

# Práctica de Deep Learning

Sergio Altares López & Ángela Ribeiro - CAR (CSIC-UPM)  
3 de Julio de 2024



# ¿Qué es detectar?

Detectar es el proceso de identificar, **reconocer o descubrir la presencia de algo**. Esto puede implicar la observación, el uso de instrumentos o técnicas específicas para encontrar algo que no es inmediatamente evidente.

Para los seres humanos, detectar implica usar los **sentidos, que actúan como sensores naturales**, para identificar y percibir estímulos del entorno.



- **Función:** Detecta la luz y permite la percepción de colores, formas, tamaños y movimientos.
- **Proceso:** La luz entra en los ojos, se enfoca a través del cristalino y llega a la retina, donde las células fotorreceptoras (bastones y conos) convierten la luz en señales eléctricas que el cerebro interpreta como imágenes.



- **Función:** Detecta las moléculas químicas en el aire y permite la percepción de olores.
- **Proceso:** Las moléculas de olor entran en la nariz y se disuelven en la cavidad nasal, donde se unen a los receptores olfativos. Estos receptores envían señales al cerebro, que interpreta las diferentes combinaciones de moléculas como olores específicos.



- **Función:** Detecta la presión, temperatura, dolor y textura.
- **Proceso:** La piel contiene una variedad de receptores que responden a diferentes estímulos táctiles. Cuando se estimulan, estos receptores envían señales a través de las fibras nerviosas al cerebro, que interpreta las señales como sensaciones táctiles.



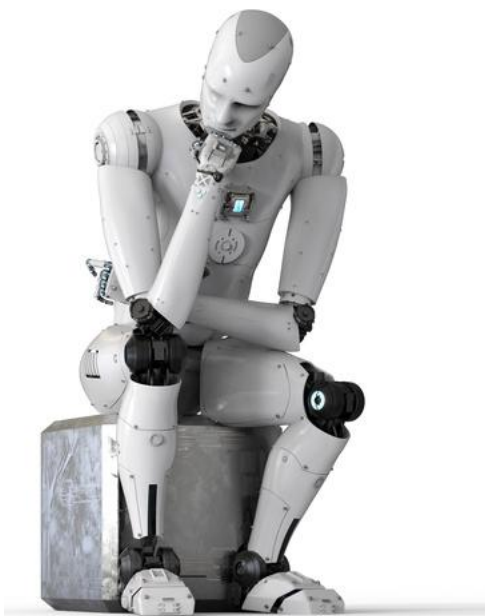
- **Función:** Detecta las ondas sonoras y permite la percepción del sonido, incluyendo su tono, volumen y dirección.
- **Proceso:** Las ondas sonoras hacen vibrar el tímpano, lo que a su vez mueve los huesecillos del oído medio. Estas vibraciones se transmiten al oído interno, donde las células ciliadas convierten las vibraciones en señales eléctricas que el cerebro interpreta como sonidos.



- **Función:** Detecta las moléculas químicas en los alimentos y permite la percepción de sabores (dulce, salado, ácido, amargo y umami).
- **Proceso:** Las moléculas de los alimentos se disuelven en la saliva y entran en contacto con las papilas gustativas, que contienen células receptoras del gusto. Estas células envían señales al cerebro, que interpreta las diferentes combinaciones de moléculas como sabores.



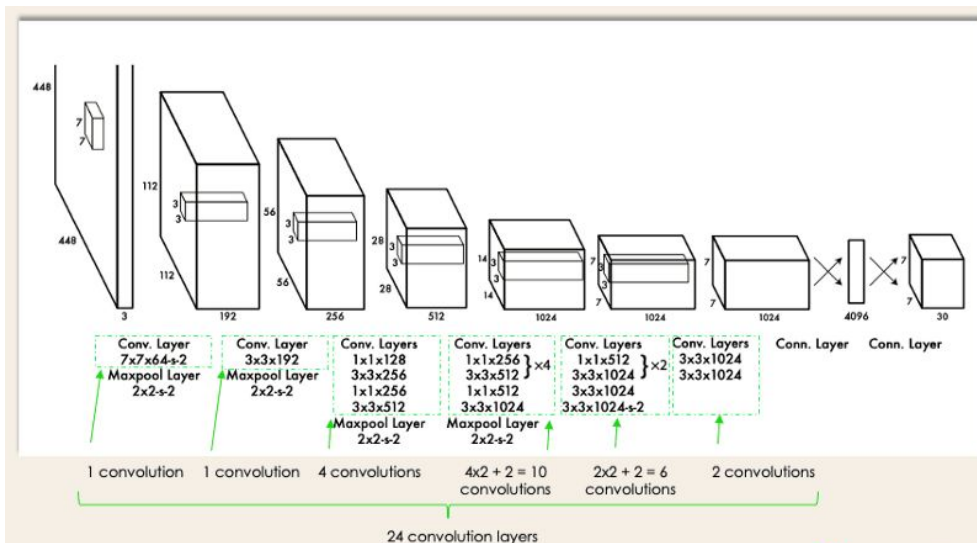
# ¿Cómo puede detectar una máquina?





# You Only Look Once (YOLO)

YOLO es un algoritmo de **detección de objetos** que realiza la tarea en una sola pasada de la red neuronal. Este enfoque le permite ser rápido, lo cual es importante para aplicaciones en tiempo real.





# You Only Look Once (YOLO)

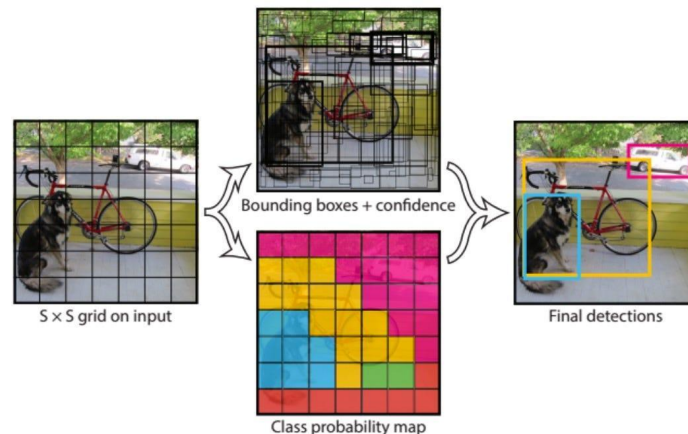
**1. Entrada de la Imagen:** La imagen de entrada se redimensiona a un tamaño fijo para ser procesada por la red.

**2. División en Cuadrícula:** La imagen redimensionada se divide en una cuadrícula de celdas.

**3. Predicción por Celda:** Cada celda de la cuadrícula es responsable de detectar objetos cuyo centro se encuentra dentro de esa celda. Para cada celda, la red predice un número fijo de cajas delimitadoras (bounding boxes), usualmente 2, y para cada caja:

- Coordenadas de la caja (posición y tamaño).
- Confianza de que la caja contiene un objeto y la precisión de esa predicción.
- Además, cada celda predice las probabilidades de las clases para los objetos dentro de esa celda.

**4. Extracción de Características:** La imagen pasa a través de varias capas convolucionales y de pooling para extraer características de alto nivel. Estas capas se encargan de detectar patrones como bordes, texturas y finalmente partes de objetos.





# ¿Qué vamos a hacer?

- Cargar un modelo YOLO v8.2
- Preparación de los datos para **reentrenar** el modelo para 5 clases de nuestro propio conjunto de datos
- Obtener las métricas de los modelos y saber interpretarlas
- Aplicación en un video e imágenes



Aplicación de red reentrenada a video



Aplicación de red reentrenada a imagen




# Datos

Es importante disponer de un Dataset con **gran número de muestras** de tal manera que el modelo pueda ver el mayor número de casos posible, sea robusto y pueda extrapolar: **generalización**.



Imagen

 0.829316 0.191781 0.118256 0.136986

Class

Etiqueta



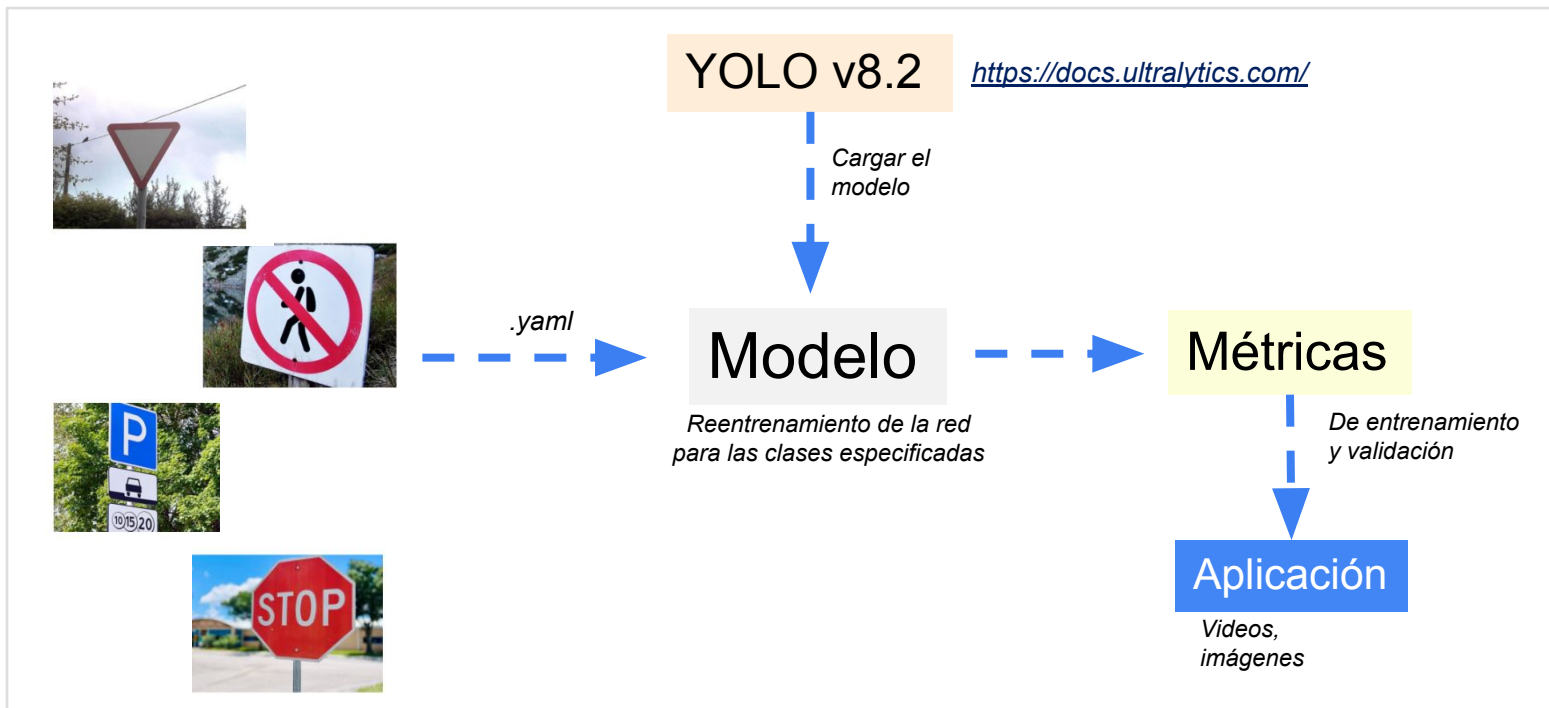
Imagen con la etiqueta



Los datos deben ser transformados a *.yaml* para que puedan ser leídos en el reentrenamiento.



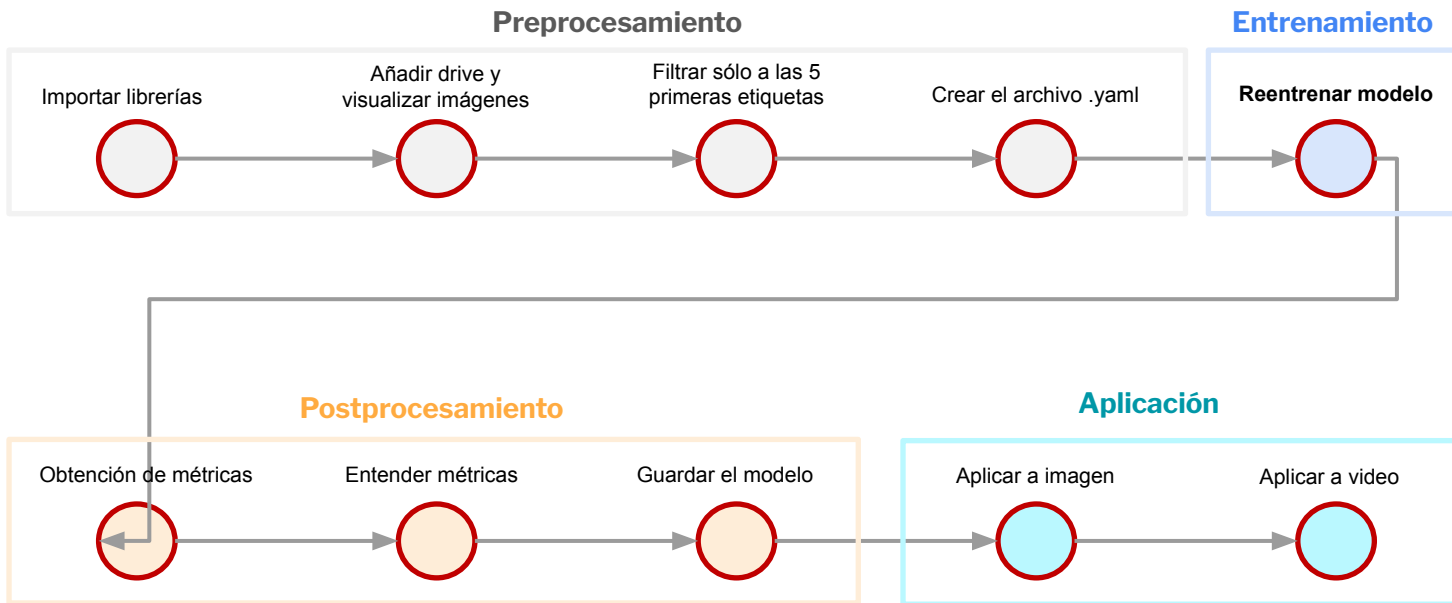
# Esquema de la práctica







# Pasos a realizar





# Pasos (I)

Próximo paso: Añadir drive y visualizar 5 imágenes del conjunto de datos

Instalamos e importamos la librerías necesarias.

En este caso vamos a utilizar Google colab, que nos permite utilizar una GPU T4, por lo que es importante cambiar la configuración del notebook



# Pasos (I)

Próximo paso: Añadir drive y visualizar 5 imágenes del conjunto de datos

Instalamos e importamos la librerías necesarias.

En este caso vamos a utilizar Google colab, que nos permite utilizar una GPU T4, por lo que es importante cambiar la configuración del notebook

```
!pip install ultralytics
# Install Essential Libraries
import locale
locale.getpreferredencoding = lambda: "UTF-8"
# Import Essential Libraries
import os
import random
import pandas as pd
from PIL import Image
import cv2
from ultralytics import YOLO
from IPython.display import Video
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='darkgrid')
import pathlib
import glob
import torch
from tqdm.notebook import trange, tqdm
import warnings
warnings.filterwarnings('ignore')
print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)
```

NVIDIA-SMI 535.104.05		Driver Version: 535.104.05		CUDA Version: 12.2	
GPU Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	
Fan Temp Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.
0 Tesla T4	Off	00000000:00:04.0 Off		0	
N/A 59C P8	10W / 70W	3MiB / 15360MiB	0%	Default	N/A



# Pasos (II)

Próximo paso: Filtrar únicamente por las 5 primeras etiquetas

Vamos a montar nuestro drive para poder guardar los resultados.

Y visualizamos 9 imágenes de entrenamiento para comprobar que las rutas son correctas.



# Pasos (II)

Próximo paso: Filtrar únicamente por las 5 primeras etiquetas

Vamos a montar nuestro drive para poder guardar los resultados.

Y visualizamos 9 imágenes de entrenamiento para comprobar que las rutas son correctas.

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
Image_dir = '/content/drive/MyDrive/dataset_vlc/images/train'

num_samples = 9
image_files = os.listdir(Image_dir)

# Randomly select num_samples images
rand_images = random.sample(image_files, num_samples)

fig, axes = plt.subplots(3, 3, figsize=(11, 11))

for i in range(num_samples):
    image = rand_images[i]
    ax = axes[i // 3, i % 3]
    ax.imshow(plt.imread(os.path.join(Image_dir, image)))
    ax.set_title(f'Image {i+1}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```





# Pasos (III)

Próximo paso: Visualizar 5 imágenes de cada una de las clases filtradas con los nombres

Nuestro conjunto de datos tiene 9 etiquetas entre señales verticales y de suelo. En este caso, vamos a simplificar el proceso utilizando únicamente las 5 primeras etiquetas correspondientes a las señales verticales.

Para ello, filtramos las etiquetas iterando en los archivos .txt.



# Pasos (III)

Nuestro conjunto de datos tiene 9 etiquetas entre señales verticales y de suelo. En este caso, vamos a simplificar el proceso utilizando únicamente las 5 primeras etiquetas correspondientes a las señales verticales.

Para ello, filtramos las etiquetas iterando en los archivos.txt.

```
import os
import yaml
import random
from collections import defaultdict
from PIL import Image
import matplotlib.pyplot as plt
```

Próximo paso: Visualizar 5 imágenes de cada una de las clases filtradas con los nombres

`desired_classes = [0, 1, 2, 3, 4]` ← 5 primeras clases

```
def filter_labels(labels_path, desired_classes):
    # Filtrar las etiquetas para mantener sólo las clases deseadas
    for root, dirs, files in os.walk(labels_path):
        for file in files:
            if file.endswith(".txt"):
                file_path = os.path.join(root, file)
                filtered_lines = []
                with open(file_path, 'r') as f:
                    for line in f.readlines():
                        class_id = int(line.split()[0])
                        if class_id in desired_classes:
                            filtered_lines.append(line)
                # Sobrescribir el archivo de etiquetas con las líneas filtradas
                with open(file_path, 'w') as f:
                    f.writelines(filtered_lines)

def get_unique_class_ids(labels_path):
    class_ids = set()
    for root, dirs, files in os.walk(labels_path):
        for file in files:
            if file.endswith(".txt"):
                file_path = os.path.join(root, file)
                with open(file_path, 'r') as f:
                    for line in f.readlines():
                        class_id = int(line.split()[0])
                        class_ids.add(class_id)
    return sorted(class_ids)
```



# Pasos (IV)

Próximo paso: Crear el archivo .yaml con los datos

Sobreescribimos los archivos .txt para dejar únicamente las etiquetas a entrenar.

Le damos un nombre a cada etiqueta única y mostramos 5 ejemplos de cada una de las etiquetas.

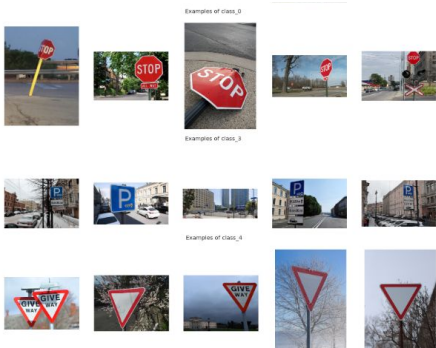


# Pasos (IV)

Próximo paso: Crear el archivo .yaml con los datos

Sobreescribimos los archivos .txt para dejar únicamente las etiquetas a entrenar.

Le damos un nombre a cada etiqueta única y mostramos 5 ejemplos de cada una de las etiquetas.



```
def create_class_name_mapping(labels_path, desired_classes):
    class_names = {}
    for root, dirs, files in os.walk(labels_path):
        for file in files:
            if file.endswith(".txt"):
                file_path = os.path.join(root, file)
                with open(file_path, 'r') as f:
                    for line in f.readlines():
                        class_id = int(line.split()[0])
                        if class_id in desired_classes and class_id not in class_names:
                            class_names[class_id] = f'class_{class_id}'
    return class_names

def get_examples_by_class(labels_path, desired_classes, num_examples=5):
    examples = defaultdict(list)
    for root, dirs, files in os.walk(labels_path):
        for file in files:
            if file.endswith(".txt"):
                file_path = os.path.join(root, file)
                image_path = file_path.replace('labels', 'images').replace('.txt', '.jpg')
                with open(file_path, 'r') as f:
                    for line in f.readlines():
                        class_id = int(line.split()[0])
                        if class_id in desired_classes:
                            examples[class_id].append(image_path)
                            break # We only need one instance of each class per image
    # Select num_examples from each class
    for class_id in examples:
        examples[class_id] = random.sample(examples[class_id], min(num_examples, len(examples[class_id])))
    return examples
```

# Pasos (V)

Una vez tenemos las rutas y las etiquetas claras, pasamos a generar el archivo `.yaml` con los datos de train y validación.

Este archivo es el que se utilizará para el reentrenamiento del modelo YOLO importado.

# Pasos (V)

Una vez tenemos las rutas y las etiquetas claras, pasamos a generar el archivo `.yaml` con los datos de train y validación.

Este archivo es el que se utilizará para el reentrenamiento del modelo YOLO importado.

```
# Rutas a Las carpetas de etiquetas
train_labels_path = "/content/drive/MyDrive/dataset_vlc/labels/train"
val_labels_path = "/content/drive/MyDrive/dataset_vlc/labels/val"

# Filtrar Las etiquetas
filter_labels(train_labels_path, desired_classes)
filter_labels(val_labels_path, desired_classes)

# Obtener IDs de clases únicos de Las anotaciones filtradas
train_class_ids = get_unique_class_ids(train_labels_path)
val_class_ids = get_unique_class_ids(val_labels_path)

# Combinar y ordenar Los IDs de clases únicos
all_class_ids = sorted(set(train_class_ids + val_class_ids))

# Crear un mapeo de nombres de clases basado en Los archivos de etiquetas filtradas
class_name_mapping = create_class_name_mapping(train_labels_path, desired_classes)

# Obtener ejemplos de cada clase
examples = get_examples_by_class(train_labels_path, desired_classes)

# Mostrar ejemplos
show_examples(examples, class_name_mapping)

# Crear una lista ordenada de nombres de clases según Los IDs de clase ordenados
class_names = [class_name_mapping[class_id] for class_id in all_class_ids]

# Crear el diccionario para el archivo data.yaml
data = {
    'train': "/content/drive/MyDrive/dataset_vlc/images/train",
    'val': "/content/drive/MyDrive/dataset_vlc/images/val",
    'nc': len(class_names),
    'names': class_names
}

# Guardar el archivo data.yaml
data_yaml_path = "/content/drive/MyDrive/dataset_vlc/data.yaml"
with open(data_yaml_path, 'w') as yaml_file:
    yaml.dump(data, yaml_file, default_flow_style=False)

print(f"Archivo data.yaml creado con éxito en: {data_yaml_path}")
```

← Convertimos a .yaml



# Pasos (VI)

Próximo paso: Ver las evoluciones de las métricas guardadas en las carpetas generadas.

Cargamos el modelo YOLO v8 y lo reentrenamos con nuestros datos utilizando GPU T4.

Vemos la evolución de las métricas durante el entrenamiento. Por defecto nos proporciona el mAP50 y mAP95, además de la precisión de la clase y de la bounding box.

# Pasos (VI)

Cargamos el modelo YOLO v8 y lo reentrenamos con nuestros datos utilizando GPU T4.

Vemos la evolución de las métricas durante el entrenamiento. Por defecto nos proporciona el mAP50 y mAP95, además de la precisión de la clase y de la bounding box.

```
!nvidia-smi
import torch
torch.cuda.is_available()
os.environ['CUDA_VISIBLE_DEVICES']: 16
# Build from YAML and transfer weights
Final_model = YOLO('yolov8n.yaml').load('yolov8n.pt')
Final_model.to('cuda')

# Training The Final Model
Result_Final_model = Final_model.train(data="/content/drive/MyDrive/dataset_vlc/data.yaml", epochs=15, imgsz = 224,
                                       batch = 64 ,lr=0.0001, dropout= 0.2, device=1)
```

Próximo paso: Ver las evoluciones de las métricas guardadas en las carpetas generadas.

Thu Jun 20 16:03:25 2024

NVIDIA-SMI 535.184.05		Driver Version: 535.184.05		CUDA Version: 12.2	
GPU Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.
					MIG M.
0 Tesla T4		Off	00000000:00:04:0	Off	0
N/A	50C	10W / 70W	3MiB / 15360MiB		Default
	P8			0%	N/A

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size		
1/15	1.47G	0.798	3.799	1.018	4	224: 100%	17/17	[00:47<00:00, 2.78 s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	2/2 [00:07<00:00, 3.53s/it]
	all	219	134	0.0039	0.761	0.258	0.221	



Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size		
15/15	1.43G	0.4648	0.5439	0.8437	2	224: 100%	17/17	[00:30<00:00, 1.77 s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	2/2 [00:01<00:00, 1.18it/s]
	all	219	134	0.966	0.908	0.948	0.843	



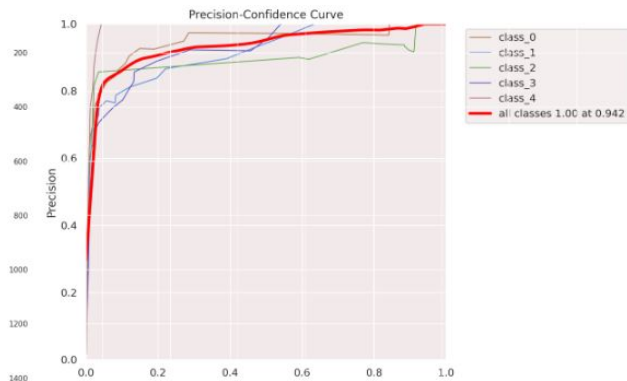
# Pasos (VII)

Se crean carpetas durante el entrenamiento donde se guardan resultados con lo que somos capaces de ver las métricas y las evoluciones de la precisión y las funciones de coste.

Próximo paso: Entender las métricas obtenidas de los modelos

# Pasos (VII)

Se crean carpetas durante el entrenamiento donde se guardan resultados con lo que somos capaces de ver las métricas y las evoluciones de la precisión y las funciones de coste.



Próximo paso: Entender las métricas obtenidas de los modelos

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95)
all	219	134	0.966	0.919	0.949	0.844
class_0	32	41	0.972	0.844	0.933	0.824
class_1	27	27	0.96	0.963	0.988	0.913
class_2	20	20	0.896	0.9	0.877	0.796
class_3	21	25	1	0.918	0.953	0.822
class_4	20	21	1	0.971	0.995	0.863

```
list_of_metrics = ["P_curve.png", "R_curve.png", "confusion_matrix.png"]
```

```
# Load the image
for i in list_of_metrics:
    image = cv2.imread(f'/content/runs/detect/train/{i}')

# Create a larger figure
plt.figure(figsize=(16, 12))

# Display the image
plt.imshow(image)

# Show the plot
plt.show()
```

```
Result_Final_model = pd.read_csv('/content/runs/detect/train/results.csv')
Result_Final_model.tail(40)
```



# Pasos (VIII)

Se crean carpetas durante el entrenamiento donde se guardan resultados con lo que somos capaces de ver las métricas y las evoluciones de la precisión y las funciones de coste.

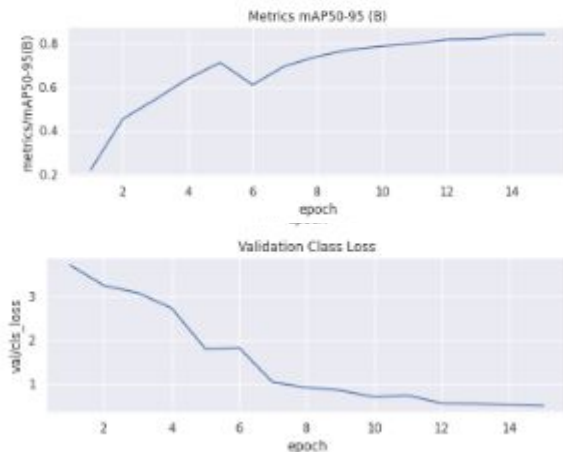
Próximo paso: usar el mejor modelo mediante la aplicación de los mejores pesos y guardar el modelo



# Pasos (VIII)

Próximo paso: usar el mejor modelo mediante la aplicación de los mejores pesos y guardar el modelo

Se crean carpetas durante el entrenamiento donde se guardan resultados con lo que somos capaces de ver las métricas y las evoluciones de la precisión y las funciones de coste.



```
# Read the results.csv file as a pandas dataframe
df = Result_Final_model
Result_Final_model.columns = df.columns.str.strip()
# Create subplots
fig, axs = plt.subplots(nrows=5, ncols=2, figsize=(15, 15))

# Plot the columns using seaborn
sns.lineplot(x='epoch', y='train/box_loss', data=df, ax=axs[0,0])
sns.lineplot(x='epoch', y='train/cls_loss', data=df, ax=axs[0,1])
sns.lineplot(x='epoch', y='train/dfl_loss', data=df, ax=axs[1,0])
sns.lineplot(x='epoch', y='metrics/precision(B)', data=df, ax=axs[1,1])
sns.lineplot(x='epoch', y='metrics/recall(B)', data=df, ax=axs[2,0])
sns.lineplot(x='epoch', y='metrics/mAP50(B)', data=df, ax=axs[2,1])
sns.lineplot(x='epoch', y='metrics/mAP50-95(B)', data=df, ax=axs[3,0])
sns.lineplot(x='epoch', y='val/box_loss', data=df, ax=axs[3,1])
sns.lineplot(x='epoch', y='val/cls_loss', data=df, ax=axs[4,0])
sns.lineplot(x='epoch', y='val/dfl_loss', data=df, ax=axs[4,1])

# Set titles and axis labels for each subplot
axs[0,0].set(title='Train Box Loss')
axs[0,1].set(title='Train Class Loss')
axs[1,0].set(title='Train DFL Loss')
axs[1,1].set(title='Metrics Precision (B)')
axs[2,0].set(title='Metrics Recall (B)')
axs[2,1].set(title='Metrics mAP50 (B)')
axs[3,0].set(title='Metrics mAP50-95 (B)')
axs[3,1].set(title='Validation Box Loss')
axs[4,0].set(title='Validation Class Loss')
axs[4,1].set(title='Validation DFL Loss')

plt.suptitle('Training Metrics and Loss', fontsize=24)
plt.subplots_adjust(top=0.8)
plt.tight_layout()
plt.show()
```



# Pasos (IX)

Próximo paso: Aplicación del modelo a imágenes

El archivo que se crea como “best.pt” es el modelo que mejores resultados ha dado, y por ende se guardan los pesos. En este caso, Valid\_model, es el mejor modelo para nuestras etiquetas.

Guardamos el modelo como .onnx para futuros usos.

# Pasos (IX)

El archivo que se crea como “best.pt” es el modelo que mejores resultados ha dado, y por ende se guardan los pesos. En este caso, Valid\_model, es el mejor modelo para nuestras etiquetas.

Guardamos el modelo como .onnx para futuros usos.

```
# Loading the best performing model
Valid_model = YOLO('/content/runs/detect/train/weights/best.pt')

# Evaluating the model on the testset
metrics = Valid_model.val(split='val')

# final results
print("precision(B): ", metrics.results_dict["metrics/precision(B)"])
print("metrics/recall(B): ", metrics.results_dict["metrics/recall(B)"])
print("metrics/mAP50(B): ", metrics.results_dict["metrics/mAP50(B)"])
print("metrics/mAP50-95(B): ", metrics.results_dict["metrics/mAP50-95(B)"])
model_export = Valid_model.export(format='onnx') # export the model to ONNX format
```

Próximo paso: Aplicación del modelo a imágenes

```
Ultralytics YOLOv8.2.36 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8n summary (fused): 168 layers, 3006623 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/drive/MyDrive/dataset_vlc/labels/val.cache... 219 images, 99 backgrounds, 0 corrupt: 100%
| 219/219 [00:00<, ?it/s]
Class      Images  Instances  Box(P)      R      mAP50  mAP50-95: 100% 14/14 [00:09<0
0:00, 1.541it/s]
  all      219      134      0.967      0.918      0.95      0.842
  class_0    32       41      0.972      0.842      0.933      0.823
  class_1    27       27      0.97      0.963      0.989      0.913
  class_2    20       20      0.896      0.9      0.88      0.792
  class_3    21       25       1      0.916      0.953      0.817
  class_4    20       21       1      0.971      0.995      0.863
```

```
Speed: 0.1ms preprocess, 1.6ms inference, 0.0ms loss, 1.9ms postprocess per image
```

```
Results saved to runs/detect/val2
precision(B): 0.9674723899447635
metrics/recall(B): 0.9181663395511995
metrics/mAP50(B): 0.949897319033267
metrics/mAP50-95(B): 0.8417630322393288
```

```
Ultralytics YOLOv8.2.36 Python-3.10.12 torch-2.3.0+cu121 CPU (Intel Xeon 2.20GHz)
```

```
PyTorch: starting from '/content/runs/detect/train/weights/best.pt' with input shape (1, 3, 224, 224) BCHW and output shape (1, 9, 1029) (5.9 MB)
```

```
requirements: Ultralytics requirement ['onnx>=1.12.0'] not found, attempting AutoUpdate...
```

```
Collecting onnx>=1.12.0
```

```
Downloading onnx-1.16.1-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (15.9 MB)
```

```
15.9/15.9 MB 211.0 MB/s eta 0:00:00
```

```
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from onnx>=1.12.0) (1.25.2)
```

```
Requirement already satisfied: protobuf>=3.20.2 in /usr/local/lib/python3.10/dist-packages (from onnx>=1.12.0) (3.20.3)
```

```
Installing collected packages: onnx
```

```
Successfully installed onnx-1.16.1
```

```
requirements: AutoUpdate success 8.6s, installed 1 package: ['onnx>=1.12.0']
```

```
requirements: Restart runtime or rerun command for updates to take effect
```

```
ONNX: starting export with onnx 1.16.1 opset 17...
```

```
ONNX: export success 9.3s, saved as '/content/runs/detect/train/weights/best.onnx' (11.5 MB)
```

```
Export complete (10.7s)
```

```
Results saved to /content/runs/detect/train/weights
```

```
Predict: yolo predict task=detect model=/content/runs/detect/train/weights/best.onnx imgsz=224
```

```
Validate: yolo val task=detect model=/content/runs/detect/train/weights/best.onnx imgsz=224 data=/content/drive/MyD
```

```
rive/dataset_vlc/data.yaml
```

```
Visualize: https://netron.app
```



# Pasos (X)

Aplicamos nuestro modelo a imágenes. En este caso vamos a tomar 2 imágenes aleatorias del conjunto de train para ver como detecta el modelo.

Próximo paso: Aplicación del modelo a video

# Pasos (X)

Próximo paso: Aplicación del modelo a video

Aplicamos nuestro modelo a imágenes. En este caso vamos a tomar 2 imágenes aleatorias del conjunto de train para ver como detecta el modelo.

image 1/1 /content/drive/MyDrive/dataset\_vlc/images/train/1004.jpg: 128x224 1 class\_2, 98.1ms  
Speed: 1.1ms preprocess, 98.1ms inference, 1.9ms postprocess per image at shape (1, 3, 128, 224)



image 1/1 /content/drive/MyDrive/dataset\_vlc/images/train/1609.jpg: 128x224 1 class\_3, 16.3ms  
Speed: 1.0ms preprocess, 16.3ms inference, 1.9ms postprocess per image at shape (1, 3, 128, 224)



```
import matplotlib.pyplot as plt
import cv2
from ultralytics import YOLO

# Cargar el modelo YOLO
model = YOLO('/content/runs/detect/train/weights/best.pt') # Ajusta la ruta según tu modelo YOLO

# Ruta de la imagen en la que deseas hacer la predicción
image_path = '/content/drive/MyDrive/dataset_vlc/images/train/1004.jpg'

# Realizar la predicción
results = model.predict(source=image_path)

# Obtener la imagen original
image = cv2.imread(image_path) # Lee la imagen con OpenCV para mantener los colores correctos

# Plotear la imagen con matplotlib
plot = results[0].plot()
plt.imshow(cv2.cvtColor(plot, cv2.COLOR_BGR2RGB)) # Convierte de BGR (formato de OpenCV) a RGB (formato de matplotlib)
plt.axis('off') # Deshabilita los ejes
plt.show()

# Ruta de la imagen en la que deseas hacer la predicción
image_path = '/content/drive/MyDrive/dataset_vlc/images/train/1609.jpg'

# Run inference on an image
results = model.predict(source=image_path)

# Plot inference results
plot = results[0].plot()
plt.imshow(cv2.cvtColor(plot, cv2.COLOR_BGR2RGB)) # Convierte de BGR (formato de OpenCV) a RGB (formato de matplotlib)
plt.axis('off') # Deshabilita los ejes
plt.show()
```



# Pasos (XI)

Aplicamos nuestro modelo a un video en .mp4. Se aplica a cada fotograma y vemos si detecta alguna señal o no.

# Pasos (XI)

Aplicamos nuestro modelo a un video en .mp4. Se aplica a cada fotograma y vemos si detecta alguna señal o no.

```
0: 128x224 (no detections), 15.4ms
Speed: 0.7ms preprocess, 15.4ms inference, 0.8ms postprocess per image at shape (1, 3, 128, 224)

0: 128x224 (no detections), 11.3ms
Speed: 0.8ms preprocess, 11.3ms inference, 0.7ms postprocess per image at shape (1, 3, 128, 224)

0: 128x224 (no detections), 11.2ms
Speed: 0.9ms preprocess, 11.2ms inference, 0.6ms postprocess per image at shape (1, 3, 128, 224)

0: 128x224 1 class_2, 10.6ms
Speed: 0.8ms preprocess, 10.6ms inference, 1.8ms postprocess per image at shape (1, 3, 128, 224)

0: 128x224 1 class_2, 10.4ms
Speed: 1.6ms preprocess, 10.4ms inference, 1.5ms postprocess per image at shape (1, 3, 128, 224)

0: 128x224 1 class_2, 8.0ms
Speed: 0.7ms preprocess, 8.0ms inference, 1.2ms postprocess per image at shape (1, 3, 128, 224)
```

```
import cv2
from matplotlib import pyplot as plt

# Replace with your actual YOLO model initialization
# Initialize YOLO model
model = YOLO('/content/runs/detect/train/weights/best.pt') # Adjust path according to your YOLO model

video_path = '/content/drive/MyDrive/Test_video_full.mp4'
cap = cv2.VideoCapture(video_path)

# Check if video file opened successfully
if not cap.isOpened():
    print("Error: No se pudo abrir el archivo de video.")
    exit()

# Define el codec y crea el objeto VideoWriter
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv2.VideoWriter('/content/drive/MyDrive/output_video_full_2.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), 10, (fr

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Perform object detection using YOLO model
    results = model.predict(frame)
    plot = results[0].plot()

    # Convert matplotlib RGB to OpenCV BGR
    plot = cv2.cvtColor(plot, cv2.COLOR_RGB2BGR)
    frame_rgb = cv2.cvtColor(plot, cv2.COLOR_BGR2RGB)

    # Guardar el frame procesada en el archivo de video de salida
    out.write(frame_rgb)

    # Mostrar el frame con la detección (opcional)
    # cv2.imshow('Frame', plot)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar recursos
cap.release()
out.release()
cv2.destroyAllWindows()
```

