2 Fine-tuning YOLOv5: "You-Only-Look-Once" (30%)

En esta sección se familiarizará con el modelo open-source YOLO, un sistema de detección de objetos optimizado para ofrecer alta velocidad y buen rendimiento. Este modelo es ampliamente utilizado en la industria para aplicaciones de clasificación y segmentación de imágenes en tiempo real. En particular, utilizaremos la API torch.hub para cargar un checkpoint preentrenado del modelo. Los detalles sobre cómo realizar este procedimiento se entregarán en la ayudantía de la tarea.

Actividad 4

Investigue sobre la arquitectura de los modelos YOLO y explique los siguientes conceptos (para cada caso use solo 5 líneas máximo):

- Diferencia entre frameworks de detección de objetos de una y dos etapas (single-stage vs. two-stage object detection).
- Función de pérdida Complete Intersection Over Union (CIoU) y su uso en la red YOLOv5.
- Función de pérdida Binary Cross Entropy (BCE) y su uso en la red YOLOv5.
- ¿Cómo se calcula la función de pérdida total en YOLOv5?

Actividad 5

Investigue sobre el output de la red YOLOv5 y cómo se traduce el tensor de salida a bounding boxes y detecciones de objetos. Además, explique cómo, en general, se obtiene solamente una detección por objeto, y no varias para todas las regiones donde el objeto está presente. Además, investigue el rol de la augmentación de datos en el entrenamiento de YOLOv5.

Actividad 6

En el notebook, deberán hacer fine-tuning para que una instancia de YOLOv5 pueda realizar segmentaciones de datos satelitales. Para esto, se encuentra implementado la carga del modelo YOLOv5 con los parámetros congelados salvo la cabeza de detección. Se encuentra además el código para cargar y preprocesar el set de datos. Se deben completar las siguientes funciones:

```
import albumentations as A
from yolov5.utils.augmentations import Albumentations

def train(model, dataloader, optimizer, epochs, transforms):
    """
    Dado un modelo, un dataloader, un optimizador, un numero de epocas
    y una composicion de transformaciones
    calcula la funcion de perdida de YOLOv5, realiza el backward pass
    y actualiza los pesos de la red por epoch iteraciones.
    """
    model.train()
    model.model.transform = Albumentations(transform)
    loss = 0
    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

def eval(model, dataloader):
    """
    Dado un modelo y un dataloader, calcula las metricas CIoU, BCE de prediccion
        de
    clase y objectiveness.
    """
    model.eval()
    ciou = 0
    bce_class = 0
```

```
    bce_obj = 0
    print(f"Mean scores: CIoU {ciou}, BCE class: {bce_class}, BCE object: {
        bce_obj}")
```

Actividad 7

Finalmente, realice fine-tuning de dos modelos YOLOv5: uno sin transformaciones de datos adicionales y otro utilizando rotaciones, es decir, aplicando aumento de datos a partir de versiones rotadas de las imágenes originales. ¿Se observa alguna diferencia notable en su rendimiento? ¿Por qué las rotaciones constituyen una buena estrategia de aumento de datos para este problema?

Elija 3 imágenes representativas del set de validación y muestre los bounding boxes encontrados por cada modelo.