Capítulo 4

Implementaciones

4.1. Implementación de los modelos

Para la selección de modelos de detección de objetos aplicados a la identificación de criaderos de mosquitos, se decidió considerar las versiones YOLOv4, YOLOv7, YOLOv8 y YOLOv12. No obstante, dado que cada una de estas versiones cuenta con múltiples variantes (por ejemplo, Tiny, S, M, L, X), se optó por realizar una comparación justa seleccionando las versiones cuyos números de parámetros fueran más similares, tomando como referencia el modelo YOLOv4 (véase la Tabla 4.1).

Por esta razón, se eligieron específicamente los modelos YOLOv4, YOLOv7X, YOLOv8X y YOLOv12X. Aunque el número de parámetros no es exactamente el mismo entre ellos, se considera que presentan una magnitud comparable que permite una evaluación justa de su rendimiento relativo. Para comparar los modelos, los conjuntos de datos fueron divididos en 60 % para el entrenamiento, 20 % para validación y 20 % para pruebas.

Adicionalmente, se incluyeron en el estudio las versiones Tiny de cada modelo, con el fin de disponer de una comparativa general que abarque tanto modelos de alto rendimiento como versiones ligeras dentro de cada arquitectura YOLO.

Tipo	YOLOv4	YOLOv7	YOLOv8	YOLOv12
Tiny	6,053,651	6,230,925	5,070,114	4,358,120
Normal	63,959,226	37,620,125	–	–
N	–	W6 - 70,426,236	3,157,200	2,603,056
S	–	–	11,368,187	9,285,632
M	–	E6 - 97,246,652	26,204,299	20,201,216
L	–	–	44,093,211	26,454,880
X	–	71,344,389	68,731,371	59,216,928

Tabla 4.1: Número de parámetros por versión y tipo de modelo YOLO. Las celdas con guiones indican que el tipo de modelo no aplica para esa versión de YOLO. La versión YOLOv7 utiliza una nomenclatura distinta para sus variantes (por ejemplo, W6, E6, etc.). Las celdas resaltadas corresponden a los modelos seleccionados para la comparativa principal. Aunque existen otras variantes de modelos en cada versión de YOLO, únicamente se consideraron las aquí mostradas para los fines de este estudio.

A continuación se muestran la implementación de los modelos:

4.1.1. YOLOv4

Existen diversas implementaciones de YOLOv4 disponibles en GitHub, sin embargo el que se utilizó, es el de la implementación en C y CUDA por Alexey Bochkovskiy, toda la información mencionada acerca de la implementación de este modelo se puede encontrar el repositorio [30].

Requisitos para Windows, Linux y macOS

Para poder utilizar el modelo YOLOv4 de manera adecuada, es indispensable contar previamente con una serie de requisitos de software. En la Tabla 4.2, se detallan los componentes esenciales que deben estar presentes en el sistema para asegurar la correcta compilación y ejecución del modelo.

Requisito	Descripción
CMake $\geq$ 3.18	Herramienta de automatización de compilación multiplataforma, utilizada para gestionar el proceso de construcción del software mediante archivos de configuración legibles.
CUDA $\geq$ 10.2	Plataforma de computación paralela desarrollada por NVIDIA, que permite el uso de la GPU para acelerar cálculos intensivos. Es esencial para la ejecución eficiente de redes neuronales profundas.
OpenCV $\geq$ 2.4	Biblioteca de visión por computadora que proporciona herramientas para el procesamiento de imágenes, acceso a cámaras, y operaciones relacionadas.
cuDNN $\geq$ 8.0.2	Biblioteca de primitivas para redes neuronales profundas optimizada para GPUs NVIDIA. Mejora el rendimiento del entrenamiento e inferencia de modelos.
GPU con CC $\geq$ 3.0	Se requiere una tarjeta gráfica compatible con CUDA Compute Capability (CC) versión 3.0 o superior, lo cual garantiza la compatibilidad con las operaciones paralelas que realiza YOLOv4.

Tabla 4.2: Requisitos de software necesarios para la implementación de YOLOv4.

Una vez satisfechos estos requisitos, es posible proceder con la instalación y uso del modelo YOLOv4. Para facilitar la gestión de dependencias y evitar conflictos con otros entornos del sistema, se recomienda realizar la instalación dentro de un entorno virtual o emplear una solución basada en contenedores, como Docker. Esta estrategia permite mantener un entorno controlado y reproducible.

Cabe señalar que en el repositorio oficial de YOLOv4 también se detallan los requisitos específicos para su implementación en diferentes frameworks, lo cual puede ser útil dependiendo del entorno de desarrollo preferido por el usuario.

Compilación en Linux

Dado que el presente trabajo se desarrolla en un sistema operativo basado en Linux, se opta por realizar la compilación del modelo en dicho entorno. Para ello, se debe acceder a la terminal y ejecutar los siguientes comandos:

```
1  git clone https://github.com/AlexeyAB/darknet
2  cd darknet
3  make
4
```

Código 4.1: basicstyle=, **Compilación en Linux de YOLOv4**

Para lograr una configuración óptima del entorno, es recomendable realizar ciertas modificaciones al archivo de configuración denominado `Makefile`, ubicado dentro de la carpeta `darknet` antes de ejecutar `make`. Como ocurre en muchos proyectos de código abierto, el `Makefile` contiene las instrucciones necesarias para compilar el código fuente y generar el ejecutable `darknet`. El comando `make` interpreta dichas instrucciones y ejecuta el proceso de compilación.

En la Tabla 4.3 se describen las principales opciones disponibles en el `Makefile` que pueden ser modificadas de acuerdo con las capacidades del sistema y los objetivos del proyecto.

Parámetro	Descripción
`GPU=1`	Activa el uso de la GPU mediante CUDA. Es altamente recomendable habilitar esta opción si se dispone de una tarjeta gráfica compatible, ya que mejora significativamente el rendimiento.
`CUDNN=1`	Habilita la biblioteca cuDNN de NVIDIA, lo que permite una aceleración adicional en el entrenamiento e inferencia de redes neuronales.
`CUDNN_HALF=1`	Permite el uso de precisión de 16 bits para aprovechar los Tensor Cores presentes en tarjetas gráficas modernas como las NVIDIA RTX o Tesla V100.
`OPENCV=1`	Activa la biblioteca OpenCV, útil para procesar imágenes, leer archivos de video, visualizar resultados, entre otras funcionalidades de visión por computadora.
`DEBUG=1`	Compila el proyecto con información adicional para depuración, lo cual es útil durante la etapa de desarrollo.
`OPENMP=1`	Habilita el uso de OpenMP, permitiendo paralelismo mediante múltiples núcleos de CPU. Esta opción es recomendable cuando no se cuenta con GPU.
`LIBSO=1`	Genera una biblioteca compartida `darknet.so`, en lugar de un único ejecutable. Esto resulta útil para integrar YOLOv4 en proyectos desarrollados en C/C++ o Python.
`ZED_CAMERA=1`	Activa el soporte para cámaras ZED, utilizadas comúnmente en aplicaciones de visión estéreo y robótica.
`ARCH=`	Define la arquitectura de la tarjeta gráfica utilizada. Por defecto se incluye la opción `-gencode arch=compute_50,code=sm_50`, la cual puede ser modificada de acuerdo con el modelo específico de GPU.

Tabla 4.3: Opciones de compilación disponibles en el archivo `Makefile` de YOLOv4.

Por defecto, todas estas opciones vienen deshabilitadas (con valor igual a `0`). Por lo tanto, es necesario activarlas (cambiando el valor a `1`) en función de los recursos disponibles y el propósito de uso. Para las tareas de detección de objetos que se abordan en este proyecto, se habilitaron los siguientes opciones: `GPU`, `CUDNN`, `CUDNN_HALF`, `OPENCV` y `OPENMP`. Asimismo, se debe ajustar el valor de `ARCH=` conforme a la

arquitectura de la GPU. En este caso, se empleó:

```
ARCH=-gencode arch=compute_75,code=[sm_75,compute_75]
```

Véase la Sección 4.2 donde se detallan las características de cómputo utilizado.

El repositorio oficial también proporciona instrucciones detalladas sobre cómo llevar a cabo la compilación en sistemas operativos Windows, así como otras configuraciones avanzadas que pueden ser de interés para desarrolladores con necesidades específicas.

Entrenamiento con datos personalizados

Muchas veces es deseable entrenar un modelo YOLO con bases de datos personalizadas para la detección de objetos, como es nuestro caso con los criaderos de mosquitos. Una vez que se cuentan con los requerimientos de software y habiendo hecho la compilación de acuerdo a la configuración hecha en `Makefile`, se puede entrenar un modelo YOLOv4 siguiendo los siguientes pasos:

1. Colocar las imágenes `jpg` previamente etiquetadas en algún directorio.

2. Debemos generar tres archivos `txt` una para el conjunto de entrenamiento, validación y prueba, donde contengan las rutas de las imagenes. Dentro del repositorio nos dirigimos a una carpeta llamada `/data`, en esta carpeta pondremos los archivos en formato `txt`.

3. Es necesario descargar un archivo de configuración para entrenar y los pesos correspondientes a dicha configuración en el repositorio oficial ya mencionado, por ejemplo `yolov4.cfg` o `yolov4-custom.cfg`, con los pesos `yolov4.conv.137.weights`. A continuación se muestran algunas configuraciones y sus respectivos pesos que se pueden considerar:

   - Para `yolov4.cfg`, `yolov4-custom.cfg`: `yolov4.conv.137.weights`.
   - Para `yolov4-tiny.cfg`, `yolov4-tiny-3l.cfg`, `yolov4-tiny-custom.cfg`: `yolov4-tiny.conv.29.weights`.

   El seleccionado por nosotros fue `yolov4-custom.cfg` con los pesos `yolov4.conv.137.weights`.

4. Una vez descargados la configuración y los pesos, se hace una copia de la configuración para realizar una configuración personalizada. Una vez hecho esto, se debe configurar lo siguiente dentro del archivo (véase las Tablas 4.4, 4.5):

   - Cambiar la línea de lote deseado, en nuestro caso `batch=8`.
   - Cambiar subdivisiones, en nuestro caso `subdivisions=8`.
   - Cambiar `max_batches` de acuerdo a la siguiente formula `número de clases` $\times$ `2000`(pero no menos que el número de imágenes de entrenamiento y no menos que 6000), para nuestro caso que consideramos 5 categorías `max_batches=10000`.
   - Cambiar la línea `steps` considerando el 80 % y 90 % de `max_batches`, para nuestro caso en el que `max_batches=10000` lo configuramos a `steps=8000,9000`.
   - Configurar el tamaño de la red `width` y `height`, nosotros usamos `width=1024` y `height=1024`.
   - Cambiar la configuración de caja de anclajes `anchors`, por defecto vienen de acuerdo a la base de datos COCO, los anchors se calculan con el siguiente comando:

     ```
     ./darknet detector calc_anchors obj.data -num_of_clusters 9
     -width 1024 -height 1024 -show
     ```

     Con este comando se calculan y se imprimen los anchors. Para `yolov4-custom.cfg` se deben modificar en las líneas 969, 1057 y 1145. Para nuestro conjunto de datos de criaderos de mosquitos los anchors son: `anchors = 13,21, 18,29, 31,40, 35,52, 45,59, 62,100, 90,82, 94,135, 142,188`.
   - Cambiar el número de clases, `classes=5` (en cada capa `[yolo]`). Para `yolov4-custom.cfg` se encuentran en las líneas 970, 1058 y 1146.
   - Cambiar los filtros de acuerdo a la siguiente formula `filters=(clases+5)*3` en las capas antes de `[yolo]`. Para `yolov4-custom.cfg` se encuentran en las líneas 963, 1051 y 1139, para nuestro caso `filters=30`. Si se usa `Gaussian_yolo`, usar `filters=(clases+9)*3`.

5. Una vez modificado el archivo de configuración. Realizamos una copia de esta configuración que funcionará como configuración para la validación del modelo, en esta configuración únicamente se modifica el valor de batches y subdivisiones a los deseados, en nuestro caso `batch=8` y `subdivisions=8`.

6. Crear un archivo `obj.names` en el directorio con los nombres de las categorías (una por línea). Por ejemplo:

```
charco
cubeta
llanta
tinaco
maceta

```

Código 4.2: basicstyle=,**Categorias en el archivo obj.names**

7. Crear un archivo `obj.data` con el siguiente contenido:

```
classes= 5
train  = data/train_60_criaderos.txt
valid  = data/val_20_criaderos.txt
names = data/obj.names
backup = backup/

```

Código 4.3: Archivo obj.data. Se especifica el número de clases, las rutas del conjunto de entrenamiento y validación, el nombre de las categorías y la ruta donde se guardarán los pesos generados en el entrenamiento.

8. Realizar el entrenamiento del modelo ejecutando el siguiente comando (para Linux) en el repositorio donde se encuentra el darknet:

```
time ./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137
```

El comando `time` es para registrar la duración del entrenamiento. Básicamente se especifica la ruta del archivo `obj.data`, la ruta de la configuración `yolo-obj.cfg` y la ruta de los pesos del modelo `yolov4.conv.137`.

9. Una vez finalizado el entrenamiento, se obtiene el archivo `yolo-obj_final.weights` en la ruta especificada en el archivo `obj.data`. Los pesos se guardan de la siguiente manera:

   - `yolo-obj_last.weights` se guarda cada 100 iteraciones.
   - `yolo-obj_xxxx.weights` se guarda cada 1000 iteraciones.
   - `yolo-obj_best.weights` se guarda de acuerpo al mejor valor de mAP obtenido.

10. Para realizar la prueba del modelo con los datos de test, es necesario modificar en el archivo `obj.data` la ruta de validación por la de nuestros datos de test, puesto que esta versión de YOLO no admite en la configuración la ruta de un conjunto de datos test. Una vez hecho esto, la prueba del modelo se realiza con el mismo comando de validación:

    ```
    ./darknet detector test data/obj.data yolo-obj.cfg backup/yolo-obj\_best.weights
    ```

    Se imprime en pantalla los resultados después del test.

Consideraciones adicionales:

- Si se cambian `width=` o `height=` en el archivo .cfg, los nuevos valores deben ser divisibles por 32.
- Se recomienda siempre cambiar la configuración de los `anchors` al de los datos personalizados.
- Después del entrenamiento, puede usar el siguiente comando para hacer una detección:

  ```
  ./darknet detector test data/obj.data yolo-obj.cfg yolo-obj_best.weights
  ```

- Si durante el entrenamiento aparece un error `Out of memory`, puede ser necesario aumentar el valor de `subdivisions` (por ejemplo, pasar de 16 a 32 o 64).

¿Cuándo dejar de entrenar?

En general se consideran suficientes 2000 iteraciones por categoría, pero no menos del número de imágenes de entrenamiento ni menos de 6000 iteraciones en total.

Cuando se realiza el entrenamiento, se genera un gráfico `chart.png` que indica la pérdida promedio (`avg loss`) y la evolución del `mAP`, que se actualiza cada determinado número de iteraciones. Estos son nuestros indicadores clave y debemos detener el entrenamiento de acuerdo con los siguientes criterios:

- Si se observa que el valor del `mAP` no mejora durante 1000 o 2000 iteraciones, entonces se debe cortar el entrenamiento.
- Cuando la pérdida promedio se aplana, es señal de que el modelo ya no está aprendiendo.

Líneas de comando

YOLOv4 puede ejecutarse directamente desde la terminal mediante una serie de comandos, los cuales son distintos dependiendo de la tarea que se desea realizar. Los comandos utilizados para ejemplificar las tareas realizadas con dichos modelos están basados en los repositorios oficiales, por lo que en su mayoría emplean la base de datos COCO. Esta base de datos ha sido utilizada de manera estándar para evaluar el rendimiento de todos los modelos YOLO, desde sus primeras versiones hasta las más recientes. A continuación, se presentan algunos de los comandos más utilizados:

- **./darknet detector train data/coco.data cfg/yolov4.cfg yolov4.weights**: Realiza el entrenamiento del modelo.
- **./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -thresh 0.25**: Ejecuta la detección de objetos en una única imagen, con un umbral de confianza de 0.25 (las detecciones con una probabilidad menor a 0.25 se descartan).
- **./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output test.mp4**: Realiza la detección de objetos en un video en formato `mp4` y muestra los resultados en tiempo real.

- **./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -c 0**: Utiliza la cámara web local para ejecutar detección en tiempo real.
- **./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -thresh 0.25 -dont_show -save_labels <data/new_train.txt**: Ejecuta la detección de objetos sobre imágenes y guarda las coordenadas de las cajas en formato YOLO en archivos `txt` individuales por imagen.
- **./darknet detector calc_anchors data/obj.data -num_of_clusters 9 -width 416 -height 416**: Calcula los *anchors* óptimos al entrenar un modelo con un conjunto de datos personalizado. Es importante especificar la resolución del modelo de entrada.
- **./darknet detector map data/obj.data yolo-obj.cfg backup/yolo-obj_7000.weights -iou_thresh 0.75**: Evalúa la precisión del modelo utilizando un umbral de intersección sobre unión (IoU) de al menos 0.75. Si no se especifica este valor, el umbral por defecto es 0.5.

Existen muchos otros comandos disponibles. Para una referencia más detallada, se recomienda consultar la sección correspondiente a líneas de comando en el repositorio oficial de YOLOv4 [30].

Configuración de hiperparámetros

La configuración se realiza en el archivo `yolov4-custom.cfg`. En nuestro caso utilizamos la siguiente configuración:

Hiperparámetros	Valor	Descripción	Modificado
batch	64	Tamaño del lote	* 8
subdivisions	16	Subdivisiones del lote	* 8
width	608	Ancho de la imagen de entrada	* 1024
height	608	Alto de la imagen de entrada	* 1024
channels	3	Número de canales (RGB)	
momentum	0.949	Momento del optimizador	
decay	0.0005	Decaimiento del peso (weight decay)	
angle	0	Rotación aleatoria de imagen	
saturation	1.5	Aumento de saturación	
exposure	1.5	Aumento de exposición	
hue	0.1	Aumento de tono (hue)	
learning_rate	0.001	Tasa de aprendizaje inicial	
burn_in	1000	Iteraciones de calentamiento (burn-in)	
max_batches	500500	Número máximo de iteraciones	* 10000
policy	steps	Política de actualización del LR	
steps	400000,450000	Iteraciones donde se reduce el LR	* 8000, 9000
scales	0.1, 0.1	Escalas para el LR en los pasos definidos	
mosaic	1	Activar data augmentation con mosaico	

Tabla 4.4: Fragmento de la configuración de hiperparámetros para YOLOv4. La configuración muestra los valores por defult y los hiperparámetros marcados con asterisco (*) indican los que fueron modificados.

Para el caso de los filtros y las clases.

Hiperparámetros	Valor	Descripción	Modificado
		[convolutional]	
size	1	Tamaño del filtro	
stride	1	Tamaño del paso (stride)	
pad	1	Padding activado (1 = sí)	
filters	255	Número de filtros ($3 \times$ [clases + 5])	* 30
activation	linear	Función de activación	
		[yolo]	
mask	0,1,2	Índices de anclas para esta capa	
anchors	12,16,...,459,401	Anclas predeterminadas (en pares ancho, alto)	* 13,21,...,142,188
classes	80	Número de clases del modelo	* 5
num	9	Número total de anclas	
jitter	0.3	Aleatorización de posición de objetos	
ignore_thresh	0.7	Umbral IoU para ignorar detecciones	
truth_thresh	1	Umbral de verdad para positivos	
scale_x_y	1.2	Escala del centro del bounding box	
iou_thresh	0.213	Umbral de IoU para pérdida	
cls_normalizer	1.0	Factor de normalización de pérdida de clase	
iou_normalizer	0.07	Factor de normalización de pérdida de IoU	
iou_loss	ciou	Tipo de pérdida de IoU	
nms_kind	greedynms	Tipo de supresión no máxima (NMS)	
beta_nms	0.6	Parámetro beta para NMS si aplica	
max_delta	5	Máxima variación permitida	

Tabla 4.5: Fragmento de la configuración para una capa [convolutional] y [yolo] para YOLOv4. La configuración muestra los valores por defult y los hiperparámetros marcados con asterisco (*) indican los que fueron modificados. Esta modificación se realiza en las tres capas correspondientes del archivo.

4.1.2. YOLOv7

La implementación oficial de YOLOv7 fue realizada en pytorch por Chien-Yao Wang, Alexey Bochkovskiy (quien también estuvo involucrado en YOLOv4), y Hong-Yuan Mark Liao, y el repositorio fue publicado por Wong Kin Yiu, un colaborador habitual en la evolución de modelos YOLO. Toda la información mencionada acerca de la implementación de este modelo pueden encontrarse en dicho repositorio [32].

Requisitos para Windows, Linux y macOS

Para poder hacer uso del modelo YOLOv7 primeramente es necesario contar con los requerimentos mostrados en la tabla 4.6.

Requisitos	Descripción
`matplotlib`≥`3.2.2`	Visualización de gráficos y resultados.
`numpy`≥`1.18.5, <1.24.0`	Cálculo numérico eficiente.
`opencv-python`≥`4.1.1`	Procesamiento de imágenes y video.
`Pillow`≥`7.1.2`	Manejo de imágenes.
`PyYAML`≥`5.3.1`	Lectura de archivos `.yaml`.
`requests`≥`2.23.0`	Descarga de recursos desde la web.
`scipy`≥`1.4.1`	Funciones científicas y cálculos.
`torch`≥`1.7.0, ≠1.12.0`	Framework base PyTorch
`torchvision`≥`0.8.1, ≠0.13.0`	Modelos adicionales de PyTorch.
`tqdm`≥`4.41.0`	Barras de progreso en consola.
`protobuf`<`4.21.3`	Requerido por ONNX y TensorFlow.
`tensorboard`≥`2.4.1`	Visualización de métricas de entrenamiento.
`pandas`≥`1.1.4`	Manipulación de datos tabulares.
`seaborn`≥`0.11.0`	Gráficos estadísticos.
`ipython`	Consola interactiva y notebooks.
`psutil`	Información del sistema (CPU, memoria, etc.)
`thop`	Cálculo de FLOPs del modelo.

Tabla 4.6: Requerimientos de dependencias para YOLOv7.

Instalación en Linux

La instalación recomendada en el repositorio es usando docker como se muestra a continuación:

```
# crear un contenedor de docker
nvidia-docker run --name yolov7 -it -v your_coco_path/:/coco/ -v your_code_path/:/yolov7 --shm-size=64g nvcr.io/nvidia/pytorch:21.08-py3

```

```
# apt install para instalar los paquetes requeridos
apt update
apt install -y zip htop screen libgl1-mesa-glx

# pip install para instalar las bibliotecas requeridas
pip install seaborn thop

# ir a la carpeta
cd /yolov7

```

Código 4.4: Instalación de YOLOv7 usando docker.

Dado la facilidad de usar un nuevo entorno, obtamos por esta opción para realizar la instalación creando un nuevo entorno usando anaconda y clonando dicho repositorio, para posteriormente instalar todos los requerimientos necesarios.

```
conda create -n yolov7 python=3.8 -y
conda activate yolov7
git clone https://github.com/WongKinYiu/yolov7
cd yolov7
pip install -r requirements.txt

```

Código 4.5: Instalación de YOLOv7 usando un nuevo entorno.

Una vez hecho toda la instalación podemos hacer uso del modelo YOLOv7.

Entrenamiento con datos personalizados

La configuración y el entrenamiento resulta ser más sencillo que YOLOv4, para realizar el entrenamiento una vez que se cuenta con los requerimientos y se haya copiado dicho repositorio, el entrenamiento se sigue como a continuación se indica:

1. Colocar las imágenes `jpg` previamente etiquetadas en algún directorio.

2. Dentro del repositorio nos dirigimos a una carpeta llamada `/data`, en esta carpeta pondremos los archivos en formato `txt` que indican las rutas de los datos de entrenamiento, validación y prueba, en nuestro caso, las mismas utilizados en YOLOv4.

3. Descargar los pesos preentrenados del tipo de modelo a utilizar, en este caso el seleccionado por nosotros fue el `yolov7x_training.pt`, dado la cercanía del número de parámetros con respecto al modelo de referencia YOLOv4.

4. No es necesario realizar la descarga de un archivo de configuración del modelo, ya que estás ya vienen incluidas en el repositorio. Para configurar este archivo basta con dirigirnos a `cfg/training/`, en esta carpeta se encuentra la configuración de todos los modelos YOLOv7, seleccionamos el de nuestro interés, en este caso para nosotros `yolov7x.yaml`.

```
# parameters
nc: 5  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple

# anchors
anchors:
- [13,21,  18,29,  31,40]  # P3/8
- [35,52,  45,59,  62,100]  # P4/16
- [90,82,  94,135,  142,188]  # P5/32

# yolov7 backbone
backbone:
# [from, number, module, args]
[[-1, 1, Conv, [40, 3, 1]],  # 0

[-1, 1, Conv, [80, 3, 2]],  # 1-P1/2
[-1, 1, Conv, [80, 3, 1]],
.
.
.

```

Código 4.6: Archivo de configuración del modelo YOLOv7x.

En estas configuraciones viene la arquitectura de la red, que es posible modificar a nuestro interés, para ser fieles a la arquitectura de la red original, no se modificó ninguna capa. Los valores que se modificaron y que son de nuestro interés es el número de categorías `nc` y los `anchors`, por defecto el número de categorías es 80 (dataset COCO), lo cambiamos a 5, y los `anchors` que se sustituyeron fueron los que se calcularon en YOLOv4, para no tener diferencias en anchors dado que se trata

del mismo dataset.

5. Una vez establecida la configuración es importante modificar el archivo de las rutas de nuestros datos, para eso podemos dirigirnos a `data/` y copiamos el archivo `coco.yaml`, lo modificamos con el nombre deseado y dentro de este archivo modificamos la ruta donde se encuentran nuestros datos de entrenamiento, validación y prueba, el número de categorías e indicamos las categorías que se necesitan, como se muestra a continuación.

```
# rutas
train: ./data/train_60_criaderos.txt
val: ./data/val_20_criaderos.txt
test: ./data/test_20_criaderos.txt

# numero de clases
nc: 5

# nombre de clases
names: ['charco', 'cubeta', 'tinaco', 'cubeta', 'llanta' ]

```

Código 4.7: Archivo de configuración YAML.

6. Como última configuración, nos dirigimos a configurar los hiperparámetros del modelo, para esto nos dirigimos `data/hyp.scratch.custom.yaml`, realizamos una copia de este archivo y podemos modificar todos los valores deseados. En nuestro caso dejamos los valores por defecto.

7. Una vez hecho las respectivas configuraciones, pasamos a entrenar al modelo, para esto nos dirigimos a la carpeta donde se encuentra el repositorio y ejecutamos el siguiente comando de acuerdo a la configuración realizada:

```
python train.py --batch-size 8 --data data/criaderos_mosquitos.yaml --img 1024 1024 --cfg cfg/training/yolov7x.yaml --weights 'yolov7x_training.pt' --name yolov7x_criadreos-mosquito --hyp data/hyp.scratch.custom.yaml

```

8. Al finalizar el entrenamiento, en la carpeta `/runs/train/` se guardan los resultados del entrenamiento en una carpeta por entrenamiento ejecutado. En esta carpeta se guarda la información de

presicion del modelo, el recall, el mAP, matrices de confusión, predicciones, etc. También podemos encontrar en la misma ruta una carpeta `weights` donde se guardan los pesos generados en el entrenamiento. Los pesos automáticamente se guardan cada 25 épocas, y se guardan los mejores pesos durante el entrenamiento basado en el *mAP*.

9. Para probar el modelo entrenado, utilizamos el mejor peso obtenido durante el entrenamiento, este se guarda con el nombre `best.pt`, el comando para realizar dicha prueba es la siguiente:

```
1     python test.py --data data/criaderos_mosquitos.yaml --img 1024 --batch 8 -- task test --
    weights /runs/train/yolov7x_criadreos-mosquito/weights/best.pt --name yolov7x_criaderos-
    mosquitos_test
2
```

Al finalizar la prueba del modelo se generará una carpeta en
`/runs/test/yolov7x_criaderos-mosquitos_test`
donde se guardarán todas las métricas del modelo. Por cada test realizado se generará una nueva carpeta.

Lineas de comando

Algunas lineas de comando importantes para realizar el entrenamiento, validación, prueba o predicción de un modelo así como su configuración se muestran a continuación:

- **Entrenamiento:**

```
1     python train.py --workers 8 --device 0 --batch-size 32 --data data/custom.yaml --img
    640 640 --cfg cfg/training/yolov7-custom.yaml --weights 'yolov7_training.pt' -- epochs
    300 --name yolov7-custom --hyp data/hyp.scratch.custom.yaml
2
```

Código 4.8: Comando de entrenamiento para YOLOv7.

  - `-data`

    Tipo: `str` *Por defecto:* `data/coco.yaml`

    Ruta al archivo `data.yaml` que especifica la configuración del conjunto de datos. Este archivo contiene parámetros específicos del conjunto de datos, incluidas las rutas a los archivos de entrenamiento y validación, nombres de las clases y número de clases.

- `-weights`

  Tipo: `str` *Por defecto:* `yolov7.pt`

  Ruta al archivo de pesos iniciales del modelo (transfer learning). Estos son pesos iniciales que se irán modificando de acuerdo al entrenamiento.

- `-cfg`

  Tipo: `str` *Por defecto:* ''

  Ruta al archivo de configuración del modelo `model.yaml`. Donde se muestra la arquitectura del modelo, se indican los `anchors` y se muestra el número de clases.

- `-hyp`

  Tipo: `str` *Por defecto:* `data/hyp.scratch.p5.yaml`

  Ruta al archivo de hiperparámetros utilizados durante el entrenamiento. La mayoría de los hiperparámetros son para el aumento de datos.

- `-device`

  Tipo: `str` *Por defecto:* ''

  Dispositivo en el que se ejecutará el entrenamiento, como `cpu`, `0` para GPU, etc.

- `-epochs`

  Tipo: `int` *Por defecto:* `300`

  Número total de épocas de entrenamiento. Cada época representa una pasada completa por todo el conjunto de datos. Ajustar este valor puede afectar a la duración del entrenamiento y al rendimiento del modelo.

- `-workers`

  Tipo: `int` *Por defecto:* `8`

  Número máximo de procesos paralelos (hilos o workers) que se utilizan para cargar los datos desde el disco durante el entrenamiento. Un mayor número puede acelerar el preprocesamiento y la lectura de datos, especialmente en conjuntos de datos grandes, aunque también puede aumentar el uso de memoria.

- `-batch-size`

  Tipo: `int` *Por defecto:* `16`

Tamaño total del lote de entrenamiento dividido entre todas las GPU disponibles. Este valor impacta directamente en el uso de memoria y la estabilidad del entrenamiento. Batches más grandes pueden mejorar la estimación del gradiente, pero requieren más memoria.

- `-img`

  Tipo: `list[int]` *Por defecto:* `[640, 640]`

  Tamaño de imagen objetivo para el entrenamiento. Todas las imágenes se redimensionan a esta dimensión antes de introducirlas en el modelo. Afecta a la precisión del modelo y a la complejidad computacional.

- `-name`

  Tipo: `str` *Por defecto:* `exp`

  Nombre de la carpeta donde se guardarán los resultados: `project/name`.

- **Prueba:**

```
1   python test.py --data data/coco.yaml --img 640 --batch 32 --conf 0.001 --iou 0.65 --
    device 0 --weights best.pt --name yolov7_640_val
2
```

Código 4.9: Comando de prueba para YOLOv7.

- `-conf`

  Tipo: `float` *Por defecto:* `0.001`

  Umbral de confianza mínimo para las detecciones. Valores bajos permiten recuperar más objetos, aunque pueden generar más falsos positivos. Se usa para calcular curvas de precisión-recuperación.

- `-iou`

  Tipo: `float` *Por defecto:* `0.65`

  Umbral de intersección sobre unión para la supresión no máxima. Valores bajos eliminan más cajas superpuestas, lo cual ayuda a reducir duplicados en la detección.

- **Inferencia:**

  Para imagenes:

```
1     python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source inference/
    images/horses.jpg
2
```

Código 4.10: Comando de inferencia para imágenes en YOLOv7.

Para realizar inferencia en un video:

```
1     python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source yourvideo.mp4
2
```

Código 4.11: Comando de inferencia para videos en YOLOv7.

- `-source`

  Tipo: `str` *Por defecto:* `inference/images`

  Especifica la fuente de entrada para la detección. Puede ser una imagen individual, una carpeta de imágenes, un video o una webcam. Define el origen de los datos que el modelo procesará.

Configuración de hiperparámetros y default

La configuración de hiperparámetros se realiza en el archivo ubicado en la ruta `data/hyp.scratch.custom.yaml`, estos valores se dejaron por default.

Parámetro	Valor	Descripción
`lr0`	0.01	Tasa de aprendizaje inicial (SGD = 1E-2, Adam = 1E-3).
`lrf`	0.1	Tasa final del ciclo OneCycleLR (`lr0 * lrf`).
`momentum`	0.937	Momento de SGD / beta1 de Adam.
`weight_decay`	0.0005	Penalización L2 para regularización.
`warmup_epochs`	3.0	Épocas de calentamiento.
`warmup_momentum`	0.8	Momento inicial en el calentamiento.
`warmup_bias_lr`	0.1	Tasa de aprendizaje inicial para bias.
`box`	0.05	Ganancia para pérdida de las cajas.
`cls`	0.3	Ganancia para pérdida de clasificación.
`cls_pw`	1.0	Peso positivo para pérdida BCE de clase.
`obj`	0.7	Ganancia para pérdida de objeto (escala con píxeles).
`obj_pw`	1.0	Peso positivo para pérdida BCE de objeto.
`iou_t`	0.20	Umbral de IoU para entrenamiento.
`anchor_t`	4.0	Umbral de múltiplo del ancla.
`fl_gamma`	0.0	Gamma para focal loss.
`hsv_h`	0.015	Aumento de tono HSV de imagen.
`hsv_s`	0.7	Aumento de saturación HSV de imagen.
`hsv_v`	0.4	Aumento de valor HSV de imagen.
`degrees`	0.0	Rotación de imagen (+/- grados).
`translate`	0.2	Traslación de imagen (+/- fracción).
`scale`	0.5	Escalado de imagen (+/- ganancia).
`shear`	0.0	Cizalladura de imagen (+/- grados).
`perspective`	0.0	Perspectiva de imagen (+/- fracción).
`flipud`	0.0	Volteo vertical (probabilidad).
`fliplr`	0.5	Volteo horizontal (probabilidad).
`mosaic`	1.0	Aumento de tipo mosaico (probabilidad).
`mixup`	0.0	Aumento de tipo mixup (probabilidad).
`copy_paste`	0.0	Copiar y pegar imágenes (probabilidad).
`paste_in`	0.0	Variante de copiar y pegar (probabilidad).
`loss_ota`	1	Usar `ComputeLossOTA` (0 para entrenamiento más rápido).

Tabla 4.7: Configuración de hiperparámetros utilizada para entrenar los modelos de YOLOv7.

4.1.3. YOLOv8 y v12

Los primeros modelos YOLO comprendidos de la versión 1 a la 4, habían sido desarrollado por investigadores y académicos, quienes a su vez se encargaban de desarrollar dichos modelos para publicarlos en repositorios. A partir del año 2020 con la publicación de YOLOv5, la empresa Ultralytics comenzó a gestionar y desarrollar los modelos de la familia YOLO posteriores, liderando la implementación de los

modelos YOLO. La implementación de los modelos YOLOv8 y YOLOv12 son basados en la librería de Ultralytics [34], [46].

Dependencias y configuración recomendada

Existe una configuración recomendada y dependencias o requisitos para la instalación de la librería Ultralytics (veáse 4.8), si bien las dependencias son instaladas automáticamente con la librería Ultralytics, se requiere prestar atención a la configuración.

Requisitos	Configuración recomendada
numpy	Python 3.8+
matplotlib	PyTorch 1.10+
pandas	NVIDIA GPU con CUDA 11.2+
pyyaml	Más de 8 GB de RAM
Pillow	
psutil	
requests≥2.23.0	
tqdm	
torch≥1.8.0	
torchvision≥0.9.0	Más de 50 GB de espacio libre en disco (para almacenamiento de conjuntos de datos y formación de modelos)

Tabla 4.8: Dependencias básicas y configuración recomendada para Ultralytics.

Instalación

La instalación de la librería Ultralytics para el uso de los modelos YOLO es aún más sencilla que los modelos anteriores, pero más restrictivo a las configuraciones. Presentamos principalmente dos maneras de realizar la instalación:

- Mediante el uso de `pip`:

```
1     pip install ultralytics
2
```

Código 4.12: Instalación de Ultralytics usando pip.

- Mediante anaconda (implementado por nosotros):

```
1     conda install -c pytorch -c nvidia -c conda-forge pytorch torchvision pytorch-cuda=11.8
    ultralytics
2
```

Código 4.13: Instalación de Ultralytics usando conda.

Esta última forma de instalación es útil cuando se encuentra en un entorno CUDA, ya que este comando instala Ultralytics, Pytorch y Pytorch-cuda. Esto permite al gestor de paquetes conda resolver cualquier conflicto. Estas dos formas, también instalan las dependencias necesarias. Existen otras formas de realizar la instalación mediante docker o clonando el repositorio de Ultralytics, también hay formas más personalizadas de instalación que pueden verse fácilmente en la página oficial, en el apartado de `inicio rápido/instalación`.

Líneas de comando

Existen 6 modalidades para operar los modelos: entrenamiento, validación, predicción, exportación de modelos, seguimiento de objetos y modelos de evaluación comparativa. En este apartado nos enfocaremos en los primeros 3. Todos los modos pueden ser ejecutados de dos formas distintas, la primera es mediante un programa de python y la segunda mediante comandos en la interfaz de líneas de comando. Por la facilidad de configuración y ejecución se optó por usar un programa de python para cada uno de los primeros 3 modos. Es importante contar con tres archivos indispensables para los distintos modos:

- Descargar la configuración `yaml` del modelo a utilizar de la página oficial de Ultralytics. En este archivo se muestra la arquitectura de la red, que puede ser modificada y el número de categorías a considerar. Esencialmente nosotros modificamos únicamente el valor del número de categorías a 5.

- Contar con la configuración propia de nuestros datos en formato `yaml` donde se indican las rutas de nuestros datos de entrenamiento, validación y test, así como el número de categorías y el nombre de las categorías. Este archivo puede ser exactamente de la misma estructura utilizada en la versión de YOLOv7.

- Los pesos del modelo a utilizar para realizar transfer learning.

Todos los modelos implementados en Ultralytics, incluidos YOLOv8 y YOlOv12 se utilizan con los mismo comandos que se mostrarán, únicamente basta con utilizar los tres principales archivos previamente mencionados para cada uno de los modelos. Las líneas de comando son las siguientes:

- **Entrenamiento:**

```
from ultralytics import YOLO

# cargar un modelo
model = YOLO("yolo12x.yaml") # construye un nuevo modelo de YAML
model = YOLO("yolo12x.pt") # carga el modelo preentrenado
model = YOLO("yolo12x.yaml").load("yolo12x.pt") # se realiza la transferencia de pesos

# entrenamiento del modelo
model.train(data="criaderos_mosquitos.yaml", epochs=300, imgsz=1024, batch=8, device='1', patience=100)

```

Código 4.14: Algoritmo de entrenamiento para modelos YOLO en Ultralytics.

- `data`

  Tipo: `str` *Por defecto:* `None`

  Ruta al archivo de configuración del conjunto de datos (por ejemplo, `coco8.yaml`).

- `epochs`

  Tipo: `int` *Por defecto:* `100`

  Número de épocas (recorridos completos del conjunto de entrenamiento).

- `imgsz`

  Tamaño de que se redimensionarán las imágenes para la entrada del modelo.

- `batch`

  Tipo: `int` *Por defecto:* `16`

  Tamaño del lote, con tres modos: como número entero (por ejemplo, `batch=16`), modo automático para una utilización del 60 % de la memoria GPU (`batch=-1`), o modo automático con fracción de uso especificada (`batch=0.70`).

- `device`

  Tipo: `int o str` *Por defecto:* `None`

Especifica el dispositivo o dispositivos informáticos para el entrenamiento: un único GPU (device=0), varias GPU (device=[0,1]), CPU (device=cpu), MPS para el silicio de Apple (device=mps), o la selección automática de la GPU más inactiva (device=-1) o varias GPU inactivas (device=[-1,-1]).

- `patience`

  Tipo: `int` *Por defecto:* `100`

  Número de épocas que hay que esperar sin que mejoren las métricas de validación antes de detener el entrenamiento. Ayuda a evitar el sobreajuste al detener el entrenamiento cuando el rendimiento se estanca.

- **Validación:**

```
from ultralytics import YOLO

# cargar modelo
model = YOLO("yolo12x.pt")  # cargar un modelo oficial
model = YOLO('path/to/best.pt') # cargar los mejores pesos

# configuracion de validacion
validation_results = model.val(data="criaderos_mosquitos.yaml", imgsz=1024, batch=8, conf=0.5, device="1", split='test')


```

Código 4.15: Algoritmo de validación para modelos YOLO en Ultralytics.

- `conf`

  Tipo: `float` *Por defecto:* `0.001`

  Establece el umbral mínimo de confianza para las detecciones. Los valores más bajos aumentan la recuperación, pero pueden introducir más falsos positivos. Se utiliza durante la validación para calcular las curvas de precisión-recuperación.

- `split`

  Tipo: `str` *Por defecto:* `'val'`

  Determina la división del conjunto de datos que se utilizará para la validación (`val`, `test` o

`train`). Permite flexibilidad a la hora de elegir el segmento de datos para la evaluación del rendimiento.

- **Predicción:**

```
from ultralytics import YOLO

# cargar modelo
model = YOLO("yolo12x.pt")  # cargar un modelo oficial
model = YOLO('path/to/best.pt') # cargar los mejores pesos

# configuracion de inferencia
model.predict("path/to/best.pt/images", save=True, imgsz=1024, conf=0.5, iou=0.7)

```

Código 4.16: Comandos de predicción de modelos YOLO con Ultralytics.

- `iou`

  Tipo: `float` *Por defecto:* `0.7`

  Umbral de intersección sobre unión (IoU) para la supresión no máxima (NMS). Los valores más bajos dan lugar a menos detecciones al eliminar las cajas superpuestas, lo que resulta útil para reducir los duplicados.

Existen otras tareas que pueden realizar estos modelos, como la tarea de segmentación, estimación de la pose, detección de objetos orientados. Para más detalles consultar la página de Ultralytics.

Configuración de hiperparámetros

En la tabla 4.9 se muestran los hiperparámetros utilizados que se ecuentran en la ruta `ultralytics/cfg/default.yaml`. Estos fueron dejados con los valores por defecto.

Argumento	Tipo	Por defecto	Gama	Descripción
`hsv_h`	float	0.015	0.0 – 1.0	Ajusta el tono de la imagen en una fracción de la rueda de color, introduciendo variabilidad cromática.
`hsv_s`	float	0.7	0.0 – 1.0	Altera la saturación de la imagen, afectando a la intensidad de los colores.
`hsv_v`	float	0.4	0.0 – 1.0	Modifica el brillo de la imagen, ayudando en condiciones de iluminación diversas.
`degrees`	float	0.0	0.0 – 180	Rota la imagen aleatoriamente dentro del rango especificado.
`translate`	float	0.1	0.0 – 1.0	Traslada la imagen horizontal y verticalmente una fracción del tamaño.
`scale`	float	0.5	$\geq$ 0.0	Escala la imagen para simular distintas distancias.
`shear`	float	0.0	-180 – +180	Corta la imagen para simular diferentes ángulos.
`perspective`	float	0.0	0.0 – 0.001	Aplica una transformación de perspectiva aleatoria.
`flipud`	float	0.0	0.0 – 1.0	Voltea la imagen verticalmente con cierta probabilidad.
`fliplr`	float	0.5	0.0 – 1.0	Voltea la imagen horizontalmente con cierta probabilidad.
`bgr`	float	0.0	0.0 – 1.0	Cambia RGB a BGR con la probabilidad especificada.
`mosaic`	float	1.0	0.0 – 1.0	Combina cuatro imágenes en una para simular escenas complejas.
`mixup`	float	0.0	0.0 – 1.0	Mezcla imágenes y etiquetas para introducir variabilidad visual.
`cutmix`	float	0.0	0.0 – 1.0	Combina partes de imágenes, manteniendo regiones diferenciadas.
`copy_paste`	float	0.0	0.0 – 1.0	(Sólo segmentación) Copia y pega objetos entre imágenes.
`copy_paste_mode`	str	flip	-	(Sólo segmentación) Estrategia usada para copy-paste: `flip` o `mixup`.
`auto_augment`	str	randaugment	-	(Sólo clasificación) Aplica políticas como `randaugment`, `autoaugment` o `augmix`.
`erasing`	float	0.4	0.0 – 0.9	(Sólo clasificación) Borra regiones de la imagen para forzar atención en características no evidentes.

Tabla 4.9: Ajuste de aumento de datos e hiperparámetros para los modelos YOLO en Ultralytics.

4.2. Características del cómputo utilizado

4.2.1. Características Principales del Sistema

- **Procesador (CPU)**: 2x Intel(R) Xeon(R) Gold 5222 @ 3.80GHz
- **Memoria RAM**: 46 GiB
- **Tarjeta Gráfica (GPU)**: ASPEED Graphics Family
- **Sistema operativo**: Ubuntu 20.04.6 LTS.

4.2.2. Detalles de GPUs NVIDIA

El sistema cuenta con dos tarjetas gráficas NVIDIA instaladas:

- **GPU 0: NVIDIA TITAN RTX**
  - **Memoria VRAM**: 24.220 GiB
- **GPU 1: NVIDIA Quadro RTX 8000**
  - **Memoria VRAM**: 48.601 GiB

En la cual la mayor parte del trabajo fue realizada en la tarjeta Nvidia Quadro RTX 8000.