



<https://matrixcpmsolutions.com/>

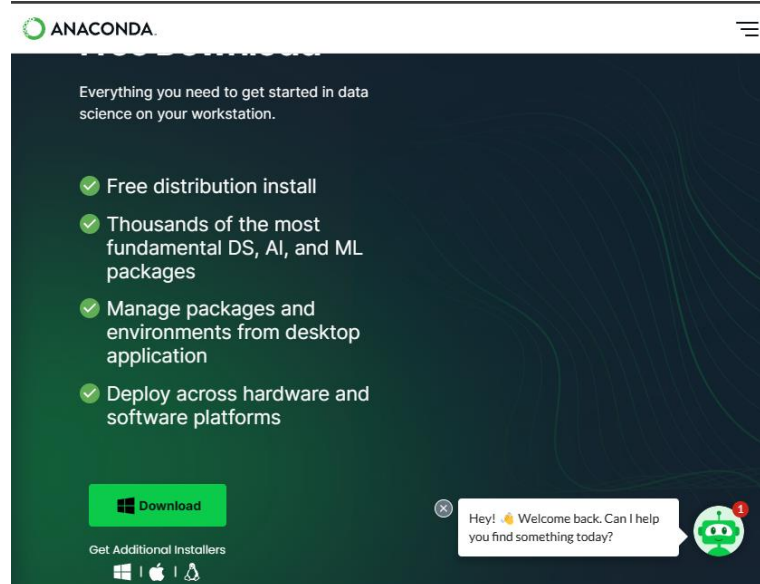
Documentación modelos YOLOv8

26 de febrero del 2024.

1. Pasos iniciales:

1. Instalación de Anaconda:

- Dirigete al sitio web oficial de Anaconda y selecciona la versión adecuada a tu sistema operativo: [Free Download | Anaconda](#)



- Una vez descargado el archivo, debes ejecutarlo para iniciar el proceso de instalación, sigue las instrucciones en pantalla, acepta los términos y condiciones y elige la ubicación de la instalación.
- Verificación terminal de Anaconda: Busca en la barra de tareas "Anaconda Prompt", si no la encuentras es posible que la instalación no haya sido satisfactoria, en ese caso repite el proceso de instalación.



2. Modificación del archivo environment.yml: El archivo environment.yml es un componente esencial en la configuración de tu entorno virtual. Este archivo contiene la lista de todas las dependencias que el proyecto necesita para funcionar correctamente. Para personalizar este entorno a tu sistema específico, necesitarás hacer algunos cambios en este archivo ubicado en la carpeta principal del proyecto:

- Cambio de la ruta prefix: En el archivo environment.yml, encontrarás una línea que comienza con la palabra prefix, normalmente está ubicada en la última línea de configuración. Esta línea especifica la ruta en tu sistema donde se instalará el entorno virtual. Es importante el cambio de la ruta a un espacio de trabajo dedicado para tus proyectos, evita guardar el entorno en una ruta con espacios o donde no tengas permisos de acceso:

Advertencia: La ruta prefix(donde se guardara el entorno virtual en tu equipo) debe ser absoluta, es decir, debe comenzar desde la raíz de tu sistema de archivos.

```
- ultralytics==8.0.220
- youtube-dl==2020.12.2
prefix: C:\Users\jemss\anaconda3\envs\pruebayolo
```

- Cambio del nombre del entorno virtual: Si prefieres, también puedes cambiar el nombre del entorno virtual. Para hacerlo, simplemente reescribe el environment.yml en la línea donde aparece "name", que suele ser la primera línea del archivo, con el nuevo nombre que desees para el entorno. Recuerda que este nombre debe ser único entre tus entornos virtuales para evitar confusiones en la activación.

```
C: > Users > matrix > pruebayolo > proyecto_yolo > environmentym
Click here to ask Blackbox to help you code faster
1 name: pruebayolo
2 channels:
3   - pytorch
4   - defaults
5 dependencies:
6   - blas=1.0
7   - brotli-python=1.0.9
8   - bzip2=1.0.8
9   - ca-certificates=2023.08.22
10  - certifi=2023.11.17
```

3. Iniciando el proceso con Anaconda Prompt: El primer paso para configurar el entorno del proyecto es abrir el Anaconda Prompt. Este terminal especial, que se incluye con la instalación de Anaconda, te permite interactuar directamente con tus entornos virtuales. Una vez que hayas abierto Anaconda Prompt, necesitarás localizar la ruta de la carpeta principal del proyecto donde se encuentra environment.yml que en el paso anterior se ha modificado. Para recrear el entorno virtual, deberás ejecutar el siguiente comando en Anaconda

Prompt, asegurándote de reemplazar <ruta/del/directorio> con la ruta exacta al archivo environment.yml en tu sistema:

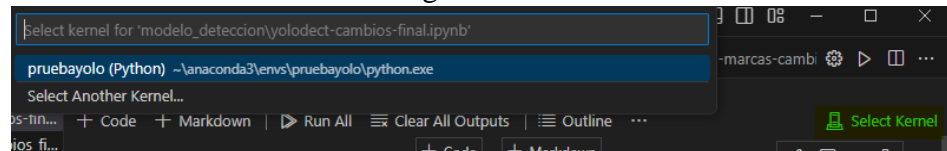
```
(base) C:\Users\jemss>conda create -f ruta/del/directorio/environment.yml
```

- Una vez que hayas recreado el entorno virtual, el siguiente paso es activarlo. La activación del entorno es un proceso que configura tu terminal para que cualquier paquete que instales o cualquier script que ejecutes se haga dentro del entorno virtual, en lugar de hacerlo en tu sistema operativo principal. Para ello debes escribir el siguiente comando en la terminal Anaconda Prompt, reemplazando <nombre_del_entorno> con el nombre de tu entorno, recuerda que el nombre es el que está en el archivo environment.yml:

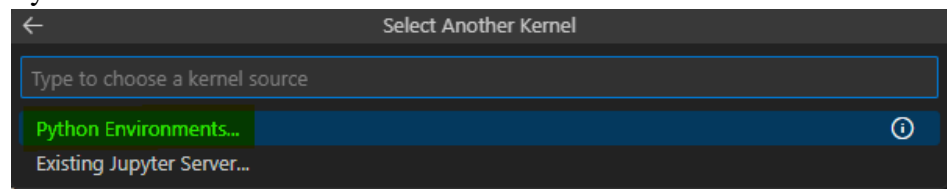
```
(base) C:\Users\jemss>conda activate <nombre_del_entorno>
```

2. Preparación del entorno para la ejecución de scripts de producción: Antes de poder ejecutar los scripts de producción, necesitarás seleccionar el entorno virtual adecuado. Para hacerlo, sigue estos pasos:

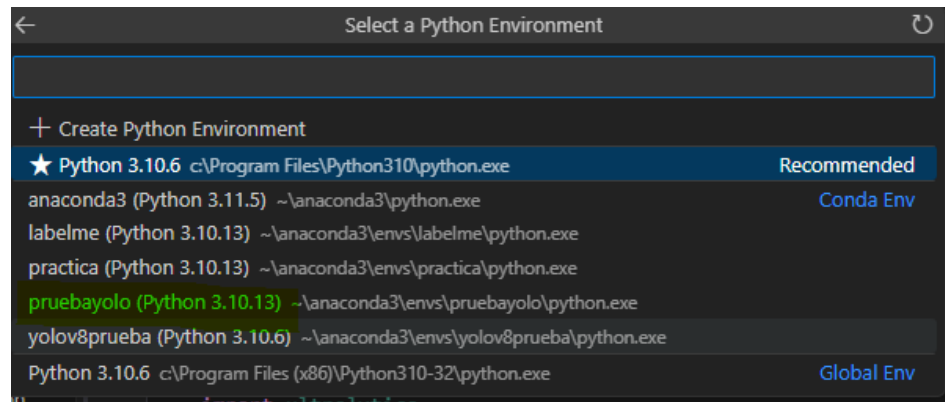
- Ubica el cursor del ratón sobre el identificador Select Kernel y haz clic en él. Esto abrirá una ventana emergente.



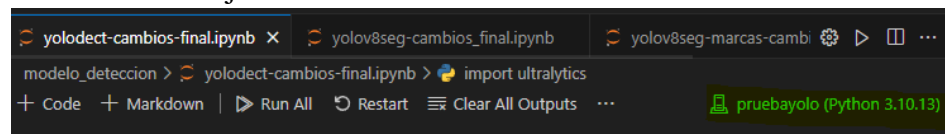
- En esta ventana, deberías ver el nombre de tu entorno virtual. Si no lo ves, no te preocupes. Simplemente haz clic en Select Another Kernel, luego en Python Environments.



- Ahora deberías ver una lista de todos tus entornos virtuales. Busca el que tenga el nombre del entorno virtual y haz clic en él para seleccionarlo.

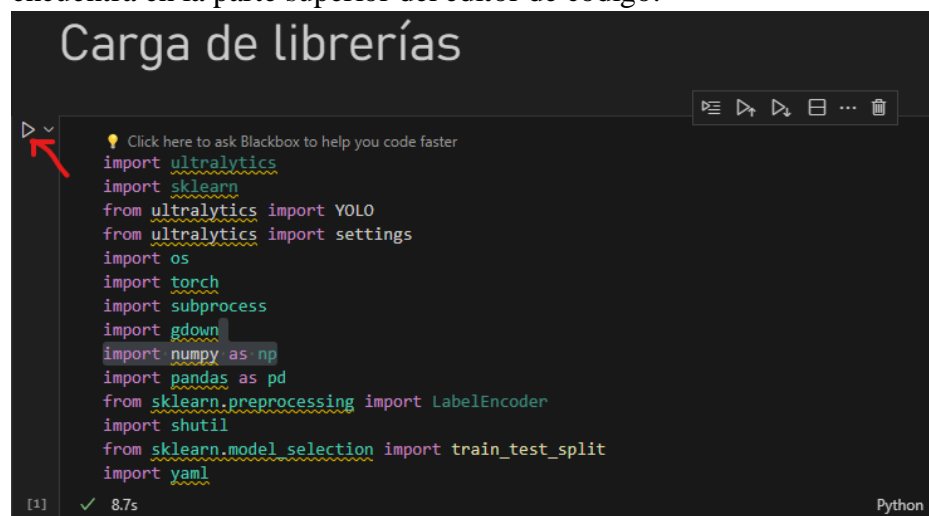


- Se debe ver reflejado de esta manera:



3. Puesta en marcha del proyecto: Es esencial que todas las librerías que se utilizan en el proyecto estén en funcionamiento. Estas librerías proporcionan diversas funcionalidades que son cruciales para la ejecución de los scripts.

- Primero, asegúrate de que tu entorno virtual está activo. Si no es así, actívalo siguiendo los pasos proporcionados anteriormente.
- Localiza el script o el apartado “Carga de librerías” y haz clic en el botón de ejecución. Este botón generalmente se representa con una flecha y se encuentra en la parte superior del editor de código.



4. Preparación para el etiquetado: Antes de poder utilizar labelme o cualquier otro paquete como labelme2yolo, ultralytics, etc., es necesario que prepares tu entorno para su correcto funcionamiento. Estos paquetes son herramientas esenciales que te

permitirán realizar los procesos de manera eficiente. Para preparar tu entorno, sigue estos pasos:

- Ubícate en el apartado “Instalación de herramientas de etiquetado” debes ejecutar la función específica para instalar los paquetes que no se encuentren ya en tu equipo. Esta función revisará tu sistema y determinará qué paquetes necesitas instalar.
- Es importante tener en cuenta que no es necesario que instales los paquetes cada vez que ingreses y quieras ejecutar el script de producción. Una vez que un paquete está instalado en tu sistema, puedes utilizarlo siempre que lo necesites sin tener que reinstalarlo.

Instalación de herramientas de etiquetado

```
Click here to ask Blackbox to help you code faster
# Función para instalar los paquetes

def package_verification(package):
    try:
        output = os.popen('pip list').read()
        return f'{package}' in output
    except Exception as e:
        print(f'Error en la verificación: {e}')
        return False

def package_installation(package_name):
    if package_verification(package_name):
        print(f"El paquete '{package_name}' ya está instalado.")
    else:
        print('Instalando el paquete... ')
        try:
            os.system(f'pip install {package_name}')
            print(f'librería {package_name} instalada correctamente! ')
        except Exception as e:
            print(f'Error en la instalación de {package_name}: {e}')
```

- Al ejecutar la siguiente línea de código, verás un atributo entre paréntesis (). Este atributo es el nombre de la librería que se va a instalar. Una vez que ejecutes esta línea de código, el sistema comenzará a instalar la librería y te mostrará un mensaje sobre el estado de la instalación. Si te encuentras con una función llamada `package_installation`, por favor, asegúrate de ejecutar la celda en la que se encuentra. Es importante recordar que, una vez que una librería se ha instalado en tu sistema, no es necesario que la reinstales cada vez que quieras utilizar el script de producción.

```
Click here to ask Blackbox to help you code faster
package_installation('labelme') # Instalación de la herramienta de etiquetado Labelme

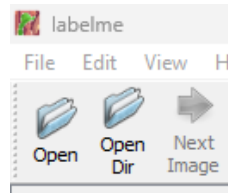
[22] Python

... El paquete 'labelme' ya está instalado.
```

5. Inicio del proceso de etiquetado: Al ejecutar la siguiente celda o script podrá abrir la herramienta `labelme` para iniciar el proceso de etiquetado:

```
Click here to ask Blackbox to help you code faster
# Abrir la herramienta para el proceso de etiquetado
try:
    subprocess.check_call(["labelme"])
except Exception as e:
    print(f'No fue posible abrir labelme: {e}')
```

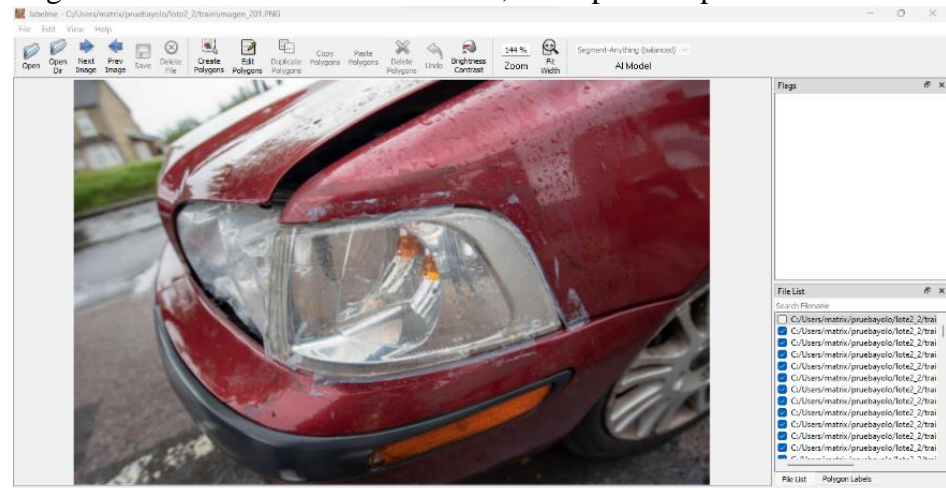
- Apertura del directorio de imágenes: En la pantalla principal de Labelme, dirígete a la esquina superior izquierda y haz clic en el ícono “Open Dir”. Esto abrirá un explorador de archivos.



- Localización de la carpeta de imágenes para el entrenamiento: Necesitarás localizar la carpeta donde se albergarán las imágenes que desees utilizar. Esta carpeta se encuentra dentro de la carpeta principal del proyecto. Dependiendo del modelo que estés utilizando, la ruta a esta carpeta será diferente:

- Para el modelo de marcas, la ruta es:
../..../proyecto_yolo/dist/marcas_global.
- Para el modelo de detección y segmentación, la ruta es:
../..../proyecto_yolo/dist/lote2_2.

- Navega a través de tu sistema de archivos hasta encontrar la carpeta correspondiente. Una vez que la hayas identificado, selecciónala y haz clic en ‘Abrir’. De este modo, todas las imágenes contenidas en la carpeta se cargarán automáticamente en Labelme, listas para ser procesadas.



Nota: Si prefieres no utilizar ninguna de las carpetas predefinidas para almacenar tus imágenes de entrenamiento, tienes la opción de crear una nueva, es indispensable que el nombre no tenga espacios. Para hacerlo, sigue estos pasos:

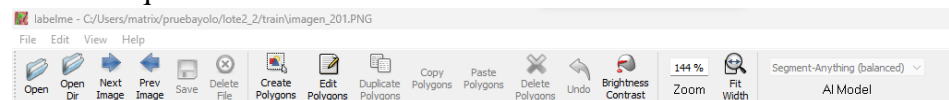
- Navega hasta la ruta ../../proyecto_yolo/dist en tu sistema de archivos. Esta es la carpeta principal donde se almacenan los datos estáticos del proyecto.
- Dentro de esta carpeta, crea una nueva con el nombre que desees. Este será el lugar donde almacenes las imágenes que desees utilizar para el entrenamiento.
- Una vez que hayas creado la carpeta, necesitarás modificar una parte del script ubicado en el apartado “Carga de rutas”. En este script, en la variable señalada debes especificar el nombre de la carpeta creada. Asegúrate de escribir el nombre de tu nueva carpeta entre comillas.

Carga rutas

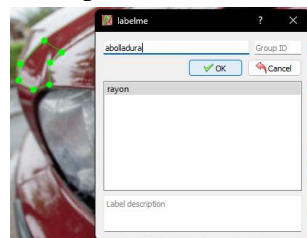
```
current_folder= os.getcwd() # Obtener ubicación actual
main_folder= os.path.abspath(os.path.join(current_folder, "../../..")) # Carpeta main del proyecto
main_staticos= os.path.abspath(os.path.join(main_folder)+'\\dist' # Carpeta main de los archivos estaticos
main_production= os.path.abspath(main_folder) +'\\src' # Carpeta main de los archivos listos para producción
dir_prueba= os.path.abspath(main_folder) + "\\src\\assets" # Carpeta de imagenes prueba para el proceso de
predecir

→ training_structure="lote2_2" # Indica el nombre de la carpeta que tiene los archivos para el entrenamiento (image,
json)
dir_train= os.path.abspath(main_staticos) + f'\\{training_structure}'
```

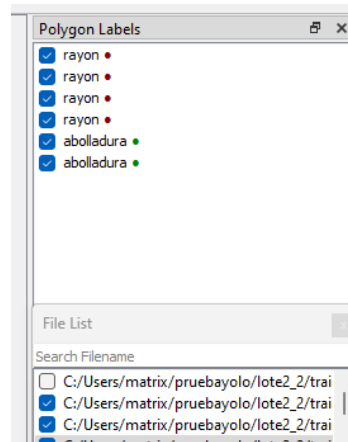
- Para etiquetar las áreas de interés en tus imágenes, utiliza la función “Create Polygons” que se encuentra en el menú superior de Labelme. Con esta herramienta, puedes dibujar polígonos alrededor de las áreas que desees etiquetar.



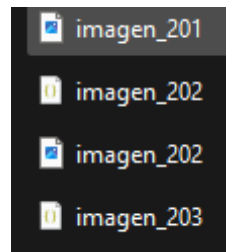
- Después de haber delineado una región con el polígono, se te pedirá que ingreses la clase a la que pertenece esa región. Ingresa la clase correspondiente en el cuadro que aparece.



- Las clases que hayas definido para cada imagen se mostrarán en el panel derecho de Labelme, justo encima de la lista de imágenes.



- Repite este proceso de etiquetado para todas las imágenes en tu carpeta de datos. A medida que vayas etiquetando las imágenes, Labelme generará automáticamente una estructura de archivos que incluye las imágenes y sus archivos JSON correspondientes. El contenido de la carpeta debe verse de esta forma:



6. Carga de direcciones: En este paso, se utilizan rutas relativas para localizar los archivos necesarios para el proyecto. Las rutas relativas son direcciones que hacen referencia a la ubicación de un archivo en relación con la ubicación actual, en lugar de su ubicación absoluta en el sistema de archivos.
 - Es importante destacar que siempre debes ejecutar esta celda cuando cierres el script. Al igual que el apartado “Carga de librerías”, la “Carga de rutas” es un paso esencial que debe realizarse cada vez que inicies el proyecto. Esto asegura que todas las rutas y librerías estén actualizadas y listas para usar.

Carga rutas

```
Click here to ask Blackbox to help you code faster
current_folder= os.getcwd() # Obtener ubicación actual
main_folder= os.path.abspath(os.path.join(current_folder, "../..")) # Carpeta main del proyecto
main_staticos= os.path.abspath(os.path.join(main_folder)+"\\dist") # Carpeta main de los archivos
estaticos
main_production= os.path.abspath(main_folder) +'\\src' # Carpeta main de los archivos listos para
producción
dir_prueba= os.path.abspath(main_folder) + "\\src\\assets" # Carpeta de imagenes prueba para el proceso
de predecir

training_structure="lote2_2" # Indica el nombre de la carpeta que tiene los archivos para el
entrenamiento (image, json)
dir_train= os.path.abspath(main_staticos) + f'\\{training_structure}'
```

[12] Python

7. Configuración de la carpeta para los procesos de entrenamiento y validación: Este paso implica preparar tus datos para los procesos de entrenamiento y validación. La función que se proporciona a continuación dividirá automáticamente la carpeta que contiene tus datos actualizados (es decir, la carpeta que has procesado a través de Labelme y que contiene tanto las imágenes como sus archivos .json correspondientes) en dos subcarpetas: una para el entrenamiento (que contendrá el 80% de los datos) y otra para la validación (que contendrá el 20% restante):

- Es importante destacar que, si ya tienes algunas imágenes seleccionadas para el entrenamiento o la validación, puedes realizar la división manual de los datos. Si prefieres hacerlo de esta manera, debes dividir la carpeta que contiene tus archivos de entrenamiento (la carpeta que contiene los archivos .jpg y .json) en dos subcarpetas: una llamada “train” para los datos de entrenamiento y otra llamada “val” para los datos de validación, recuerda que cada imagen tiene un .json asignado, por lo tanto en la distribución debe ir con su .json correspondiente. Es recomendable utilizar esta nomenclatura para mantener el correcto funcionamiento. Para esta opción evita ejecutar la función de división automática y simplemente pasa al siguiente paso, que es la “Conversión a formato YOLO”.
- *Nota:* Una solución más simple implica que antes del proceso de etiquetado (utilizar la herramienta labelme) separe las imágenes que quiere para entrenamiento y las que desea para validación en la ruta específica para el procesamiento de imágenes (../..proyecto_yolo/dist/), realice el proceso de etiquetado, y se dirija al apartado “conversión a formato YOLO”.

Configuración de rutas de entrenamiento

```
Click here to ask Blackbox to help you code faster
# Función para separar la carpeta global de todos los datos en un 80% entrenamiento y 20% validación.
Utilice esta función solo si no ha ordenado los datos en una carpeta para entrenamiento y otra para
validación.

def file_separation(orig_folder):
    file_list = os.listdir(orig_folder)
    train, val = train_test_split(file_list, test_size=0.2, random_state=100)
    if not os.path.exists("train"):
        train_path = os.path.join(orig_folder, "train")
        os.makedirs(train_path, exist_ok=True)
        move_files(train, orig_folder, train_path)

    if not os.path.exists("val"):
        val_path = os.path.join(orig_folder, "val")
        os.makedirs(val_path, exist_ok=True)
        move_files(val, orig_folder, val_path)

    return train_path, val_path

def move_files(file_list, orig_folder, dest_folder):
    for file in file_list:
        if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif')):
            orig_image_path = os.path.join(orig_folder, file)
            json_name = os.path.splitext(file)[0] + ".json"
```

8. Procedimiento para el cambio de formato JSON a YOLO: Este es un paso crucial en la preparación de tus datos para el entrenamiento, validación y predicción. Deberás ejecutar el siguiente script que convertirá tus archivos de formato .json a formato YOLO, formato reconocido por Ultralytics, la biblioteca que estás utilizando para manejar tus tareas de aprendizaje automático.

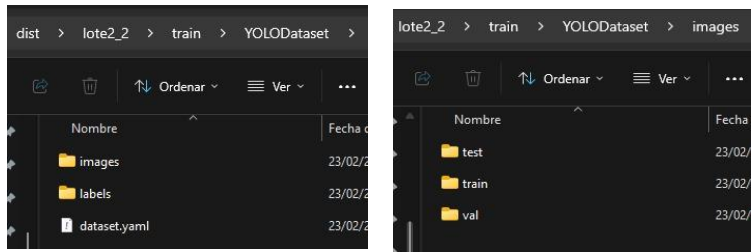
Conversión a formato YOLO

```
Click here to ask Blackbox to help you code faster
# Función para convertir la carpeta de entrenamiento y validación en formato YOLO

def convert_to_yolo_format(training_directory, validation_directory):
    try:
        subprocess.check_call(["labelme2yolo", "--json_dir", training_directory])
        print(f'Conversión completada para la carpeta {training_directory}')
        subprocess.check_call(["labelme2yolo", "--json_dir", validation_directory])
        print(f'Conversión completada para la carpeta {validation_directory}')
    except subprocess.CalledProcessError as e:
        print(f'Error al ejecutar función labelme2yolo: {e}')

convert_to_yolo_format(train_path, val_path)
```

- Como resultado de este proceso, se crearán dos carpetas con el nombre YOLODataset. Una de estas carpetas será para los datos de entrenamiento (train) y la otra para los datos de validación (val). Se proporciona el siguiente set de funciones para ordenar el contenido de la carpeta YOLODataset debido a que tiene una estructura compleja que incluye varias subcarpetas y un archivo dataset.yaml.



- La primera función que debes ejecutar se encargará de ordenar el contenido de esta carpeta, descomprimiendo las subcarpetas y asegurándose de que el contenido de las carpetas principales corresponda al tipo de nombre.

```

Click here to ask Blackbox to help you code faster
# Organización de la estructura del repositorio YOLODataset (resultado de la conversión .json a formato YOLO) creado por defecto

def reorganization(path):
    try:
        new_path = os.path.abspath(os.path.join(path, "YOLODataset"))
        main_folders = os.listdir(new_path)
        for folder_instance in main_folders:
            secondary_folder_path = os.path.join(new_path, folder_instance)
            if os.path.isdir(secondary_folder_path):
                secondary_folders = os.listdir(secondary_folder_path)
                for secondary_folder in secondary_folders:
                    subfolder_secondary_path = os.path.join(secondary_folder_path, secondary_folder)
                    try:
                        if os.listdir(subfolder_secondary_path) != []:
                            for file in os.listdir(subfolder_secondary_path):
                                orig_path = os.path.join(subfolder_secondary_path, file)
                                dest_path = os.path.abspath(os.path.join(subfolder_secondary_path, "../"))
                                new_dest_path = os.path.join(dest_path, file)
                                shutil.move(orig_path, new_dest_path)
                                os.rmdir(subfolder_secondary_path)
                            else:
                                os.rmdir(subfolder_secondary_path)
                    except NotADirectoryError as e:
                        print(f'Error: the element is not a folder: {e}')
        return new_path

```

- La segunda función que debes ejecutar se encargará de mover la carpeta modificada a una nueva ubicación en el directorio de los archivos estáticos (../proyecto_yolo/dist/). Dependiendo del modelo que estés utilizando, la carpeta recibirá el nombre data_afectacion (para el modelo de segmentación y detección) o data_marcas (para el modelo de marcas). Esto se hace para mantener una estructura ordenada y asegurar que la carpeta donde se realizó la conversión a formato YOLO esté destinada solo para subir archivos.

```

Click here to ask Blackbox to help you code faster
# Función para mover las carpetas yolo

def acomodar_rutas_dataset(main_staticos, dataset_train, dataset_val):
    try:
        if not os.path.exists("data_afectacion"):
            try:
                paths_segmentation= os.path.join(main_staticos, "data_afectacion")
                os.makedirs(paths_segmentation, exist_ok=True)
                if os.listdir(paths_segmentation) == []:
                    shutil.move(dataset_train, os.path.join(paths_segmentation, "YOLODataset_train"))
                    shutil.move(dataset_val, os.path.join(paths_segmentation, "YOLODataset_val"))
                    print(f'Carpetas movidas exitosamente a {paths_segmentation}')
                else:
                    print(f'Las carpetas ya se encuentran en {paths_segmentation}')
            except OSError as e:
                print(f'Error al intentar crear la carpeta en {paths_segmentation}: {e}')
        return paths_segmentation
    except TypeError as e:
        print(f'Tipo de dato incorrecto para el argumento: {e}')

detect_routes= acomodar_rutas_dataset(main_staticos, yolo_train_data, yolo_val_data)

```

- La última función que debes ejecutar se utiliza para verificar y realizar cambios en el archivo dataset.yaml de cada carpeta YOLODataset. El objetivo de esta función es evitar que las clases queden separadas y dejar solo un archivo de configuración (dataset.yaml).

```

Click here to ask Blackbox to help you code faster
# Set de funciones para ordenar la estructura de los dataset individuales.

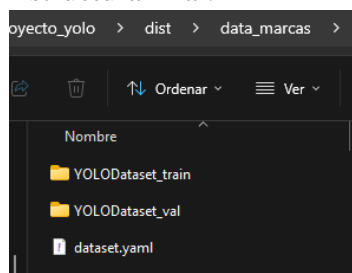
def arrange_dataset_paths(train, val):
    destination = os.path.abspath(os.path.join(train, "../.."))
    shutil.move(train, destination)
    if "dataset.yaml" in os.listdir(os.path.join(val, "../..")):
        os.remove(val)

# Función para reescribir el dataset
def rewrite_dataset(dir_train_converted, combined_list):
    with open(dir_train_converted, 'w') as new_dataset:
        new_dataset.write(
            f'train: {os.path.abspath(os.path.join(dir_train_converted, "../.."))}\n'
            f'val: {os.path.abspath(os.path.join(dir_train_converted, "../.."))}\YOLODataset_val\n'
            'test: \n'
            f'nc: {len(combined_list)}\n'
            f'names: {combined_list}\n'
        )

# Función para combinar los dos dataset generados
def total_labels(list1, list2, update_dataset_train):
    set1=set(list1)
    set2=set(list2)
    union_without_repeating = set1.union(set2)
    combined_list = list(union_without_repeating)
    rewrite_dataset(update_dataset_train, combined_list)

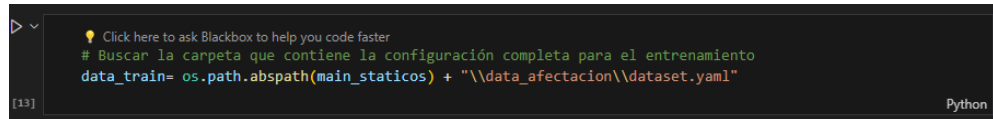
```

Estructura final:



- Por último, ejecutarás una celda que proporcionará la ruta de la modificación anterior. Esta ruta será la que compone la estructura final que se utilizará para los procesos de entrenamiento, validación y predicción.

Advertencia sobre la ejecución de funciones: Es importante tener en cuenta que el conjunto de funciones mencionadas anteriormente solo debe ejecutarse cuando necesites ordenar o actualizar tu conjunto de datos. Estas funciones son responsables de preparar y organizar tus datos para el entrenamiento y la validación, y no necesitan ser ejecutadas a menos que estés realizando cambios en tus datos. En su lugar, solo debes ejecutar la siguiente celda que está diseñada para que el sistema reconozca la ruta donde se encuentra la configuración que ya has realizado.



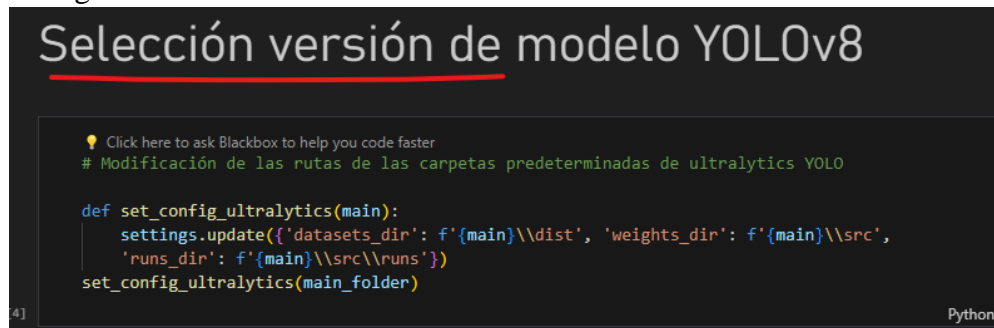
```
> Click here to ask Blackbox to help you code faster
# Buscar la carpeta que contiene la configuración completa para el entrenamiento
data_train= os.path.abspath(main_staticos) + "\\data_afectacion\\dataset.yaml"
```

[13] Python

9. Selección modelo YOLOV8: En la sección “Selección versión de modelo YOLOv8”, tendrás la opción de elegir entre dos tipos de modelos para los modos de entrenamiento, validación y predicción.

Advertencia: Si no ejecutas las celdas que te permiten utilizar tanto una versión del modelo original como un modelo pre-entrenado cada vez que utilizas el script de producción, más adelante en los modos no podrás elegir con qué modelo deseas trabajar.

- Como primer paso, es necesario ejecutar esta celda que establece las rutas donde se guardarán las carpetas predefinidas cuando se realice alguno de los tres modos. Solo es necesario ejecutarla una vez para establecer la configuración.



```
Selección versión de modelo YOLOv8

Click here to ask Blackbox to help you code faster
# Modificación de las rutas de las carpetas predeterminadas de ultralytics YOLO

def set_config_ultralytics(main):
    settings.update({'datasets_dir': f'{main}\\dist', 'weights_dir': f'{main}\\src',
                    'runs_dir': f'{main}\\src\\runs'})
    set_config_ultralytics(main_folder)
```

4] Python

- El modelo original:
 - Incluye varias versiones que puedes instalar según tu elección. Para hacerlo, ejecuta la siguiente celda. Aparecerá un cuadro de diálogo en el que deberás especificar el identificador correspondiente a la versión que deseas utilizar.

Identificadores para las variantes del modelo YOLOv8:

Modelo original

Tabla de variantes en la Segmentación de instancias con YOLOv8

Modelo	Nombres de archivo	Identificador
YOLOv8-seg	<u>yolov8n-seg.pt</u> yolov8s-seg.pt yolov8m-seg.pt yolov8l-seg.pt yolov8x-seg.pt	n, s, m, l, x

Click here to ask Blackbox to help you code faster

```
package_installation('gdown')
```

Python

Celda de ejecución:

Click here to ask Blackbox to help you code faster

```
package_installation('gdown') # Librería que le permitira descargar las versiones de YOLOv8
```

Python

```
def get_model_version():
    available_versions= ['n', 's', 'm', 'l', 'x'] # yolov8n.pt, yolov8s.pt, yolov8m.pt, yolov8l.pt, yolov8x.pt
    control= True
    while control:
        version_selection= input("Ingrese la versión del modelo que desea utilizar, entre las disponibles están: n, s, m, l, x. Advertencia: Si no desea instalar ninguna ingrese la palabra 'salir'!")
        if version_selection in available_versions:
            id_version= f'yolov8{version_selection}.pt'
            return id_version
        elif version_selection == "salir":
            control=False
            return None
        else:
```

- Examine la tabla de rendimiento proporcionada para comprender qué características tiene cada versión; están proporcionadas de menor a mayor índice:
 - Tarea de detención:

Rendimiento

[Detección \(COCO\)](#)
[Detección \(Imágenes abiertas V7\)](#)
[Segmentación \(COCO\)](#)
[Clasificación \(ImageNet\)](#)
[Pose \(COCO\)](#)

Consulta los [Documentos de Detección](#) para ver ejemplos de uso con estos modelos entrenados en COCO, que incluyen 80 clases preentrenadas.

Modelo	tamaño (píxeles)	mAPval 50-95	Velocidad CPU ONNX (ms)	Velocidad A100 TensorRT (ms)	parámetros (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- Tarea de segmentación:

Rendimiento							
Detección (COCO) Detección (Imágenes abiertas V7) Segmentación (COCO) Clasificación (ImageNet) Pose (COCO)							
Consulta Segmentation Docs para ver ejemplos de uso con estos modelos entrenados en COCO, que incluyen 80 clases preentrenadas.							
Modelo	tamaño (píxeles)	mAPbox 50-95	mAPmask 50-95	Velocidad CPU ONNX (ms)	Velocidad A100 TensorRT (ms)	parámetros (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

- El modelo pre-entrenado:

- Como se mencionó anteriormente, para utilizar el modelo pre-entrenado, es necesario haber completado el modo de entrenamiento con un modelo original. De lo contrario, aparecerá un mensaje de advertencia indicando que no existen experimentos.
- Si has seguido el proceso de entrenamiento correctamente, la función seleccionará automáticamente el último experimento de entrenamiento disponible.
- Es crucial llevar un seguimiento de los entrenamientos en la carpeta de destino “.../.../proyecto_yolo/src/runs” para asegurarte de tener acceso a los modelos pre-entrenados cuando los necesites.

Modelo entrenado

```

Click here to ask Blackbox to help you code faster
# Función para obtener el último entrenamiento

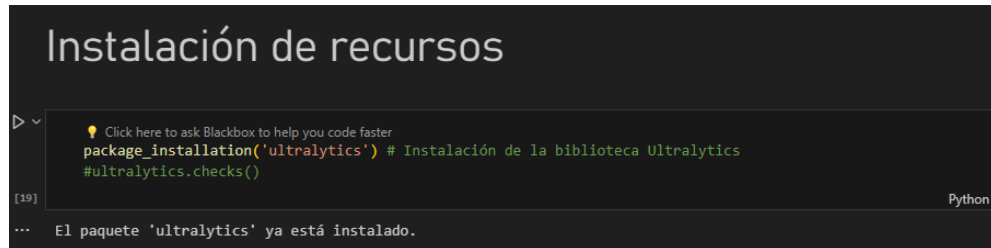
def get_last_training(dir_src):
    try:
        yolo_model_folders= os.listdir(dir_src)
        for folder_name in yolo_model_folders:
            if folder_name == "deteccion":
                base_path = os.path.abspath(os.path.join(dir_src, folder_name))
                files= os.listdir(base_path)
                last_experiment = sorted(files)[-1]
                return os.path.join(base_path, last_experiment, "weights", "best.pt")
        return None
    except FileNotFoundError as e:
        print(f'Error: {e}')

version= "srcv3"
dir_model_version=os.path.abspath(os.path.join(main_production, version)+ "\\modelo_deteccion")
model_train= get_last_training(dir_model_version)

if model_train:
    print(f'Los pesos del experimento que se van a utilizar son {model_train}')
else:
    print(f'No existe ningún experimento de entrenamiento en la carpeta {dir_model_version}')

```

10. Última Configuración: Para poder utilizar los modos de YOLOv8, sigue estos pasos en la sección de “Instalación de Recursos”:

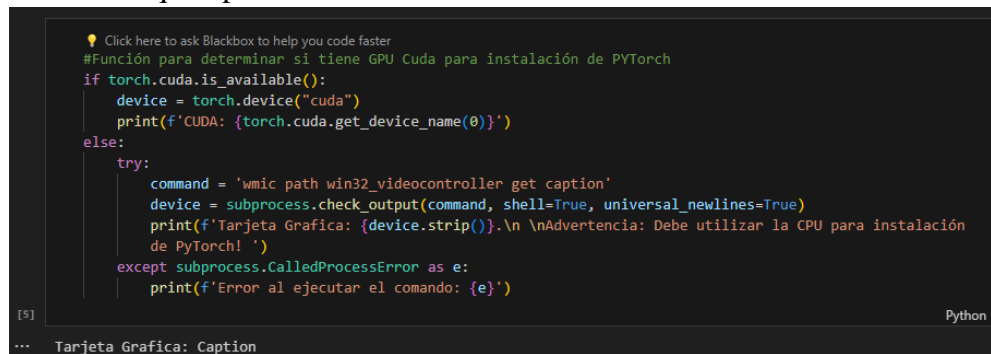


```
Click here to ask Blackbox to help you code faster
package_installation('ultralytics') # Instalación de la biblioteca Ultralytics
#ultralytics.checks()

[19] Python

... El paquete 'ultralytics' ya está instalado.
```

- Ejecuta la segunda celda para verificar la compatibilidad de tu sistema con la instalación de PyTorch, el marco de trabajo de aprendizaje automático. Esta función te proporcionará información sobre la tarjeta gráfica de tu sistema y te indicará qué opción debes utilizar durante la instalación.

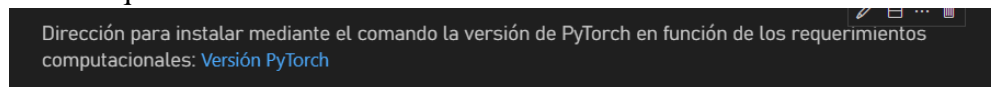


```
Click here to ask Blackbox to help you code faster
#Función para determinar si tiene GPU Cuda para instalación de PyTorch
def torch_cuda_is_available():
    if torch.cuda.is_available():
        device = torch.device("cuda")
        print(f'CUDA: {torch.cuda.get_device_name(0)}')
    else:
        try:
            command = 'wmic path win32_videocontroller get caption'
            device = subprocess.check_output(command, shell=True, universal_newlines=True)
            print(f'Tarjeta Grafica: {device.strip()}.\\n \\n Advertencia: Debe utilizar la CPU para instalación de PyTorch! ')
        except subprocess.CalledProcessError as e:
            print(f'Error al ejecutar el comando: {e}')

[5] Python

... Tarjeta Grafica: Caption
```

- Para instalar la versión de PyTorch que corresponde a tus requisitos computacionales, ingrese a la página oficial de PyTorch en la sección “get Started” o diríjase al enlace directo haciendo click en “Versión PyTorch” en la celda que muestra esta información:



```
Dirección para instalar mediante el comando la versión de PyTorch en función de los requerimientos computacionales: Versión PyTorch
```

- Elija las funciones que se ajusten a su sistema informático haciendo clic en la versión correspondiente; estas se mostrarán resaltadas en color naranja después de su selección. Recuerde que en la sección "Run this Command" recibirá una instrucción específica; asegúrese de sustituirla entre comillas en la variable correspondiente.

PyTorch

Get Started Ecosystem Edge Blog Tutorials Docs Resources GitHub

Shortcuts

Prerequisites

Supported Windows Distributions

Python

Package Manager

Installation

Anaconda

pip

Verification

Building from source

Prerequisites

recommended package manager since it installs all dependencies. You can also [install previous versions of PyTorch](#). Note that LibTorch is only available for C++.

NOTE: Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

PyTorch Build	Stable (2.2.0)	Preview (Nightly)
Your OS	Linux	Mac
Package	Conda	Pip
Language	Python	C++ / Java
Compute Platform	CUDA 11.8	CUDA 12.1
Run this Command:	pip3 install torch torchvision torchaudio	

- Contenido de la variable a modificar: Esto forma parte de la configuración, por lo que no es necesario ejecutarlo cada vez que desee iniciar el script.

Dirección para instalar mediante el comando la versión de PyTorch en función de los requerimientos computacionales: [Versión PyTorch](#)

```

Click here to ask Blackbox to help you code faster
command_pytorch = 'pip3 install torch torchvision torchaudio' # Comando de instalación según la version
personalizada a sus requerimientos computacionales
substrings = command_pytorch.split(" ")
try:
    subprocess.check_call(substrings)
except subprocess.CalledProcessError as e:
    print(f'Error al instalar la version de PyTorch: {e}')
  
```

A partir de aquí podrá utilizar los modos de YOLOv8 mencionados (ENTRENAMIENTO, EXPORTACIÓN, VALIDACIÓN Y PREDICCIÓN).

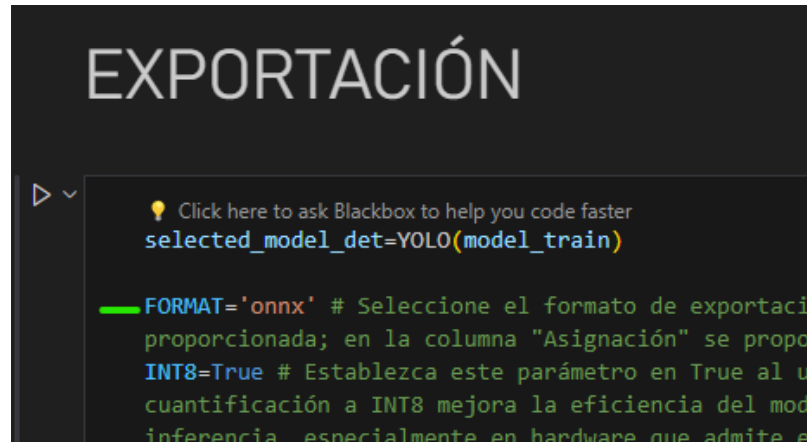
1. Proceso de exportación: En la tabla puedes ver el formato de exportación para que el modelo pueda ser utilizado en diferentes entornos:

Configuración para la exportación

A continuación se muestra una tabla de referencia para exportar un modelo YOLOv8 entrenado en la tarea detección de objetos. Tenga en cuenta que configurar el parámetro de `format` es fundamental para el proceso de exportación. Antes de continuar, asegúrese de verificar y ajustar estos valores a sus requisitos específicos:

Formatos	Asignación	Extensión	Hyperparámetros	Descripción
PyTorch	-	yolov8n.pt	-	Modelo en formato PyTorch
TorchScript	"torchscript"	yolov8n.torchscript	imgsz, optimize	Simplifica la implementación de modelos PyTorch en entornos de producción y aplicaciones eficientes, mejorando la portabilidad y el rendimiento al permitir ejecutar una representación intermedia en entornos sin Python.
ONNX	"onnx"	yolov8n.onnx	imgsz, half, dynamic, simplify, opset	Desarrollado para promover la interoperabilidad, la optimización del hardware y la colaboración entre comunidades, al tiempo que responde a la necesidad de portabilidad de los modelos entre distintos marcos y herramientas de aprendizaje automático.
				Elaborado para promover la interoperabilidad, la optimización del hardware y el despliegue eficiente

- Según la información de la tabla, puede determinar los parámetros requeridos. En la variable FORMAT se debe indicar el valor que se encuentra en la columna "Asignación":



- Cuando encuentres la siguiente línea de comando en la misma celda, configúrala conforme al contenido de la columna "Hyperparámetros"; es decir, entre los paréntesis deben ajustarse los valores indicados.

```
selected_model_det.export(format=FORMAT, imgsz=640, dynamic=False, simplify=False, opset=False)
```

2. Proceso de Predicción:

- Puede emplear diversas fuentes para el proceso de predicción, las cuales están disponibles en la sección siguiente. Si desea obtener información detallada sobre cómo implementarlas, visite el sitio oficial de Ultralytics:

Configuración de fuentes			
Para utilizar múltiples fuentes de datos al realizar predicciones con el modelo, se requiere que se ajuste el parámetro 'source' a sus necesidades, tal como se indica en la siguiente tabla:			
Fuentes	Asignación	Tipo	Notas
image	'image.jpg'	str or Path	Archivo que contiene una única imagen.
URL	'https://ultralytics.com/images/bus.jpg'	str	Dirección que especifica la ubicación de una imagen en la web.
screenshot	'screen'	str	El sistema captura la imagen actualmente visible en la pantalla y la utiliza como entrada para el modelo.
PIL	Image.open('im.jpg')	PIL.Image	Utilizado para cargar imágenes en formato HWC (altura, ancho, canales) con canales RGB (rojo, verde y azul) mediante la biblioteca Python Imaging Library (PIL).
OpenCV	cv2.imread('im.jpg')	np.ndarray	Permite la lectura de una imagen desde un archivo en formato HWC con canales BGR (azul, verde, rojo) utilizando la biblioteca OpenCV, almacenando la imagen como un array de NumPy.

Advertencia: Para los modelos de detección y segmentación, tenga en cuenta que las últimas funciones relacionadas con el cálculo del área solo podrán activarse después de ejecutar la tarea de Predicción.

Función de área para cajas detectadas

Dataframe con las áreas totales

```
Click here to ask Blackbox to help you code faster

def calculate_area(mask):
    sum_area = 0
    if mask is not None:
        for e in mask:
            area = torch.round(e[2] * e[3]).int()
            sum_area += area.sum().item()
        return sum_area

def search_area_by_instance(results, image_instance):
    for i in results:
        if i.path == image_instance:
            return calculate_area(i.bboxes.xywh) if i.bboxes is not None and len(i.bboxes) > 0 else 0

def process_main_detection(dent, scratch, car_results):
    data = {'imagen':[], 'area abolladura':[], 'area rayon':[], 'area carro':[], 'afectacion':[]}
    for image_id, e_dent in enumerate(dent):
```