CEIA Visión por Computadora II
Trabajo Práctico Integrador

Integrantes:

Oksana Bokhonok

Ezequiel Guinsburg

Objetivo

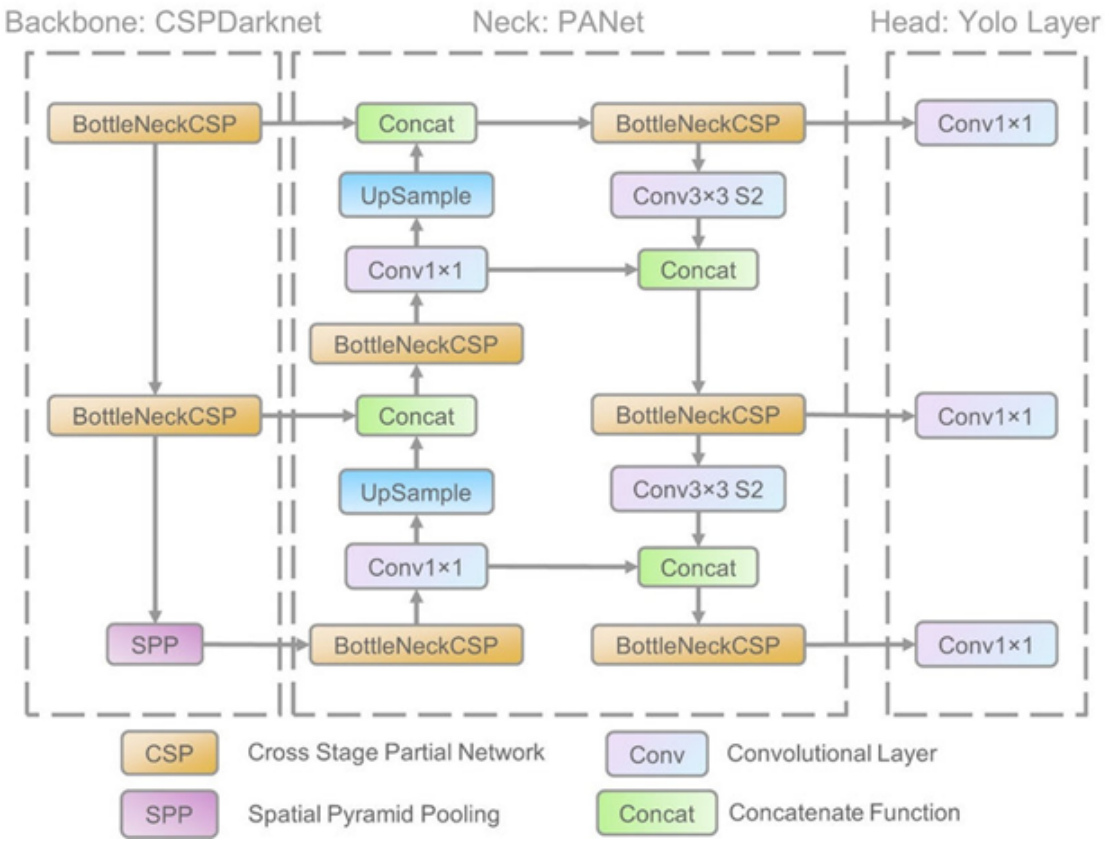
El objetivo del presente trabajo es poder parametrizar y comparar diferentes modelos de visión por computadora sobre el dataset de TACO (ver LINK).

Dicho dataset se trata de imágenes fotográficas de residuos generados por humanos en diferentes entornos, naturales y urbanos. Las imágenes fueron manualmente etiquetadas y segmentadas en el mismo formato que el dataset COCO (ver LINK). Esto nos permite realizar "transfer learning" a partir de los modelos entrenados sobre dataset de COCO.

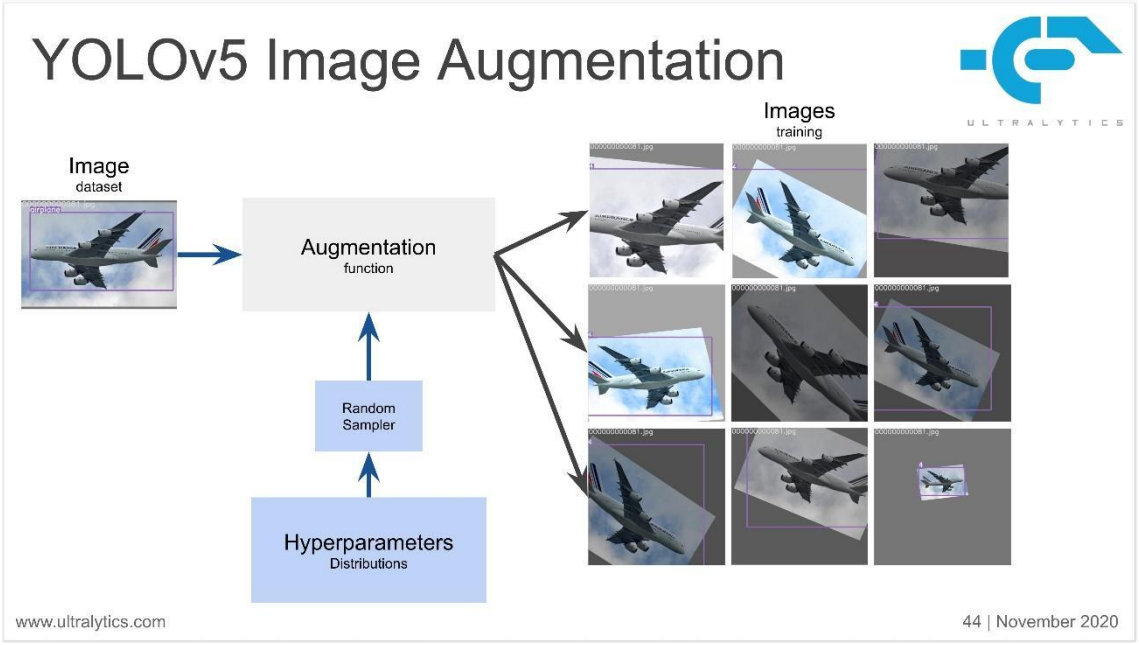
Las características del dataset TACO son las siguientes:

- **Cantidad de imágenes:** 1500
- **Cantidad de etiquetas:** 4784
- **Cantidad de categorías:** 60
- **Estructura de las anotaciones:** Límites del bounding box, categorías corrrespondientes y segmentaciones a nivel pixel en archivos .yaml

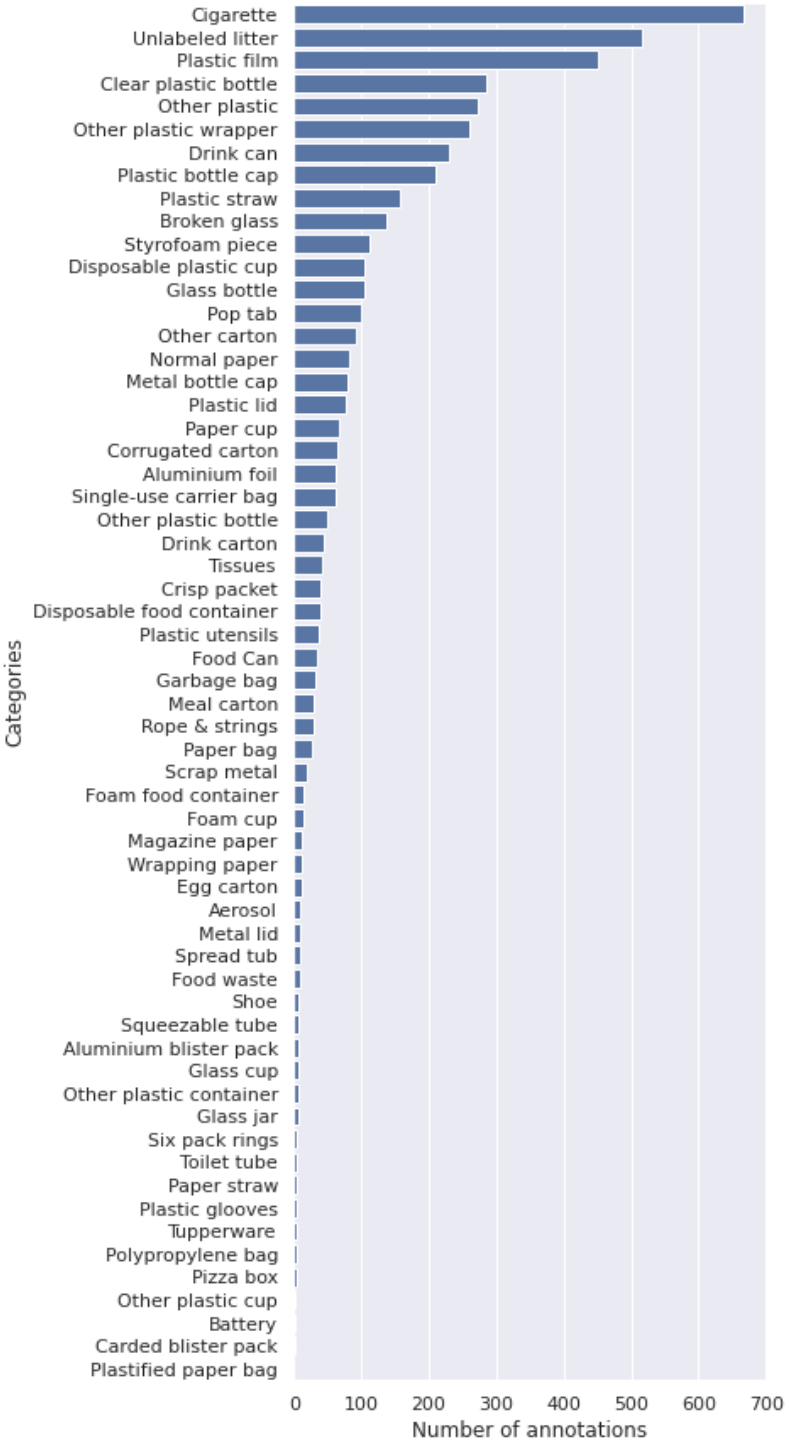
El modelo base que se utiliza para este proyecto es el YOLO ("You Look At Once") en su versión 5 (YOLOV5) que divide las imágenes en grillas, donde cada una de ellas es responsable de detectar los objetos que contienen. Las características generales del algoritmo son su velocidad y precisión. Se compone de cuatro etapas bien diferenciadas: input, backbone, neck y salida como se ve en el siguiente esquema:



El modelo cuenta con la posibilidad de realizar Data Augmentation internamente, integrado con el paquete de código abierto Albumentations (Ver LINK). La siguiente imagen muestra ejemplos de las posibilidades de esta librería:



Distribución de clases en dataset TACO:



Corridas y modelos entrenados:

- **Estructura**: YOLO V5 / **Pesos pre-entrenados**: Sobre dataset COCO

#1.

!python train.py --img 640 --epochs 50 --data taco.yaml --batch-size 64 --cache

- Tamaño de imagen cuadrada: img 640
- Cantidad de epochs: 50
- Archivo de configuración que contiene path train/val y cantidad y tipo de categorías del dato a utilizar para training: taco.yaml
- Batch-size: 64
- Hiperparametros: default con data augmentation

#2.

!python train.py --img 640 --epochs 50 --data taco.yaml --weights yolov5s.pt --hyp hyp_noDA.yaml --batch-size 64 --freeze 10 --cache

- Transfer Learning, capas frizadas: 10 (backbone de yolov5)
- Tamaño de imagen cuadrada: img 640
- Cantidad de epochs: 50
- Archivo de configuración que contiene path train/val y cantidad y tipo de categorías del dato a utilizar para training: taco.yaml
- Batch-size: 64
- Hiperparametros: default sin data augmentation

#3.

!python train.py --img 1280 --epochs 50 --data taco.yaml --weights yolov5x6.pt --hyp hyp_noDA.yaml --batch-size 6 --freeze 10 --cache

Por recomendación de yolov vamos a agrandar la imagen a 1280 y vamos a utilizar los pesos preentrenados para imágenes de este tamaño con el mejor performance según https://github.com/ultralytics/yolov5/releases

- Transfer Learning, capas frizadas: 10 (backbone de yolov5)
- Tamaño de imagen cuadrada: img 1280 (pesos yolov5x6.pt, preentrenados en imágenes de 1280)
- Cantidad de epochs: 50

- Archivo de configuración que contiene path train/val y cantidad y tipo de categorías del dato a utilizar para training: taco.yaml
- Batch-size: 6 (A mayor batch, se satura ram)
- Hiperparametros: default sin data augmentation

#4.

!python train.py --img 640 --epochs 50 --data taco.yaml --weights yolov5s.pt --batch-size 64 --freeze 10 --cache

- Transfer Learning, capas frizadas: 10 (backbone de yolov5)
- Tamaño de imagen cuadrada: img 640 (pesos preentrenados en imágenes de 640, yolov5s.pt)
- Cantidad de epochs: 50
- Archivo de configuración que contiene path train/val y cantidad y tipo de categorías del dato a utilizar para training: taco.yaml
- Batch-size: 64
- Hiperparametros: default con data augmentation

#5.

!python train.py --img 1280 --epochs 50 --data taco.yaml --weights yolov5x6.pt --batch-size 8 --freeze 10 --cache

- Transfer Learning, capas frizadas: 10 (backbone de yolov5)
- Tamaño de imagen cuadrada: img 1280 (pesos yolov5x6.pt, preentrenados en imágenes de 1280)
- Cantidad de epochs: 50
- Archivo de configuración que contiene path train/val y cantidad y tipo de categorías del dato a utilizar para training: taco.yaml
- Batch-size: 8 (A mayor batch, se satura ram)
- Hiperparametros: default con data augmentation

HIPER-PARÁMETROS:

Para todas las corridas:

- **Dataset de entrenamiento**: TACO
- **lr0:** 0.01 initial learning rate (SGD=1E-2, Adam=1E-3)
- **lrf: 0.1** final OneCycleLR learning rate (lr0 * lrf)
- **momentum:** 0.937 SGD momentum/Adam beta1
- **weight_decay**: 0.0005 optimizer weight decay 5e-4
- **warmup_epochs:** 3.0 warmup epochs (fractions ok)
- **warmup_momentum:** 0.8 warmup initial momentum
- **warmup_bias_lr:** 0.1 warmup initial bias lr
- **box:** 0.05 box loss gain
- **cls: 0.3** cls loss gain
- **cls_pw:** 1.0 cls BCELoss positive_weight
- **obj:** 0.7 obj loss gain (scale with pixels)
- **obj_pw:** 1.0 obj BCELoss positive_weight
- **iou_t:** 0.20 IoU training threshold
- **anchor_t:** 4.0 anchor-multiple threshold
- **# anchors:** 3 anchors per output layer (0 to ignore)
- **fl_gamma:** 0.0 focal loss gamma (efficientDet default gamma=1.5)

Data Augmentation:

- **hsv_h:** 0.015 image HSV-Hue augmentation (fraction)
- **hsv_s:** 0.7 image HSV-Saturation augmentation (fraction)
- **hsv_v:** 0.4 image HSV-Value augmentation (fraction)
- **degrees:** 0.0 image rotation (+/- deg)
- **translate:** 0.1 image translation (+/- fraction)
- **scale:** 0.9 image scale (+/- gain)
- **shear:** 0.0 image shear (+/- deg)
- **perspective:** 0.0 image perspective (+/- fraction), range 0-0.001
- **flipud:** 0.0 image flip up-down (probability)
- **fliplr:** 0.5 image flip left-right (probability)
- **mosaic:** 1.0 image mosaic (probability)
- **mixup:** 0.1 image mixup (probability)
- **copy_paste:** 0.1 segment copy-paste (probability)

GRAFICOS:

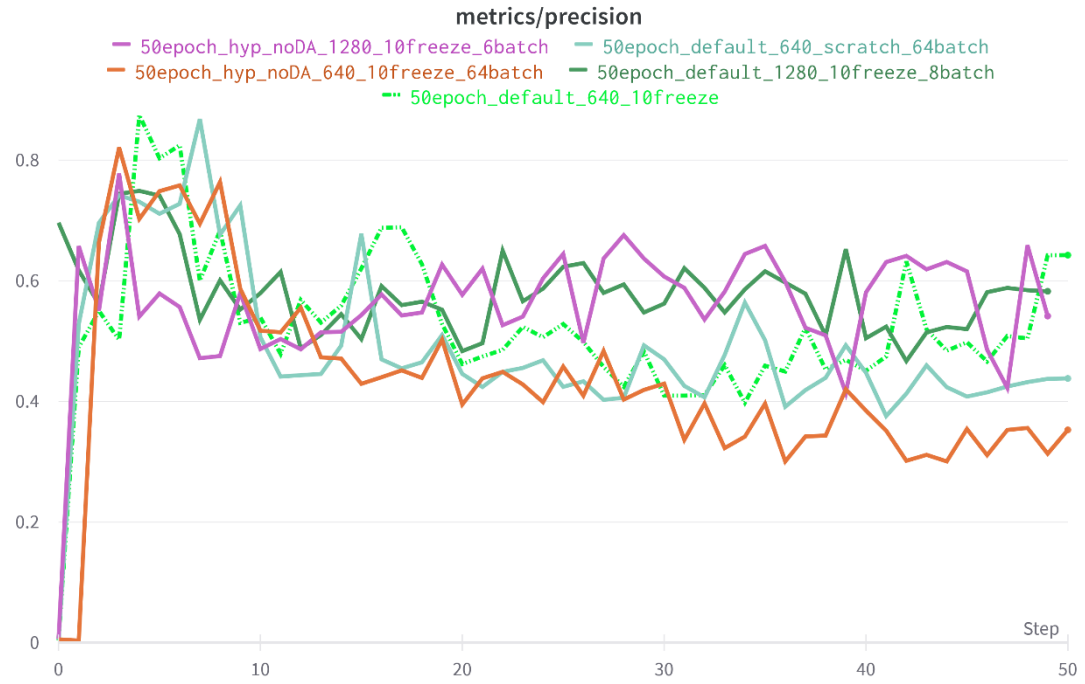


Figure 1: PRECISIÓN

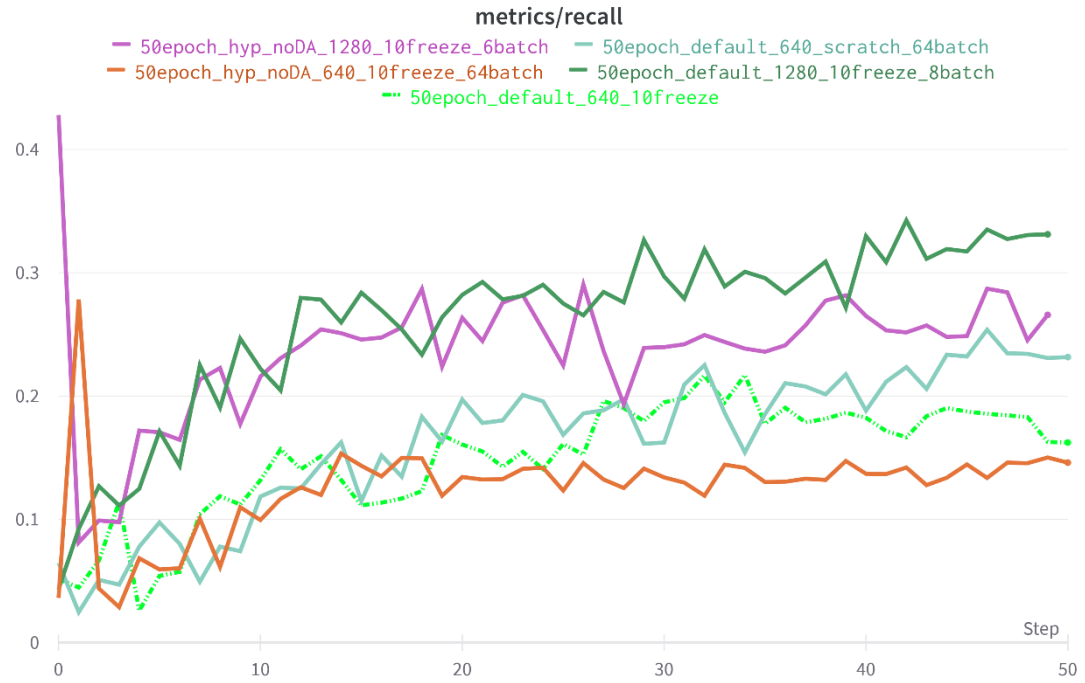


Figure 2: RECALL

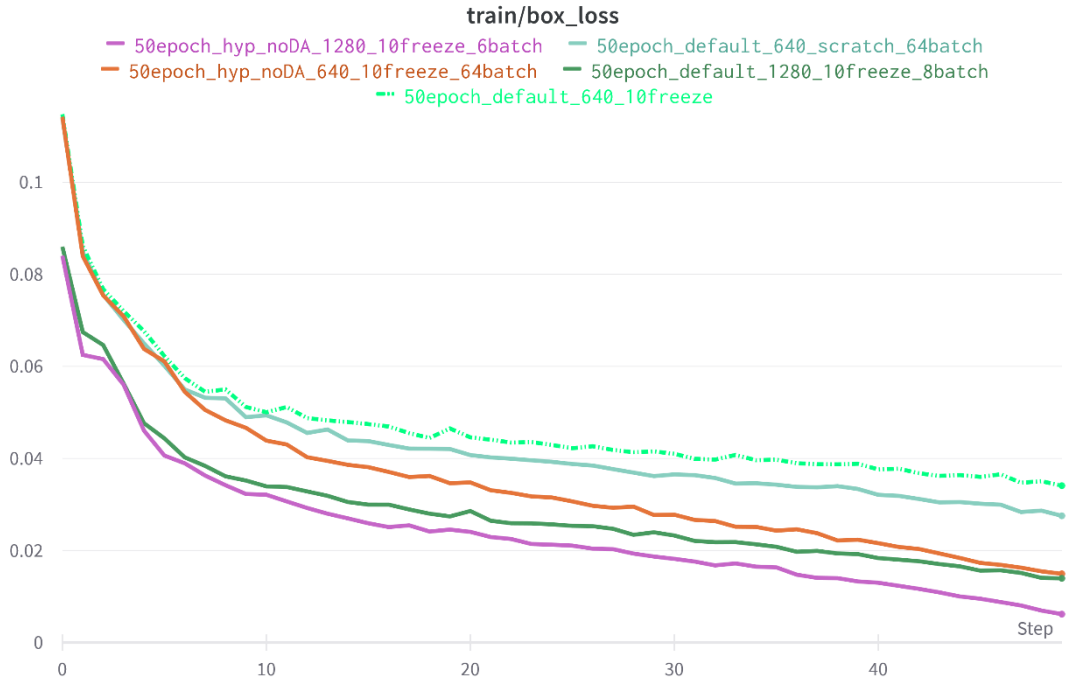


Figure 3: BOX LOSS SOBRE SET DE ENTRENAMIENTO

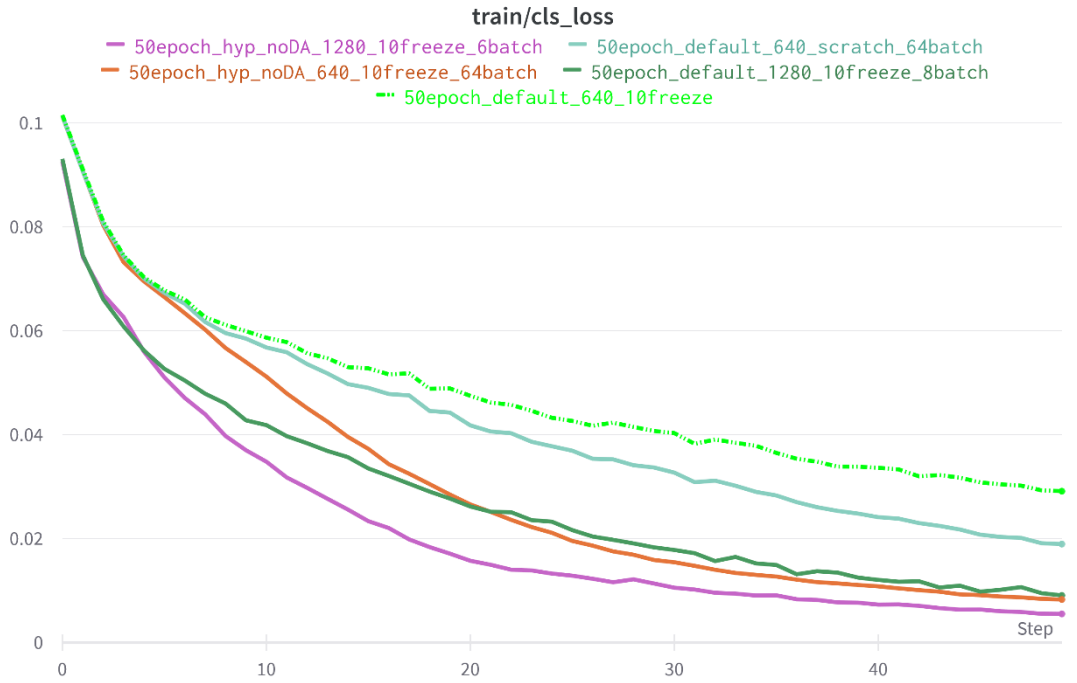


Figure 4: CLS LOSS SOBRE SET DE ENTRENAMIENTO

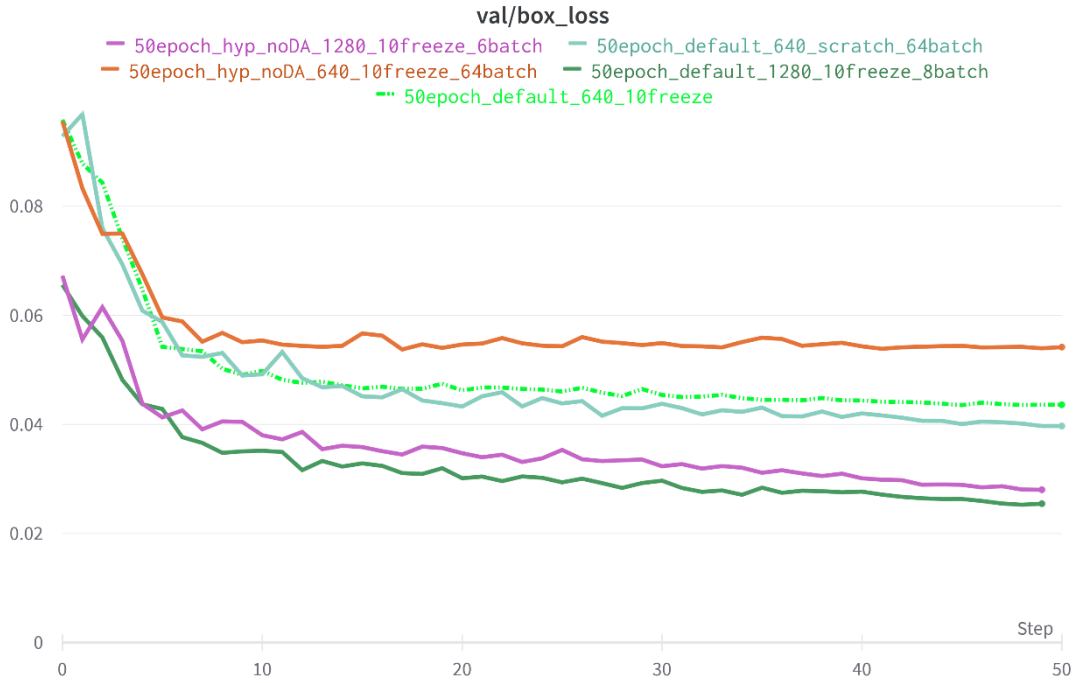
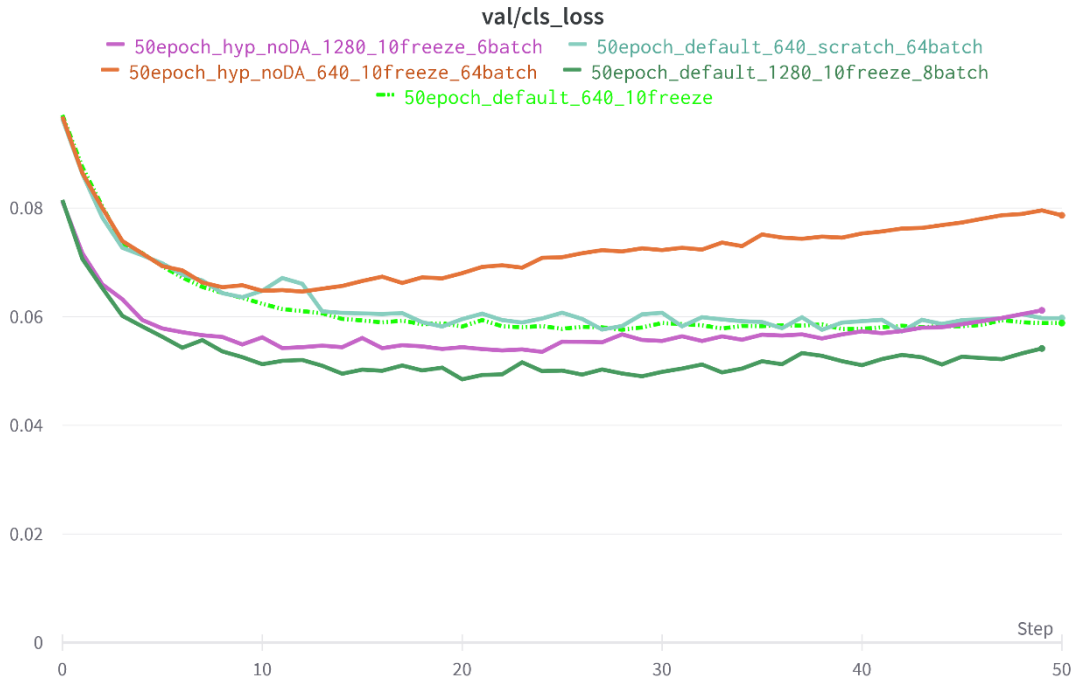


Figure 5: BOX LOSS SOBRE SET DE VALIDACIÓN



CONCLUSIONES:

- El modelo con mejor performance general es el **#5** debido al tamaño mayor de imágenes de entrada.

- Los modelos entrenados con Data Augmentation tuvieron mejor performance (**#5** vs. **#3** y **#4** vs. **#2**)

- El modelo entrenado sin pesos pre-entrenados, llamativamente, tiene una performance del orden de los análogos, pero utiliza mayor cantidad de recursos (GPU).

- Las configuraciones sin Data Augmentation tienen mayor tendencia a sobre-entrenar para la misma cantidad de épocas. Sobre todo en el modelo de imágenes de entrada más reducidas (640x640).

- En general, y como se observa en el ejemplo de imagen que se encuentra abajo, se logran detectar los objetos. Pero en caso de segmentación, se observa que no todas las etiquetas son correctas. Los objetos de las categorías más representativas, como las botellas plásticas transparentes, son los que se identifican con mayor facilidad.

- Como conclusión de lo mencionado, se podría mejorar el modelo haciendo las siguientes pruebas:

  aumentar data augmentation;

  aumentar tamaño de imágenes;

  aumentar cantidad de epochs;

  agregar imágenes de interés para hacer el dataset más significativo;

  eliminar las etiquetas de los objetos que no presentan interés y no son bien representados en el dataset.

  Se recomienda persistir con transfer learning, frisando backbone (primeras 10 capas) y además hacer pruebas para mayor cantidad de capas frizadas.

RESULTADOS DEL MODELO SOBRE IMÁGENES DEL SET DE VALIDACIÓN EN CONFIGURACIÓN “E”