTsdRIh
18fBL3Lh1ka/view?usp=sharing",
]

BASE_PATH = "drive/MyDrive/TPFINALCV2.0/dataset"
RAW_DATASET = f"{BASE_PATH}/downloaded"
DST_DATASET = f"{BASE_PATH}/yolo-format"
CONVERTER_OUTPUT = f"{BASE_PATH}/bbox_check_folder"

os.makedirs(BASE_PATH, exist_ok=True)
os.makedirs(RAW_DATASET, exist_ok=True)
os.makedirs(CONVERTER_OUTPUT + "/img", exist_ok=True)
os.makedirs(CONVERTER_OUTPUT + "/labels", exist_ok=True)

#Descargo las imágenes
for d in datasets:
    d = d.replace("/view?usp=sharing", "")
    splitted_url = d.split("/")
    url, id, output = splitted_url[:-1], splitted_url[-1],
splitted_url[-1].split("?")[0]

    if "folders" in url:
        output = f"{BASE_PATH}/{output}"
        gdown.download_folder(d, output=output, quiet=False)
    else:
```

```

else:
    d = d.replace("file/d/", "uc?id=")
    output = f"{BASE_PATH}/{output}.zip"
    gdown.download(d, output=output, quiet=False, fuzzy
=True)
    shutil.unpack_archive(output, f"{RAW_DATASET}/")

```

Creo el esquema de mi dataset

El mismo va a contar en un principio con 6 carpetas (3 de img, 3 de labels), para 3 grupos distintos, “train, val, augmented”, que finalmente solo usare train y val para entrenar y validar respectivamente, el modelo.

También en esta etapa defino la arquitectura de mi archivo .yaml, archivo utilizado para darle las indicaciones de la configuración del dataset y de entrenamiento a nuestro modelo Yolo.

```

yolo_ds_dirs = {
    "img_train": DST_DATASET + "/images/train/",
    "img_train_aug": DST_DATASET + "/images/augmented_trai
n/",
    "img_val": DST_DATASET + "/images/val/",
    "lbl_train": DST_DATASET + "/labels/train/",
    "lbl_train_aug": DST_DATASET + "/labels/augmented_trai
n/",
    "lbl_val": DST_DATASET + "/labels/val/"
}

yolo_ds_config = {
"train": "./images/train/",
"val": "./images/val/",
"nc": 51,
"names": [
    "10", "1C", "1E", "1B",
    "20", "2C", "2E", "2B",
    "30", "3C", "3E", "3B",
    "40", "4C", "4E", "4B",
    "50", "5C", "5E", "5B",
    "60", "6C", "6E", "6B",
]

```

```

    "70", "7C", "7E", "7B",
    "80", "8C", "8E", "8B",
    "90", "9C", "9E", "9B",
    "100", "10C", "10E", "10B",
    "110", "11C", "11E", "11B",
    "120", "12C", "12E", "12B",
    "J", "SKIP", "SSKIP"
]
}

# Crear directorios si no existen
for d in yolo_ds_dirs.values():
    os.makedirs(d, exist_ok=True)

```

Normalización de Bounding Boxes

Defino una función que convierta los bounding boxes que vienen en el formato `(x_min, y_min, width, height)` al formato normalizado que utiliza YOLO `(x_center, y_center, width, height)`.

```

def bnd_box_to_yolo_line(box, img_size):
    x_min, y_min, w, h = box[:4] #Obtengo las coordenadas del bounding box
    x_max = x_min + w
    y_max = y_min + h
        # Calculo el centro del bounding box
    x_center = float((x_min + x_max)) / 2 / img_size[1]
    y_center = float((y_min + y_max)) / 2 / img_size[0]

        # Normalizo el ancho y alto del bounding box
    w = float(w) / img_size[1]
    h = float(h) / img_size[0]

    return np.float64(x_center), np.float64(y_center), np.float64(w), np.float64(h)

```

Preproceso y muevo las imágenes normalizadas a la carpeta intermedia “CONVERTER_OUTPUT”

Ahora recorremos todas las imágenes con sus respectivas etiquetas, que descargamos anteriormente, las preprocesamos con la función anteriormente definida, y las movemos a nuestra carpeta intermedia.

```
for label_file in glob.glob(f"{working_dir}/*.txt"):

    base_name = os.path.basename(label_file).split(".")[0]
    image_path = [ glob.glob(f"{working_dir}/{base_name}.{e
xt}") for ext in ["jpg", "png", "jpeg"] ]
    image_path = [ img[0] for img in image_path if img ]
    image_path = image_path[0] if image_path else None

    if not image_path:
        print(f"Image not found for {label_file}")

    continue

    print(f"Processing {image_path}...")
    image = cv.imread(image_path)
    image_h, image_w, _ = image.shape

    new_lines = []

    with open(label_file) as f:
        content = f.readlines()
        for line in content:
            line = line.split()
            if len(line) == 0:
                continue

            elif len(line) != 5:
                class_name = line[0]
                points = [ float(p) for p in line[1:] ]
                points = np.array(points).reshape(-1,2).ast
                ype(np.float32)
                points = points * np.array([image_w, image_
                h])
                points = points.astype(np.int32)
```

```

        x,y,w,h = cv.boundingRect(points)

        cv.rectangle(image, (x, y), (x + w, y + h),
(0, 255, 0), 2)
        cv.imwrite(CONVERTER_OUTPUT + "/img/" + base_name +
".png", image)
        yolo_x, yolo_y, yolo_w, yolo_h = bnd_box_to
_yolo_line([x,y,w,h], [image_h, image_w])

        new_lines.append(f"{class_name} {yolo_x} {y
olo_y} {yolo_w} {yolo_h}\n")

    if new_lines:
        with open(CONVERTER_OUTPUT + "/labels/" + base_name
+ ".txt", "w") as f:
            f.writelines(new_lines)

```

Armo mi dataset de entrenamiento en “yolo_ds_dirs”

Una vez normalice todas mis imágenes ahora queda moverlas a sus respectivas carpetas finales que va a ser usadas por el modelo.

Acá muevo todas a la carpeta de train, y luego saco un 20% de las mismas para validation

```

train_files = glob.glob(f"{CONVERTER_OUTPUT}/labels/*.txt")
print( f"Moving {len(train_files)} files to training set {y
olo_ds_dirs['lbl_train']}")

for f in train_files:
    dest = os.path.join(yolo_ds_dirs["lbl_train"], os.path.
basename(f))
    if not os.path.exists(dest):
        shutil.move(f, yolo_ds_dirs["lbl_train"])
    else:
        print(f"File {os.path.basename(f)} already exists i
n {yolo_ds_dirs['lbl_train']}, skipping.")

```

Muevo las correspondientes a validation

```

# Obtener archivos de etiquetas
label_files = glob.glob(f'{CONVERTER_OUTPUT}/labels/*.txt')

val_files = set( random.choices( label_files, k=int(len(label_files)*0.20) ) )
print( f"Moving {len(val_files)} files to validation set {yolo_ds_dirs['lbl_val']}")

for f in val_files:
    shutil.move(f, yolo_ds_dirs["lbl_val"])

for f_name in glob.glob(f'{yolo_ds_dirs['lbl_val']}/*.txt'):
    base_name = os.path.basename(f_name).split('.')[0]
    img_name = glob.glob(f'{CONVERTER_OUTPUT}/img/{base_name}.*')
    if img_name:
        img_name = img_name[0]
        print(f"Moving {img_name} to {yolo_ds_dirs['img_val']}")
        shutil.move(img_name, yolo_ds_dirs["img_val"])
    else:
        continue

for f_name in glob.glob(f'{yolo_ds_dirs['lbl_train']}/*.txt'):
    base_name = os.path.basename(f_name).split('.')[0]
    img_name = glob.glob(f'{CONVERTER_OUTPUT}/img/{base_name}.*')
    if img_name:
        img_name = img_name[0]
        print(f"Moving {img_name} to {yolo_ds_dirs['img_train']}")
        shutil.move(img_name, yolo_ds_dirs["img_train"])
    else:
        continue

```

Aumentación de datos

Como dije en la introducción, la cantidad de imágenes que me quedaron para entrenar el modelo (662) me parecían pocas, por lo que tome la decisión de a cada imagen aumentarla 5 veces, obteniendo así un total de 3972 imágenes finales.

Y las transformaciones que le realice a cada imagen fueron:

- Rotación
- Desenfoque por Movimiento
- Ruido Gaussiano
- Ajuste Aleatorio de Brillo y Contraste
- Volteo Horizontal

Y luego redimensiono cada imagen a 640×640 px

```
transformations = [
    A.Compose([
        A.Rotate(limit=45, p=1.0),
        A.Resize(640, 640)
    ], bbox_params=A.BboxParams(format='yolo', label_fields
=['category_ids']),

    A.Compose([
        A.MotionBlur(p=1.0),
        A.Resize(640, 640)
    ], bbox_params=A.BboxParams(format='yolo', label_fields
=['category_ids']),

    A.Compose([
        A.GaussNoise(p=1.0),
        A.Resize(640, 640)
    ], bbox_params=A.BboxParams(format='yolo', label_fields
=['category_ids']),

    A.Compose([
        A.RandomBrightnessContrast(p=1.0),
        A.Resize(640, 640)
```

```

        ], bbox_params=A.BboxParams(format='yolo', label_fields
=['category_ids'))),

A.Compose([
    A.HorizontalFlip(p=1.0),
    A.Resize(640, 640)
], bbox_params=A.BboxParams(format='yolo', label_fields
=['category_ids']))
]

#Codigo para aumentar cada imagen
for img_name in tqdm(os.listdir(train_img_dir)):
    img_path = os.path.join(train_img_dir, img_name)
    label_path = os.path.join(train_label_dir, img_name.replace('.png', '.txt'))

    # Leer la imagen
    image = cv.imread(img_path)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # Leer las etiquetas
    labels = read_yolo_labels(label_path)
    bboxes = [label[1:] for label in labels]
    category_ids = [int(label[0]) for label in labels]

    for i, transform in enumerate(transformations): # Generar imágenes aumentadas con diferentes transformaciones
        try:
            # Aplicar las transformaciones
            transformed = transform(image=image, bboxes=bboxes, category_ids=category_ids)
            augmented_image = transformed['image']
            augmented_bboxes = transformed['bboxes']
            augmented_category_ids = transformed['category_ids']

            # Guardar la imagen aumentada
            augmented_image = cv.cvtColor(augmented_image,
cv.COLOR_RGB2BGR)

```

```

        augmented_img_path = os.path.join(augmented_img
_dir, f"{os.path.splitext(img_name)[0]}_aug_{i}.png")
cv.imwrite(augmented_img_path, augmented_image)

# Guardar las etiquetas aumentadas
augmented_labels = [[augmented_category_ids[j]]
+ list(augmented_bboxes[j]) for j in range(len(augmented_bb
oxes))]

augmented_label_path = os.path.join(augmented_l
abel_dir, f"{os.path.splitext(img_name)[0]}_aug_{i}.txt")
write_yolo_labels(augmented_label_path, augment
ed_labels)

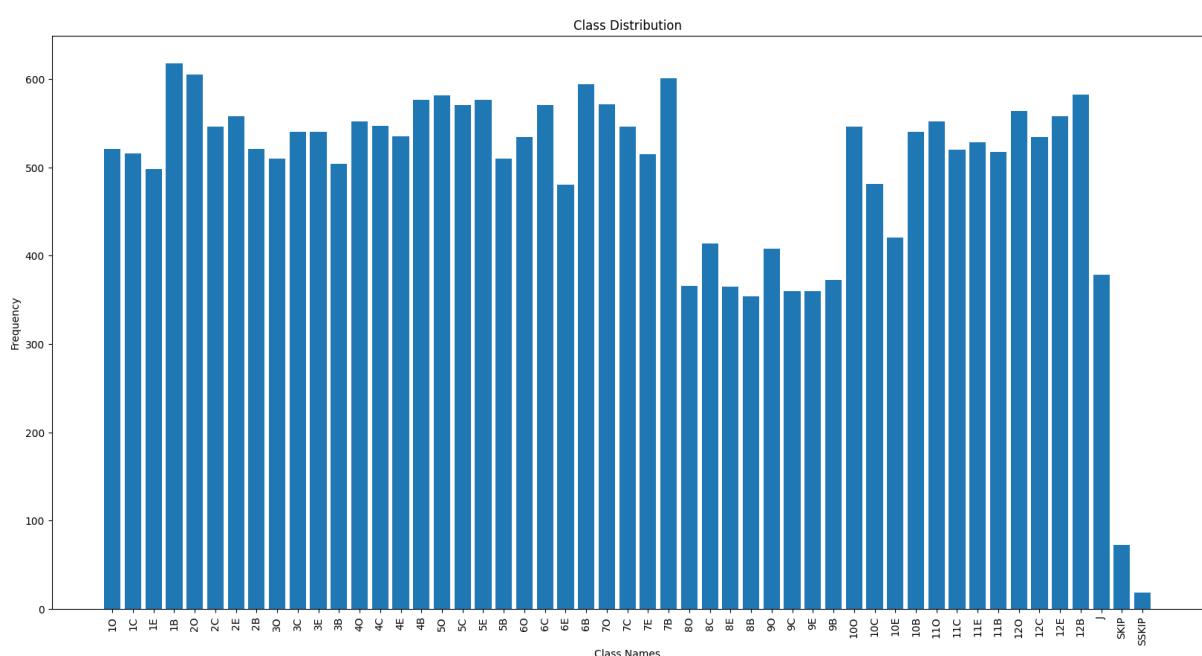
except Exception as e:
    print(f"Error processing {img_name}: {e}")
    continue

```

Análisis de balanceo de clases

Una vez finalizado el preprocesamiento de las imágenes, realice un análisis de las mismas, cuantas cartas tenia en total para entrenar mi modelo, que tan balanceadas estaban las clases, es decir, cuantas etiquetas por carta tenia, para asi detectar algun desbalanceo que pueda hacer sesgar mi modelo.

Adjunto el histograma de clases:



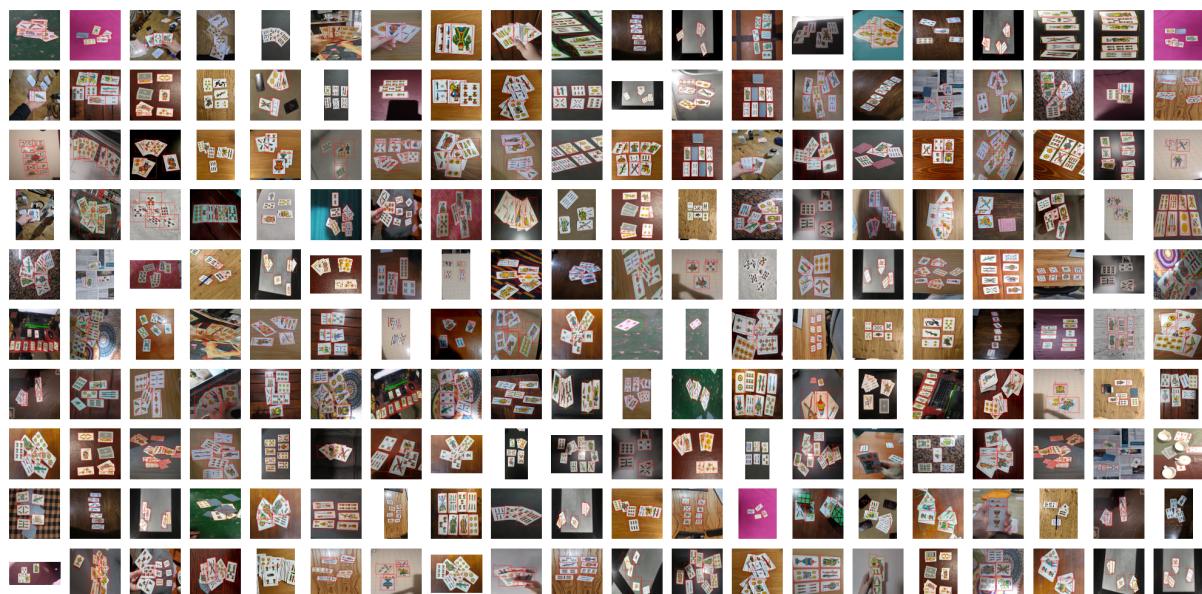
A simple vista podemos ver que la cantidad de clases esta bastante pareja, dando un promedio de **500 ejemplares por carta** y un total de **25144 cartas detectadas**. Podemos ver también que disminuye la cantidad de 8s y 9s, y esto se debe a que quizás algunxs compañerxs obviaron estos por que en el truco no se utilizan.

También como se pueden ver hay 2 categorías adicionales que son "skip" y "sskip" y hacen referencia a labels mal escritas y así aprovechar igual la foto, pero se ve que hay pocas.

Con esta información concluí que es factible entrenar mi modelo Yolo con estas imágenes.

Visualización del dataset

Para visualizar el dataset lo hice con matplotlib, en este caso muestro 200 imágenes al azar. Y podemos ver que las etiquetas estan bien!



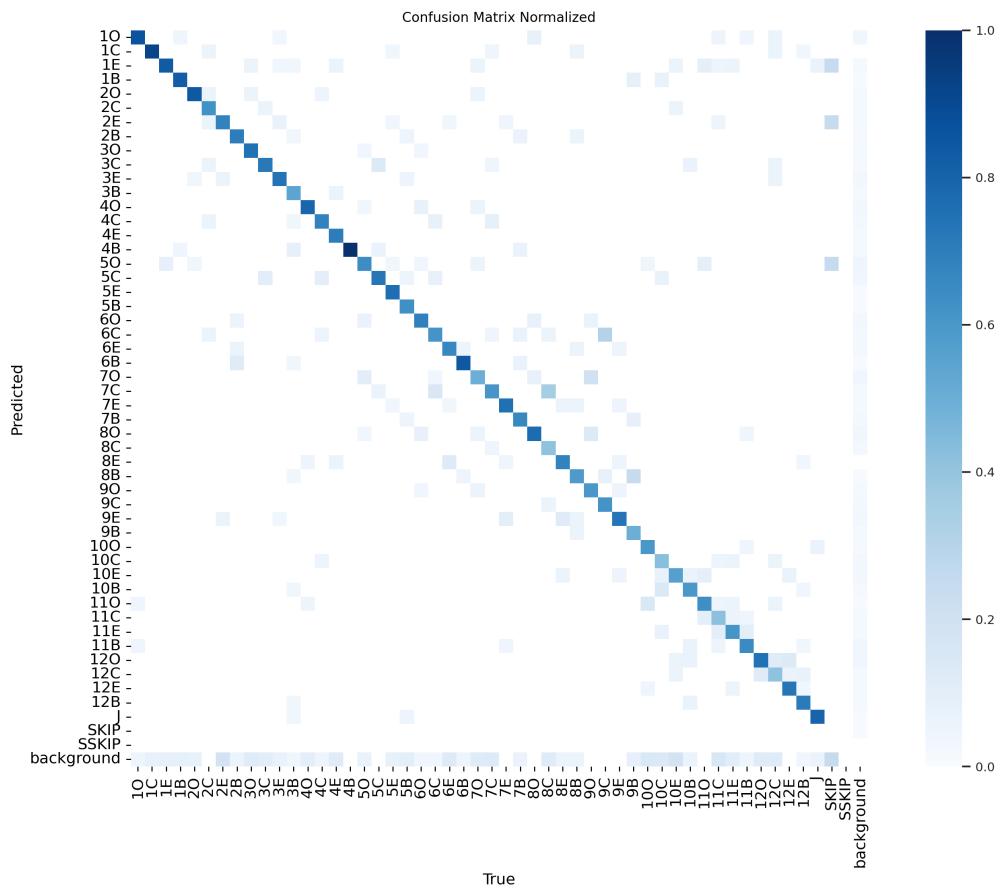
Construcción y entrenamiento del modelo de DL

Luego de instalar la librería Ultralytics, utilice el siguiente comando para con el archivo .yaml, entrenar un modelo de YoloV8n, a 100 epochas en batches de 64.

```
!yolo task=detect mode=train model=yolov8n.pt data=/content/drive/MyDrive/TPFINALCV/dataset/yolo-format/data.yaml epochs=100 batch=64
```

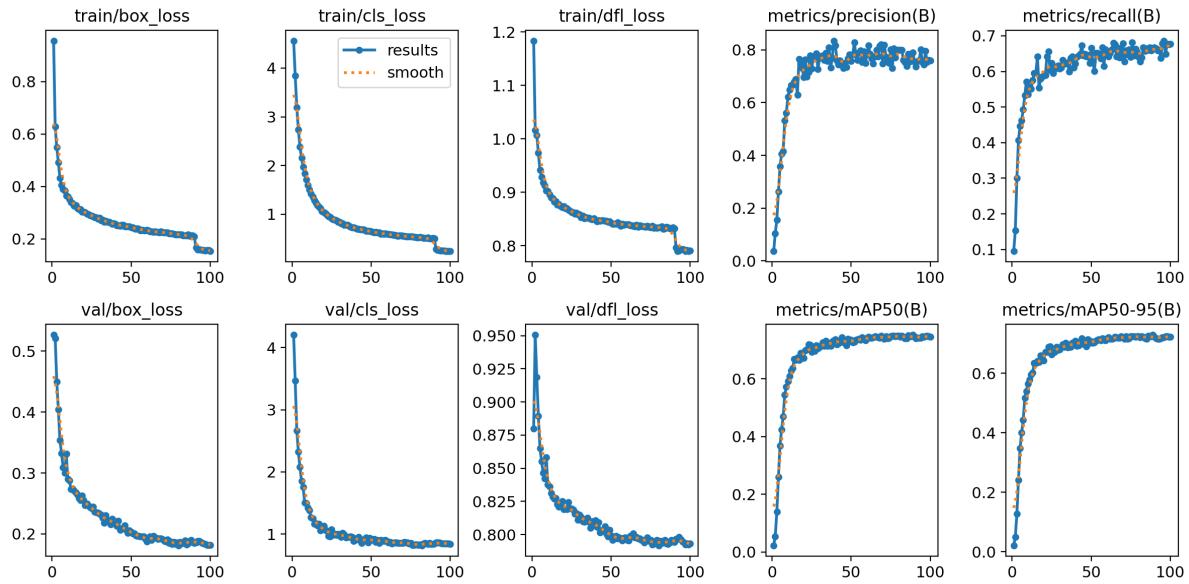
Analicemos el rendimiento del mismo:

Matriz de confusión de clases:



Podemos ver que el modelo acierta en la gran mayoría de los casos la clase correcta

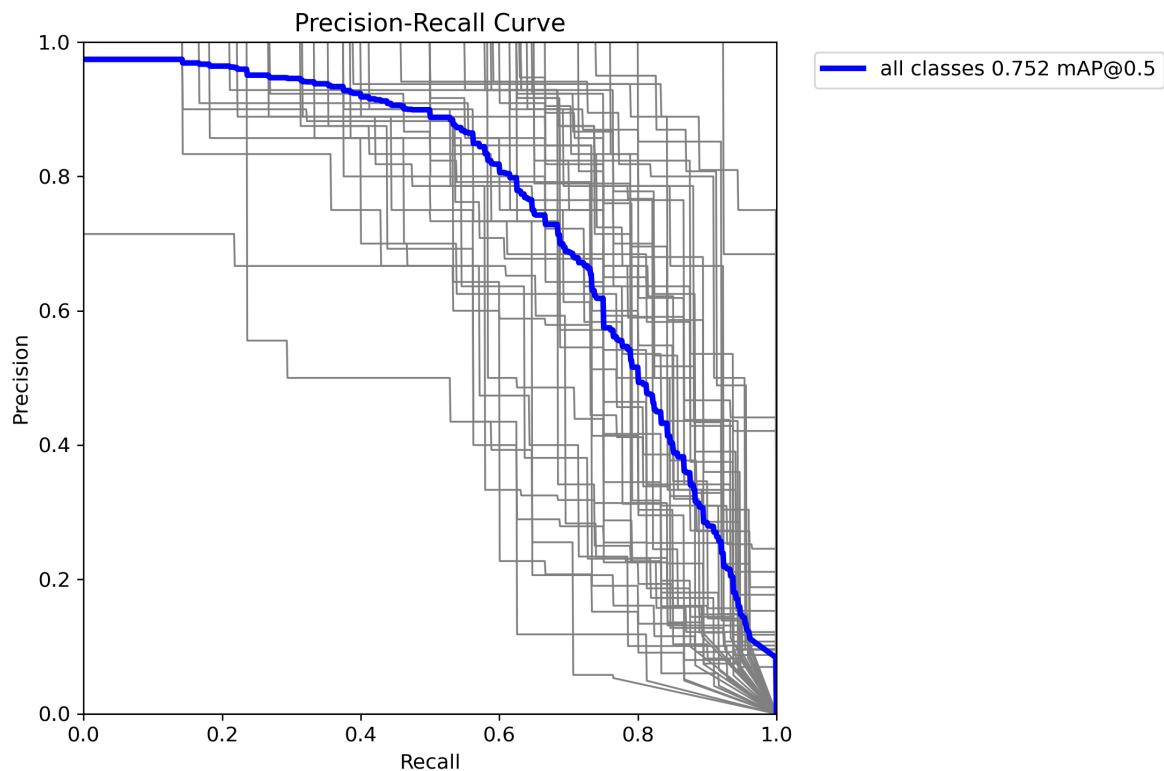
Análisis de métricas con el correr de las épocas:



Basandonos en el recall y la precision, podemos ver que le modelo a partir de la época 20-25 ya "aprende" lo suficiente y tiene un comportamiento muy bueno.

Análisis de la curva ROC, MaP0.5 y MaP0.95:

Teniendo en cuenta que el modelo nos da como resultado en val MaP0.5 = 0.70 y MaP0.95 = 0.67, y la siguiente curva ROC.



- **mAP50 (0.70):**
 - **Significado:** Este valor indica que, en promedio, el modelo tiene una precisión del 70% en la detección de objetos cuando se considera una superposición mínima del 50% entre los BB predichos y los BB verdaderas.
 - **Interpretación:** Un mAP50 de 0.70 es razonablemente bueno y sugiere que el modelo tiene una capacidad decente para detectar objetos con una superposición del 50% o más. Esto indica un buen rendimiento general en la detección de objetos.
- **mAP50-95 (0.67):**
 - **Significado:** Este valor promedia la precisión en una serie de umbrales de IoU que van desde 50% hasta 95%. Proporciona una visión más completa del rendimiento del modelo en términos de precisión en diferentes niveles de superposición.
 - **Interpretación:** Un mAP50-95 de 0.67 es ligeramente inferior al mAP50, lo que indica que el modelo tiene una precisión ligeramente menor en niveles más altos de superposición. Sin embargo, sigue siendo un buen resultado y sugiere que el modelo es relativamente robusto a través de diferentes niveles de umbral de superposición.

En conclusión creo que el modelo tiene un rendimiento sólido y es efectivo en la detección de cartas en este caso, pero hay oportunidades de mejoras tanto del modelo como de la fuente de datos.

Por lo que este modelo es el que utilizaré para la construcción del software capaz de cantar envío en base a la detección de 3 cartas.

Software capaz de cantar envío en base a 3 cartas

Para armar el software final lo dividi en 3 programas, uno para detectar cartas que tengo, el palo y el numero, otro con la lógica del envío y otro con la capacidad de decir en voz alta cuantos puntos tengo.

La lógica es la siguiente mediante la cámara web de mi notebook voy a hacer que el modelo detecte 3 cartas, una vez detectadas las 3 cartas voy a obtener sus palo y su numero. Estos 6 datos (2 de cada carta) se los envió a mi código con la lógica de cantar envío, que va a decidir cuánto envío tengo y pasarselo al código de la voz para que lo cante.

Vamos con los códigos:

Detección de cartas y obtención de palos y números

Aca lo que hacemos es cargar el modelo pre-entrenado en google colab, lo importamos con la libreria ultralytics, y lo utilizamos para detectar las cartas que le vamos a “mostrar” a nuestra webcam.

Nos quedamos con las 3 predicciones de mayor confianza, porque puede pasar que mientras que enfoca no detecte bien la carta.

A su vez le puse un threshold de frames de detección, es decir que no me haga la predicción frame a frame, si no que se tome 5 frames para predecir. Este es un Hiperparametro que tambien se puede ir optimizando para obtener mejores resultados.

```
# import streamlit as st
import cv2
import numpy as np
from PIL import Image
from ultralytics import YOLO
from envido import contarEnvido
from canto import hablar

# Cargar el modelo YOLOv8
model_path = r"train1344/train13/weights/best.pt"
model = YOLO(model_path)

def predict(image):
    # Realiza la predicción usando el modelo YOLOv8
    results = model.predict(image)
    return results

def draw_boxes(image, results):
    top3_labels = []
    for result in results:
        # Ordenar las detecciones por confianza en orden descendente
        sorted_boxes = sorted(result.boxes, key=lambda box: box.conf)[-3:]

        # Tomar las 3 detecciones con mayor confianza
        top3_boxes = sorted_boxes[:3]
```

```

# Recorrer las 3 detecciones principales
for box in top3_boxes:
    x_min, y_min, x_max, y_max = map(int, box.xyxy[0])
    conf = box.conf[0]
    cls = int(box.cls[0])
    label = f'{model.names[cls]} {conf:.2f}'
    top3_labels.append(label.split()[0]) # Almacena
    cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (255, 0, 0), 2)
    cv2.putText(image, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36, 255, 12), 2)
return image, top3_labels

def main():
    # Abrir la webcam
    cap = cv2.VideoCapture(0)
    frames_thr = 5
    cont = 0
    while True:
        # Capturar frame-by-frame
        ret, frame = cap.read()

        if not ret:
            break
        # Realizar predicciones
        results = predict(frame)

        # Dibujar las bounding boxes
        frame, top3 = draw_boxes(frame, results)
        # Mostrar el frame con las detecciones

        if len(top3) == 3:
            cont += 1
            if cont >= frames_thr:
                txt = contarEnvido(top3)
                hablar(txt)

        cv2.imshow('YOLOv8 Object Detection', frame)
        # Salir con la tecla 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

```

```

# Liberar la cámara y cerrar todas las ventanas
cap.release()
cv2.destroyAllWindows()

if __name__ == "__main__":
    main()

```

Una vez que detecto las cartas se las pasa a las siguiente funciones

Lógica del envido

La predicción del modelo Yolo será una lista con 3 valores, que serán los 3 labels, ejemplo ["1E", "2E", "11B"]. Por lo que inicialmente lo que hago es obtener la información de cada carta, obtengo el palo y el numero, y al numero inmediatamente lo transformo en "valor", es decir, en el envido los números 10,11,12 valen 0, entonces si me toco alguno de esos lo transformo en 0 en valor de envido.

Luego aplico la lógica del envido, chequeo si hay alguno palo coincidente y sumo los valores + 20, y devuelvo el "canto".

Si tenemos 3 palos iguales, como se juega sin flor, lo que hago es quedarme con los 2 valores mas grandes, y canto envido con esos dos.

Va el codigo:

```

from collections import Counter

def dividirmano(mano):
    carta1 = mano[0]
    carta2 = mano[1]
    carta3 = mano[2]

    return carta1, carta2, carta3

def extraerDatos(carta):
    palo = carta[-1]
    aux = carta[0:-1]
    valor = int(aux)

    if (valor >= 10):
        valor = 0

```

```

        return palo, valor

def contarEnvido(mano):
    carta1, carta2, carta3 = dividirmano(mano)
    palo1, aux1 = extraerDatos(carta1)
    palo2, aux2 = extraerDatos(carta2)
    palo3, aux3 = extraerDatos(carta3)

    if ((palo1 == palo2) and (palo2 == palo3)):
        valores = [aux1, aux2, aux3]
        valores.sort()
        canto = valores[1:]
        suma = 20 + canto[0] + canto[1]
        texto = "La suma del envido es: " + str(suma)

    elif ((palo1 == palo2) and (palo2 != palo3)):
        suma = 20 + aux1 + aux2
        texto = "La suma del envido es: " + str(suma)
        #print("La suma del envido es: {}".format(suma))

    elif ((palo1 == palo3) and (palo1 != palo2)):
        suma = 20 + aux1 + aux3
        texto = "La suma del envido es: " + str(suma)
        #print("La suma del envido es: {}".format(suma))

    elif ((palo2 == palo3) and (palo1 != palo2)):
        suma = 20 + aux2 + aux3
        texto = "La suma del envido es: " + str(suma)
        #print("La suma del envido es: {}".format(suma))

    else:
        #print("Mentiste. No tenias nada para el envido")
        suma = max([aux1, aux2, aux3])
        texto = "La suma del envido es: " + str(suma)
        #print("La suma del envido es: {}".format(suma))

```

```
    return texto
```

Por ultimo, utilizando la libreria “pyttsx3”, hacemos que una voz cante el envido siguiendo la logica de arriba.

```
import pyttsx3
def hablar(texto):
    engine = pyttsx3.init()
    engine.say(texto)
    engine.runAndWait()
```

Y con esto damos por finalizado el trabajo!

Conclusiones del trabajo y la materia.

Comenzando con el trabajo, fue un trabajo muy desafiante ya que nunca había hecho algo similar, pero fue muy divertido y sobretodo muy útil. Quizás hubiese estado bueno que sea un trabajo en pareja, ya que había mucho por desarrollar y quizás haciéndolo de a 2 se podía empujar un poco mas el limite y obtener mejores resultados, y además al haber sido un trabajo tan desafiante pensarlo de a 2 creo que hubiese estado interesante. Por otro lado el hecho de haber sido individual me llevo a conocer una faceta mia que no conocia je, ya que en una etapa del trabajo estaba totalmente desconcertado, y fueron fin de semanas enteros haciendo research, estudiando, buscando informacion, para llevarlo adelante. Espero haber cumplido con las consignas!

Yendo un poco mas a la materia, me pareció de lo mas interesante de la carrera, creo que el computer vision es una rama increíble, un mundo por explotar, y con gran futuro y demanda laboral. A su vez creo que da mucho hincapié para crear algo, solucionar algún problema de la vida real con las herramientas que nos fueron dando.

A pesar de haber sido la primera camada me pareció excelente la cantidad de material teórico y practico que nos queda, felicito a la catedra por eso, muy buen laburo.

Quedo a disposición para lo que necesiten!

Muchas gracias!

Timoteo Garcia!