Campus Monterrey

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 501)

Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning.

Aldo Daniel Villaseñor Fierro A01637907

Aplicación de modelos de deeplearnin en proyecto de equipo

Para la realización del reto presentado en la materia se hace uso de varios modelos que nos permiten procesar imágenes, una de las tareas que se deben de realizar en el proyecto es la detección de humanos para realizar un crop de la región en donde se encuentra un humano y pasar la imagen a mediapipe dado que este ultimo solo puede realizar análisis de pose para una persona en la imagen.

YOLOv8

YOLO (You Only Look Once) Se trata de un enfoque popular en el campo de la visión por computadora y el aprendizaje profundo (deep learning) utilizado para la detección de objetos en imágenes y videos. Fue desarrollado por Joseph Redmon y Santosh Divvala en la Universidad de Washington.

La característica distintiva de YOLO es su capacidad para detectar múltiples objetos en una sola pasada a través de una imagen o un cuadro de video, en lugar de requerir múltiples análisis de la imagen como lo hacen algunos otros enfoques de detección de objetos. YOLO logra esto mediante la división de la imagen en una cuadrícula y la asignación de cajas delimitadoras (bounding boxes) a regiones específicas de la cuadrícula para identificar objetos. Luego, el modelo de aprendizaje profundo predice las clases y las ubicaciones de los objetos en cada caja delimitadora.

YOLO es conocido por su velocidad y eficiencia. Puede ejecutarse en tiempo real en hardware no muy potente, lo que lo hace adecuado para aplicaciones en tiempo real como la detección de objetos en videos en tiempo real. Cuenta con la capacidad de detectar y reconocer hasta 80 objetos, entre ellos humanos.

Tranfer Learning

A pesar del gran desempeño que tiene en gran cantidad de circunstancias YOLO puede mejorar, en especial si el objetivo de su implementación está orientado a la detección de una sola clase, como es necesario para el proyecto. Este problema puede solucionarse en menor o mayor medida al utilizar técnicas de transfer learning para “especializar” a YOLO en la detección de un objeto concreto.

Tranfer learning es una técnica en el campo del aprendizaje profundo (deep learning) en la que un modelo previamente entrenado en una tarea específica se utiliza como punto de partida para entrenar un nuevo modelo en una tarea relacionada pero diferente. Esta técnica aprovecha el conocimiento adquirido por el modelo previamente entrenado y lo "transfiere" al nuevo modelo, lo que a menudo acelera y mejora el rendimiento del nuevo modelo.

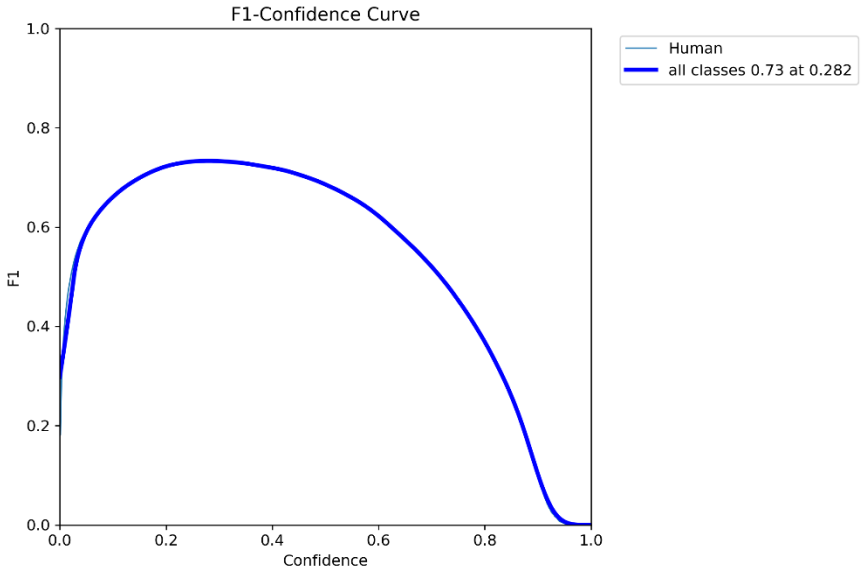
Reentrenamiento de YOLO

Existen varias maneras de implementar modelos YOLO, en este documento se presenta una implementación utilizando Python y el framework de ultralytics, este posee soporte para realizar entrenamiento de varios modelos de YOLO (YOLOv8 en este caso).

Para el reentrenamiento de YOLO se utilizo un dataset el cual contiene imágenes en donde aparecen humanos y sus respectivas bounding boxes en un archivo de texto (uno para cada imagen) conteniendo la ubicación en la imagen de dichas cajas. El dataset de divide en 2220 imágenes de entrenamiento y 1642 imágenes de validación. Además, se agregaron imágenes obtenidas en el salón de clases, sin embargo, por cuestione de tiempo no fue posible generar un set suficientemente grande para generar cambios significativos en los resultados

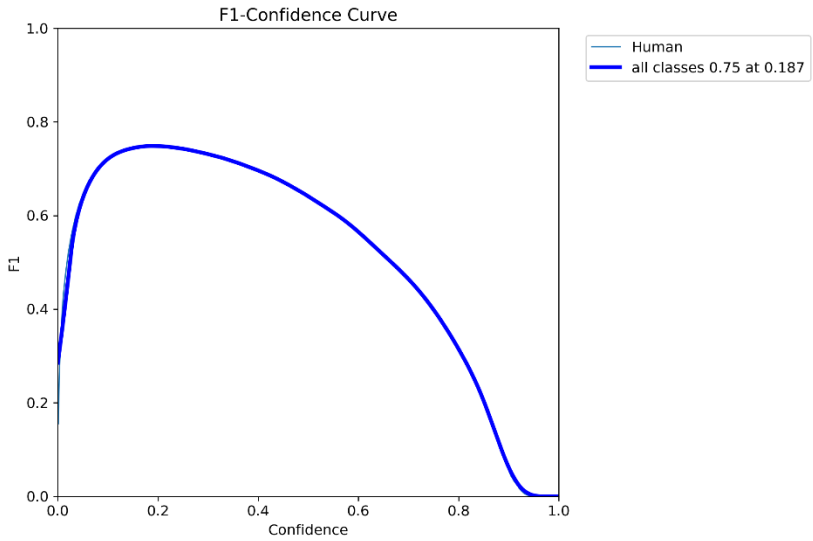
Primeramente, se realizó un entrenamiento usando 10 épocas, un batchsize de 16 y un tamaño de imagen de 640x640 (tamaño estándar para aplicaciones de YOLOv8) obteniendo los siguientes resultados.

box_loss	cls_loss	dfl_loss	Box(P	R	mAP50	mAP50-95)
0.861	0.59	0.9712	0.836	0.679	0.778	0.51



En una segunda corrida del entrenamiento se usaron 20 épocas, un batchsize de 16 y un tamaño de imagen de 640x640 (tamaño estándar para aplicaciones de YOLOv8) obteniendo los siguientes resultados.

box_loss	cls_loss	dfl_loss	Box(P	R	mAP50	mAP50-95)
0.7926	0.5348	0.9446	0.837	0.652	0.77	0.499



Además del número de epocas, batchsize y el tamaño de imagen ultralytics permite variar otro gran número de parámetros los cuales no se mencionan dado que se dejaron en el valor default, estos se agregan como anexo al final del reporte.

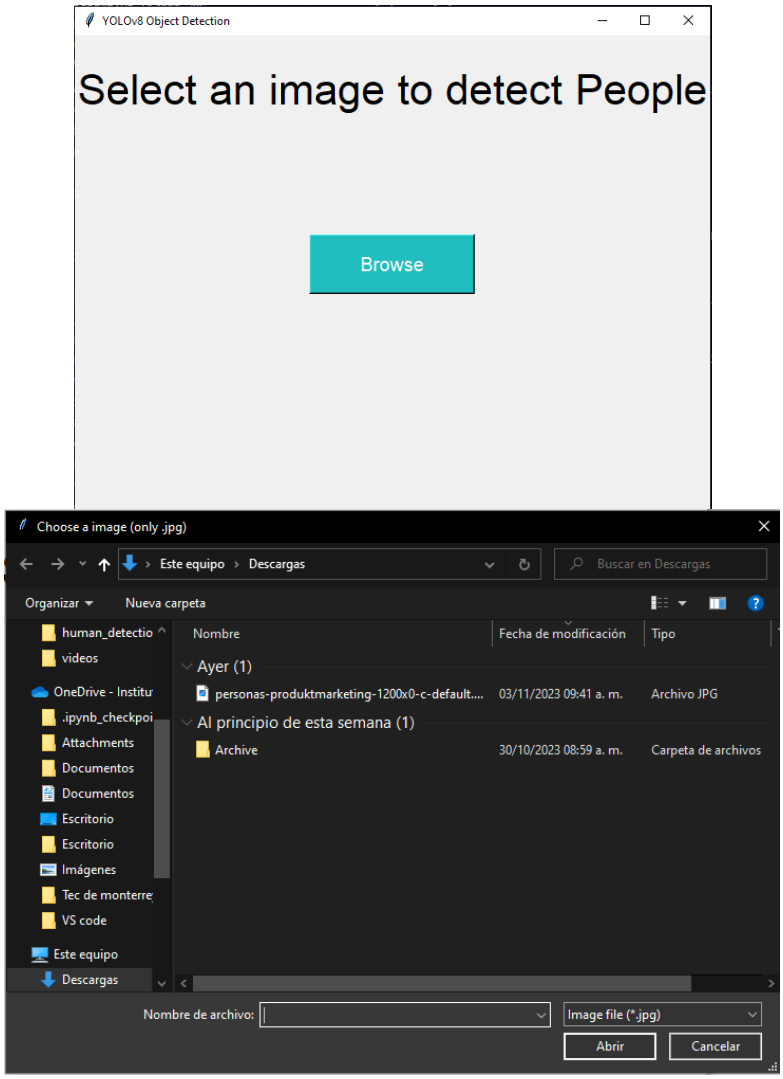
Análisis del resultado.

En general se obtuvieron mejores métricas para el modelo con 20 épocas. Es importante denotar que solo estamos realizando la detección de una clase por lo cual métricas como mAP y class loss no son de mucho interés, la principal métrica que se observa es box loss, la cual explica que tan bueno es el modelo para encontrar el centro de un objeto en la imagen. Al observar esta métrica se observa una gran mejoría entre el primer modelo y el segundo.

Implementación mediante app de Python.

Mediante el módulo TKinter nos es posible crear una aplicación con la cual podemos desplegar nuestro modelo para el análisis de una imagen.

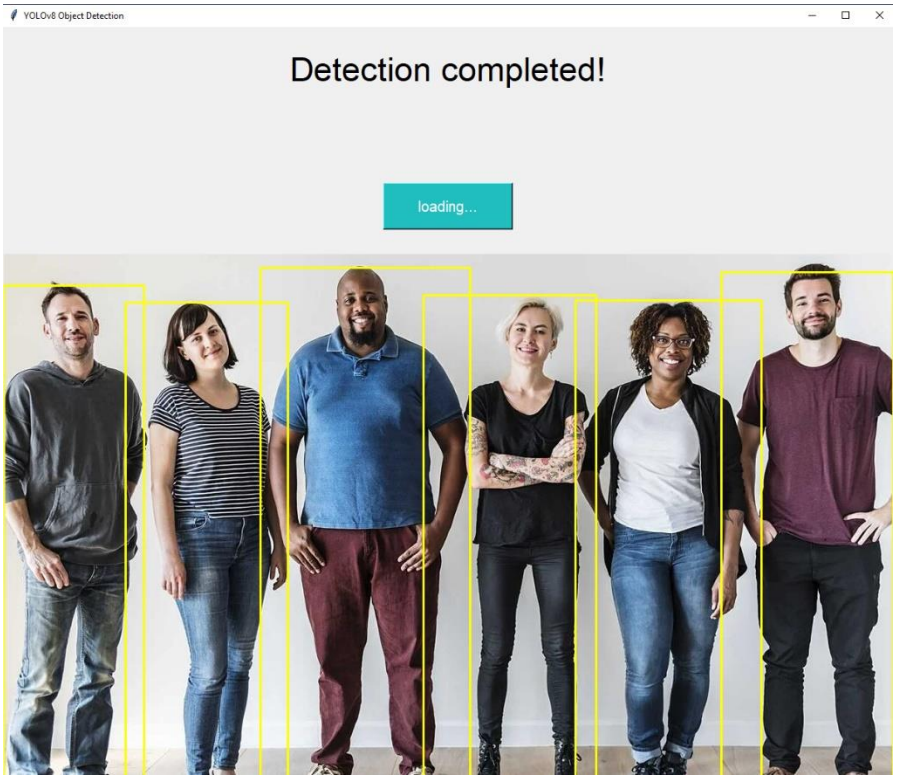
Primero se le pide al usuario que busque en su explorador de archivos la imagen que desee analizar.



Una vez seleccionada la imagen se muestra el resultado del proceso del modelo previamente entrenado en donde se logran observar las cajas encerrando a las personas.

Imagen original:

Imagen generada con la app:



Entrenamiento utilizando TensorFlow keras

Es posible realizar el entrenamiento mediante el la API de tensorflow para Keras. Eso es posible ya que ultralytics permite exportar cualquier modelo en diferentes formatos los cuales después de ciertos ajustos pueden ser precargados en un modelo secuancial de keras, sin embargo esto no nos permite manipular la arquitectura de la red, solo nos permitiría hacer el mismo proceso de entrenamiento ya mencionado pero utilizando tf.keras.

A pesar de lo ya mencionado es posible replicar la arquitectura de YOLO al observar el ouput de la función de Python que realiza el entrenamiento median ultralytics

```
results = model.train(
  data="data.yaml",
  imgsz=640,
  epochs=10,
  batch=8,
  workers=8)
```

la cual da como salida entre otras cosas.

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.block.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.block.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.block.C2f	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.block.SPPF	[512, 512, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	591360	ultralytics.nn.modules.block.C2f	[768, 256, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.block.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2116435	ultralytics.nn.modules.head.Detect	[1, [128, 256, 512]]

De aquí podemos inferir la arquitectura al revisar las funciones de ultralytics que definen la red. Estas se encuentran en la documentación de ultralytics en https://github.com/ultralytics/ultralytics.git.

Cada una de las funciones que se mencionan pueden ser replicables en tensorflow, por ejemplo la función ultralytics.nn.modules.conv.Conv la cual es la más utilizada en la arquitectura puede ser replicada en tensorflow. Sería

interesante replicar la arquitectura y entrenar al modelo únicamente con imágenes en donde se haga detección de humanos y compara los resultados de este modelo con los de los previamente obtenidos realizando transferlearning con YOLO.

```python
class Conv(nn.Module):
    """Standard convolution with args(ch_in, ch_out, kernel, stride, padding, groups, dilation, activation)."""
    default_act = nn.SiLU()  # default activation

    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, d=1, act=True):
        """Initialize Conv layer with given arguments including activation."""
        super().__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p, d), groups=g, dilation=d, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = self.default_act if act is True else act if isinstance(act, nn.Module) else nn.Identity()

    def forward(self, x):
        """Apply convolution, batch normalization and activation to input tensor."""
        return self.act(self.bn(self.conv(x)))

    def forward_fuse(self, x):
        """Perform transposed convolution of 2D data."""
        return self.act(self.conv(x))
```

Anexos

Liga a drive con los archivos .ipynb:

https://drive.google.com/drive/folders/1wFMrFEzSHqy317zEtOMB0KDQxI-zdXY8?usp=drive_link

Parámetros default para el modelo YOLOv8:

```
task=detect, mode=train, model=yolov8s_train1.pt, data=data.yaml, epochs=10,
patience=50, batch=8, imgsz=640, save=True, save_period=-1, cache=False,
device=None, workers=8, project=None, name=train, exist_ok=False,
pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True,
single_cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False,
amp=True, fraction=1.0, profile=False, freeze=None, overlap_mask=True,
mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False,
save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False,
plots=True, source=None, show=False, save_txt=False, save_conf=False,
save_crop=False, show_labels=True, show_conf=True, vid_stride=1,
stream_buffer=False, line_width=None, visualize=False, augment=False,
agnostic_nms=False, classes=None, retina_masks=False, boxes=True,
format=torchscript, keras=False, optimize=False, int8=False, dynamic=False,
simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01,
momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8,
warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0,
label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0,
translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0,
fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0, cfg=None,
tracker=botsort.yaml, save_dir=runs\detect\train
```