Documentación

-Segmentación y etiquetamiento de datos:

Los datos de prueba destinados para la detección de elementos, inicialmente tenía un formato de video, por lo que para poder seccionar y entrenar los datos fue necesario convertir el video a fotogramas por segundo.

Utilizando el programa FFmpeg, se dividió el video original en múltiples fotogramas, que luego fueron utilizados en el software de YoloLabel para etiquetar manualmente los elementos identificados, las etiquetas utilizadas fueron: Vehículos, Construcciones, Vías, Ríos y Minería. Luego de haber etiquetado todos los datos, YoloLabel, devuelve un formato de imagen con un archivo correspondiente de .txt, que corresponde a las coordenadas en las que fueron detectados elementos en la imagen.

-Utilización de las librerías:

keras, numpy, pandas, ultralytics, tensorflow, random, os, matplotlib, sklearn, glob, cv2, shutil, IPython.display, PIL, tqdm, copy.

Para hacer uso de estas librerías se realizaron las correspondientes importaciones y descargas. Las descargas se hicieron bajo el comando: pip install <librería>

-Carga de archivos y división de datos por carpetas

Se crearon dos carpetas principales: images y labels. En images se encuentran los archivos .jpg, y en labels los archivos .txt, que corresponden a las coordenadas de las etiquetas definidas a través de YoloLabel.

Se les asignaron porcentajes de cubrimiento de los archivos a cada carpeta (test, train, val). En este caso se asignó un 70% de los datos a la carpeta de train, 10% a la carpeta de val y 20% a la carpeta de test. Luego se aleatorizó el orden de los archivos, pero de una forma conjunta entre images y labels, esto debido a que los documentos que queden en cada carpeta deben ser coincidenciales con su contraparte, es decir, cada archivo de images debe tener su correspondiente en labels.

-Creación de un archivo config

Se creó un archivo config.yaml, en el que se declararon las etiquetas utilizadas para la detección de objetos. Estas etiquetas fueron: ['Vehiculos', 'Construcciones', 'Vias', 'Rios', 'Mineria'].

-Funciones para la visualización del etiquetado de imágenes

Estas funciones se encuentran comentadas, para poder usarlas no hace falta más que descomentarlas y correr el archivo.

Se definieron ciertas funciones para visualizar el etiquetado y detección de elementos de una imagen de acuerdo con las coordenadas establecidas en labels. Para probar esta funcionalidad, se especificó el path al que se desea acceder y se establece aleatoriamente un número para visualizar la imagen, se comprueba que efectivamente este archivo exista y despliega el archivo con una comparación de la imagen plana y su versión con una etiqueta.

-Entrenamiento del modelo de detección de imágenes

Para el entrenamiento del modelo de detección de imágenes, se utilizó la librería se ultralytics. Con la función model.train, se especificó el path del archivo config.yaml, y se determinó un número de 10 epochs, es decir, el número de veces que se van a pasar cada ejemplo de entrenamiento por la red.

Los resultados obtenidos, quedaron registrados en la carpeta runs/detect/train2. Con estos resultados, es posible evaluar el rendimiento del modelo.

-Evaluación de resultados

Para evaluar y analizar los resultados del entrenamiento del modelo, se definieron las funciones de evaluación de las métricas de entrenamiento, validación y prueba. Los resultados obtenidos de estas validaciones quedaron guardados en: runs/detect/val, runs/detect/val2, runs/detect/val3, respectivamente.

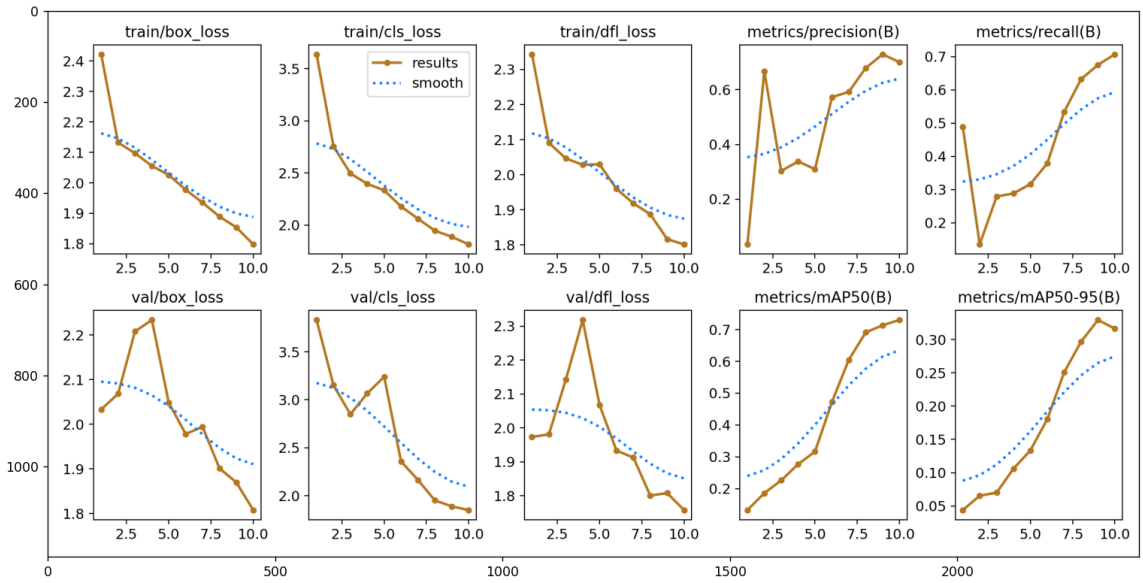
-Visualización de resultados

-Sí se desea visualizar los datos en formato de video con las etiquetas de las predicciones hechas por el modelo, tan solo es necesario correr el archivo video.py y este retornará un output video de los resultados.

-Para obtener un reporte de las secciones del video con más elementos detectados, se debe correr el archivo reporte.py que se encarga de realizar las predicciones en base al modelo de cada una de las imágenes, luego recopila estos resultados junto con el número de predicciones y los almacena en un Excel, en donde cada imágenes está relacionada con el número de elementos identificados, para que después, el mismo programa acceda a los resultados tabulados y en intervalos de tres imágenes sume la cantidad de elementos y devuelve el top 50 de rangos de imágenes con mas detecciones. Finalmente, se podrán visualizar los 5 rangos con más predicciones en formato de imagen, se desplegarán las imágenes originales junto con las predicciones para cada uno de los 5 rangos.

Los resultados obtenidos pueden ser visualizados gráficamente con las funciones definidas al final del proyecto.

Resultados:



Pérdida de Caja (Box Loss) de Entrenamiento y Validación: Ambas disminuyen, lo cual es positivo y muestra que el modelo está aprendiendo a predecir la ubicación de las cajas delimitadoras de manera más precisa.

Pérdida de Clasificación (Cls Loss) de Entrenamiento y Validación: Esta métrica también está disminuyendo, indicando que el modelo mejora en la clasificación correcta de los objetos dentro de las cajas delimitadoras.

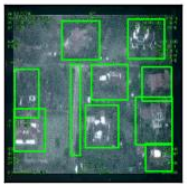
Pérdida de DFL de Entrenamiento y Validación: La pérdida de DFL (Loss Distribution Focal Loss) disminuye en general, señalando que el modelo está aprendiendo adecuadamente tanto la clasificación como la localización.

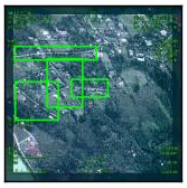
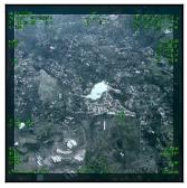
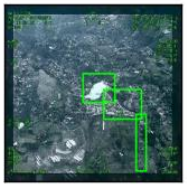
Precisión (Precision): La precisión presenta oscilaciones pero tiende a aumentar, lo cual sugiere que el modelo está mejorando en su capacidad de no etiquetar ejemplos negativos como positivos.

Exhaustividad (Recall): La exhaustividad aumenta, indicando que el modelo se vuelve más efectivo en identificar todas las muestras positivas.

mAP@50: Esta métrica de precisión promedio aumenta, lo que significa que hay más coincidencias entre las cajas delimitadoras predichas por el modelo y las cajas verdaderas.

mAP@50-95: También en aumento, lo que sugiere mejoras en la detección precisa de objetos a varios niveles de exigencia de solapamiento con las cajas verdaderas.

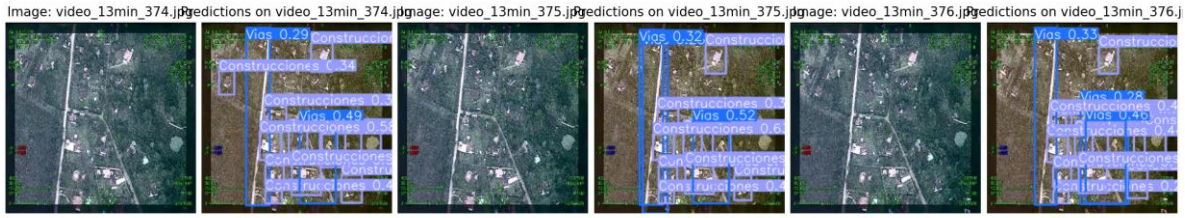




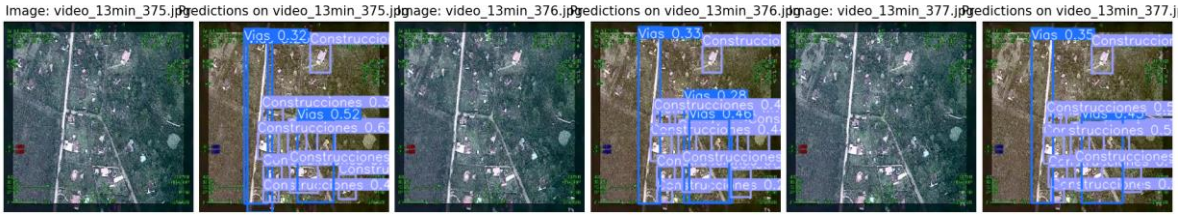
-Entrenamiento nuevo modelo con mayor cantidad de datos:

Este modelo se comporta mejor en la detección de más diversos elementos, ya que reconoce un mayor espectro de elementos en diversas condiciones.

```
                 all        372        754      0.229      0.244      0.224     0.0789

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/10         0G      2.111      2.495      2.185         11        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
                   all        372        754      0.336      0.368       0.31      0.108

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/10         0G      2.074      2.391      2.152         10        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
                   all        372        754      0.371      0.372      0.325      0.113

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/10         0G       2.03      2.297      2.121          8        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
                   all        372        754      0.417       0.39       0.38      0.138

10 epochs completed in 4.827 hours.
Optimizer stripped from runs\detect\train4\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train4\weights\best.pt, 6.2MB
```

Estos fueron los resultados obtenidos al correr el archivo de reporte.py con la mitad de los datos:

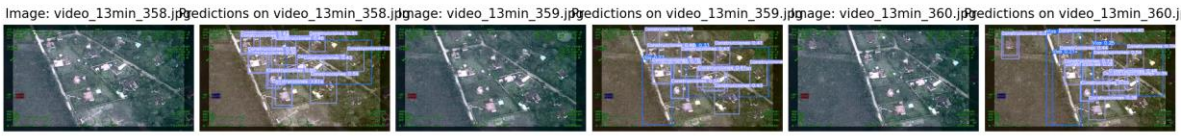


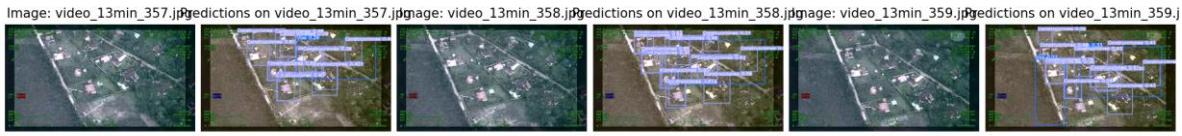


	Start Image	End Image	Total Detections
1099	video_13min_626.jpg	video_13min_629.jpg	40
927	video_13min_374.jpg	video_13min_376.jpg	40
928	video_13min_375.jpg	video_13min_377.jpg	38
926	video_13min_373.jpg	video_13min_375.jpg	38
931	video_13min_380.jpg	video_13min_382.jpg	38
925	video_13min_372.jpg	video_13min_374.jpg	37
929	video_13min_376.jpg	video_13min_380.jpg	37
930	video_13min_377.jpg	video_13min_381.jpg	37
913	video_13min_357.jpg	video_13min_359.jpg	35
920	video_13min_365.jpg	video_13min_369.jpg	34
1124	video_13min_665.jpg	video_13min_668.jpg	32
912	video_13min_356.jpg	video_13min_358.jpg	32
1125	video_13min_667.jpg	video_13min_669.jpg	32
1098	video_13min_624.jpg	video_13min_627.jpg	31
1106	video_13min_639.jpg	video_13min_642.jpg	31
932	video_13min_381.jpg	video_13min_384.jpg	31
1123	video_13min_663.jpg	video_13min_667.jpg	31
1100	video_13min_627.jpg	video_13min_632.jpg	30
919	video_13min_364.jpg	video_13min_367.jpg	30
1107	video_13min_640.jpg	video_13min_643.jpg	30
671	video_13min_011.jpg	video_13min_014.jpg	30
1101	video_13min_629.jpg	video_13min_634.jpg	29
1109	video_13min_643.jpg	video_13min_645.jpg	28

Para generar el archivo de reporte con todos los datos (cerca de 4000 imágenes), el programa tardó 9 minutos, y se obtuvieron los siguientes resultados:

A continuación, se muestran el top 5 de imágenes en rangos de 3 imágenes consecutivas con mayor número de detecciones:





Estos fueron los resultados obtenidos del top 50 de rangos de imágenes con más detecciones:

	Start Image	End Image	Total Detections
2618	video_13min_358.jpg	video_13min_360.jpg	51
2617	video_13min_357.jpg	video_13min_359.jpg	47
3411	video_30min-11to22_360.jpg	video_30min-11to22_362.jpg	46
3423	video_30min-11to22_372.jpg	video_30min-11to22_374.jpg	44
2652	video_13min_392.jpg	video_13min_394.jpg	44
2624	video_13min_364.jpg	video_13min_366.jpg	43
2616	video_13min_356.jpg	video_13min_358.jpg	41
2634	video_13min_374.jpg	video_13min_376.jpg	40
2651	video_13min_391.jpg	video_13min_393.jpg	40
3422	video_30min-11to22_371.jpg	video_30min-11to22_373.jpg	39
2640	video_13min_380.jpg	video_13min_382.jpg	39
2633	video_13min_373.jpg	video_13min_375.jpg	38
2635	video_13min_375.jpg	video_13min_377.jpg	38
2639	video_13min_379.jpg	video_13min_381.jpg	37
2887	video_13min_627.jpg	video_13min_629.jpg	37
3410	video_30min-11to22_359.jpg	video_30min-11to22_361.jpg	36
2632	video_13min_372.jpg	video_13min_374.jpg	35
2619	video_13min_359.jpg	video_13min_361.jpg	34