

Deep Learning en QGIS: De modelos listos a la creación de nuestros propios modelos



....

Aplicación con QGIS, DeepNess, Roboflow y Yolo

Conceptos Fundamentales



Inteligencia Artificial (IA)

Sistemas que realizan tareas que requieren inteligencia humana (aprendizaje, razonamiento).



Machine Learning (ML)

Algoritmos que permiten a las máquinas aprender de los datos sin ser programadas explícitamente.



Deep Learning (DL)

Usa redes neuronales profundas para tareas complejas. Simula el cerebro con nodos (neuronas).



Visión Artificial

Rama clave del ML que permite a las máquinas interpretar imágenes y videos (ej. detección satelital).

Dos etapas para el Deep Learning en SIG

Etapa 1: Usar Modelos Existentes

Nos centramos en el uso inmediato de modelos pre-entrenados directamente en QGIS a través del plugin **DeepNess**. Es el camino rápido para obtener resultados a fin de ingresar a la temática.

Etapa 2: Crear un Modelo Propio

Detallamos el proceso completo: desde la preparación de datos y el etiquetado, hasta el entrenamiento personalizado y la evaluación del modelo.

Etapa 1: Utilizar modelos pre-entrenados



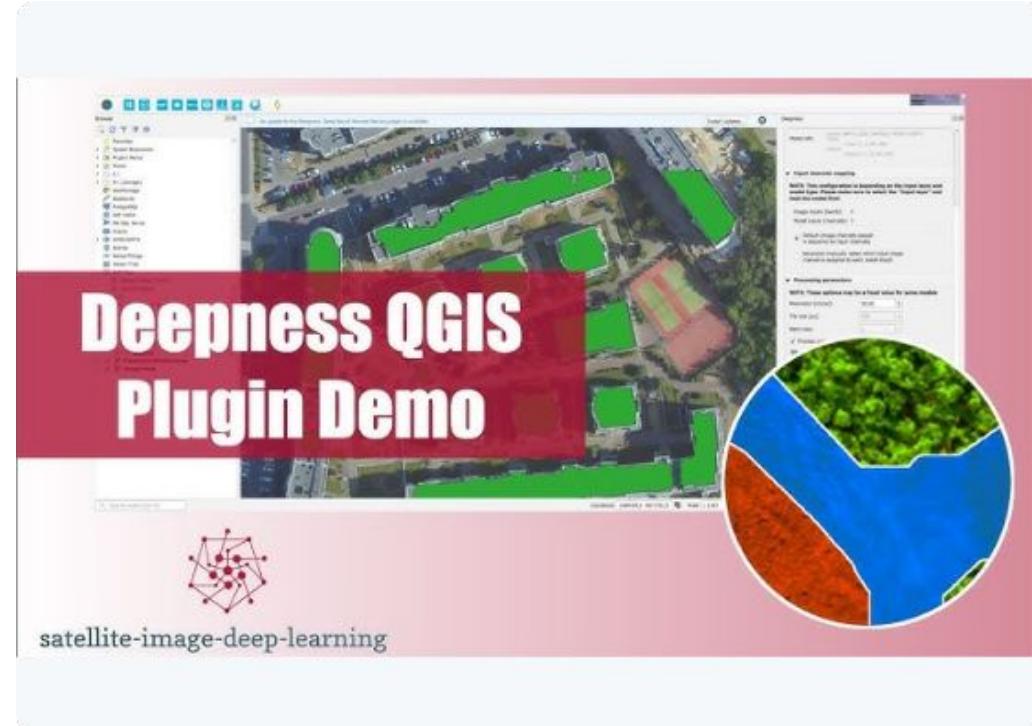
Aplicación inmediata en QGIS

El Plugin DeepNess para QGIS

Instalar este plugin en QGIS con el Administrador de Complementos de QGIS

DeepNess es una herramienta de código abierto diseñada para facilitar el uso de redes neuronales profundas en tareas de teledetección (segmentación y detección) directamente en QGIS.

- Permite procesar cualquier capa ráster.
- Incluye modelos listos para usar.
- Se instala desde el Administrador de complementos de QGIS



<https://qgis-plugin-deepness.readthedocs.io/en/latest/>

Ejemplo de Segmentación con Modelos Existentes

Segmentación Semántica

Consiste en clasificar **cada píxel** de una imagen en una de las clases posibles. DeepNess permite usar modelos pre-entrenados del "Model ZOO" para esta tarea.

Ejemplo:

- **Modelo:** "Land Cover Segmentation"
- **Clases:** Construcciones, espacios verdes, agua y caminos (hay una capa extra con el fondo).
- **Resultado:** Una nueva capa poligonal para cada clase detectada.
- **Utilidad:** Permite calcular áreas aproximadas de las clases, como por ejemplo área de construcciones.

Segmentación semántica: Asigna una **clase** (etiqueta) a cada píxel, identificando *qué* es (ej. "coche", "persona", "fondo"), pero sin distinguir instancias individuales del mismo objeto. Ejemplo: Todos los coches en una imagen se etiquetan como "coche", sin separar uno de otro.

Segmentación de instancias: Más precisa; distingue **instancias individuales** de la misma clase (ej. "coche 1", "coche 2"). Combina detección de objetos con segmentación.

Segmentación panorámica: Une semántica e instancias, etiquetando todo en la imagen (incluyendo fondo) y separando objetos únicos.

Segmentación con Modelos Existentes

Segmentación Semántica con el modelo Land Cover Segmentation

Imagen a procesar

The screenshot shows a GIS application window with the following components:

- Navegador (Toolbar):** Includes icons for Favorites, Spatial Markers, Project Start, and a folder labeled "C:\Users\Usuario\Desktop\IA\charla 18_11_25".
- Capas (Layers):** Shows layers for "OpenStreetMap" and "Google Satellite".
- Imagen a procesar (Processed Image):** A satellite view of a residential neighborhood with a large construction site in the center-right. A river flows along the right side of the image.
- Deepness (Processing Parameters):**
 - Model input 1: Green (band 2)
 - Model input 2: Blue (band 3)
- Processing parameters:**

NOTE: These options may be a fixed value for some models

 - Resolution [cm/px]: 50,00
 - Tile size [px]: 512
 - Batch size: 1
 - Process using local cache
 - Tiles overlap:
 - [%] 15
 - [px] 0
- Segmentation parameters:**

NOTE: Applicable only if a segmentation model is used

 - Argmax (most probable class only)
 - Apply class probability threshold: 0,50
 - Remove small segment areas (dilate/erode size) [px]: 9
- Training data export:**
- Bottom status bar:** Fuentes de datos locales actualizadas, Coordenada -34,859745° -57,901122°, Escala 1:1492, Amplificador 100%, Rotación 0,0 °, Representar, EPSG:4326.

Segmentación con Modelos Existentes

Segmentación Semántica con el modelo Land Cover Segmentation

Imagen procesada. Observar capas vectoriales nuevas.

Nuevas capas vectoriales agregadas por el plugin

Deepness

Model input 1: Green (band 2)
Model input 2: Blue (band 3)

Processing parameters

NOTE: These options may be a fixed value for some models

Resolution [cm/px]: 50,00
Tile size [px]: 512
Batch size: 1
 Process using local cache

Tiles overlap:

[%] 15 [px] 0

Segmentation parameters

NOTE: Applicable only if a segmentation model is used

Argmax (most probable class only)
 Apply class probability threshold: 0,50
 Remove small segment areas (dilate/erode size) [px]: 9

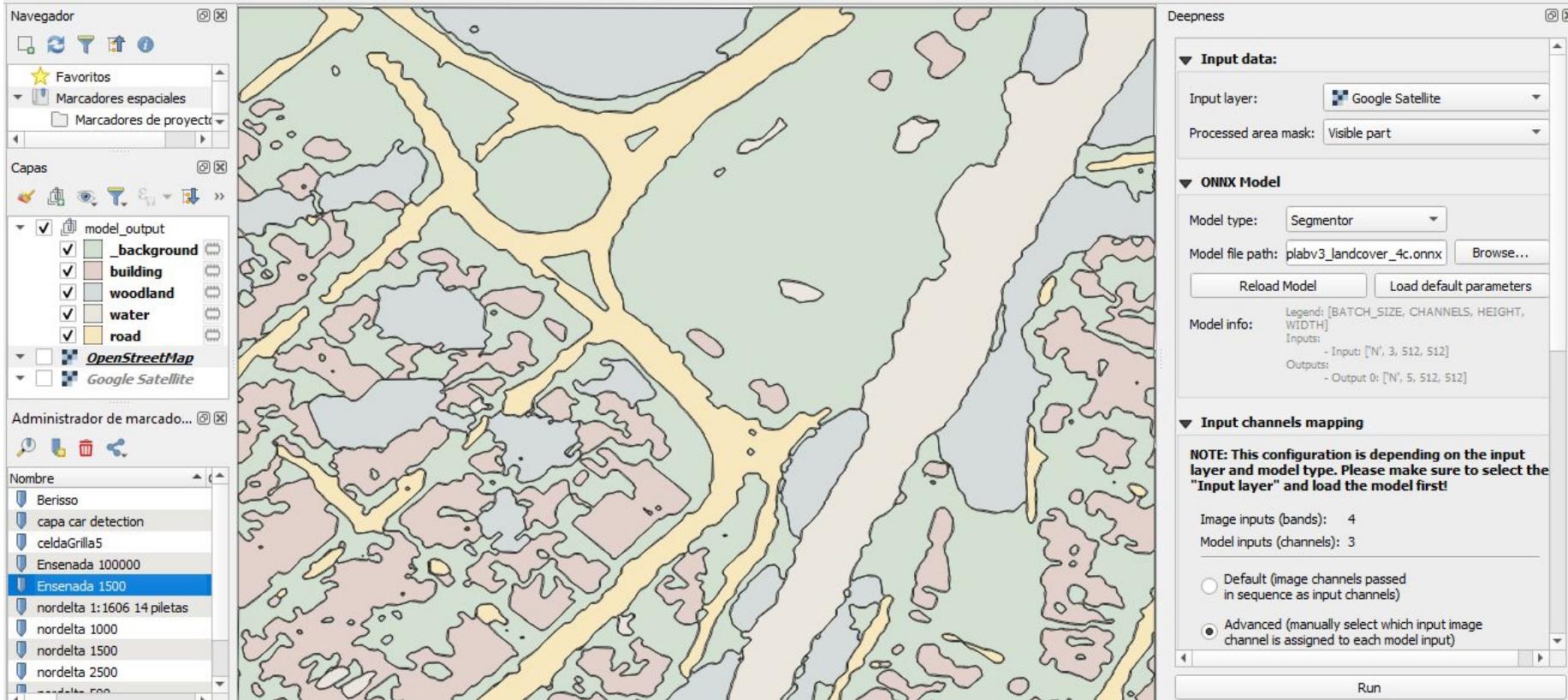
Training data export

Run

Segmentación con Modelos Existentes

Segmentación Semántica con el modelo Land Cover Segmentation

Imagen procesada. Observar capas vectoriales nuevas.



Ejemplo de Detección de objetos con Modelos Existentes

Detección de objetos

La detección de objetos no solo identifica qué objeto es, sino donde está, delimitando el objeto con un cuadro (bounding box). Es crucial para actualizar cartografía o gestionar infraestructuras

Ejemplo:

- **Modelo:** “Aerial car detection”, del model zoo
- **Input:** Imagen de una autopista.
- **Resultado:** Una nueva capa vectorial de salida donde se detectan y demarcan los vehículos como objetos .

Con el plugin DeepNess se puede utilizar cualquier modelo pre-entrenado que se encuentre en formato ONNX. Hay fuentes de este tipo de modelos, como por ejemplo HugginFace

Detección de objetos con Modelos Existentes

Fuentes de datos para obtener modelos

Con el plugin DeepNess se puede utilizar cualquier modelo pre-entrenado que se encuentre en formato ONNX. Hay gran variedad de fuentes de este tipo de modelos, como por ejemplo:

- **HuggingFace** <https://huggingface.co/models> filtrar por tarea ("object-detection", "image-segmentation") y formato. Muchos modelos ahí tienen versiones ONNX disponibles o conversiones hechas por la comunidad.
- **ONNX Model Zoo** <https://github.com/onnx/models> es el repositorio oficial de ONNX con modelos pre-entrenados de calidad, incluyendo detectores como YOLO, Faster R-CNN, y modelos de segmentación como DeepLab. Los modelos están listos para usar y bien documentados.
- **Ultralytics** <https://github.com/ultralytics> y <https://yolov8.com/> proporciona modelos YOLO en formato ONNX. YOLOv5 y YOLOv8 son excelentes para detección de objetos en tiempo real y ofrecen conversión directa a ONNX. Página de YOLO: <https://docs.ultralytics.com/es/>
- **OpenMMLab** <https://github.com/open-mmlab> ofrece repositorios especializados como MMDetection, MMSegmentation y MMPose con modelos ONNX disponibles. Tienen segmentación semántica de alta calidad.

Detección de objetos con Modelos Existentes

Imagen a procesar - Escala 1:1000

En una primera aproximación probamos con los parámetros por defecto del plugin.

The screenshot shows the QGIS application interface with the Deepness plugin loaded. On the left, the 'Navegador' (Navigator) panel displays a map with several layers: 'model_output', 'OpenStreetMap', and 'Google Satellite'. The 'Google Satellite' layer is selected. Below it, the 'Capas' (Layers) panel shows the same three layers. The 'Administrador de marcad...' (Marker Manager) panel is also visible. The main canvas displays an aerial view of a multi-lane highway with traffic, surrounded by residential buildings with red roofs. On the right, the 'Deepness' configuration panel is open. It includes sections for 'Input data' (set to 'Google Satellite' input layer and 'Visible part' mask), 'ONNX Model' (set to 'Detector' model type and 'ion_yolo7_ITCVD_deepness.onnx' file), 'Model info' (listing inputs and outputs), 'Input channels mapping' (with a note about configuration depending on input layer and model type), and 'Run' button at the bottom. The 'Input channels mapping' section also includes options for 'Default' (image channels passed in sequence as input channels) and 'Advanced' (manually select which input image channel is assigned to each model input).

Navegador

Favoritos

Marcadores espaciales

Marcadores de proyecto

Capas

model_output

OpenStreetMap

Google Satellite

Administrador de marcad...

Nombre

- Berisso
- capa car detection
- celdaGrilla5
- Ensenada 100000
- Ensenada 1500
- nordelta 1:1606 14 piletas
- nordelta 1000
- nordelta 1500
- nordelta 2500
- nordelta 3500

Deepness

Input data:

- Input layer: Google Satellite
- Processed area mask: Visible part

ONNX Model

- Model type: Detector
- Model file path: ion_yolo7_ITCVD_deepness.onnx
- Reload Model
- Load default parameters

Model info:

- Legend: [BATCH_SIZE, CHANNELS, HEIGHT, WIDTH]
- Inputs:
 - Input: [1, 3, 640, 640]
- Outputs:
 - Output 0: [1, 25200, 6]

Input channels mapping

NOTE: This configuration is depending on the input layer and model type. Please make sure to select the "Input layer" and load the model first!

Image inputs (bands): 4

Model inputs (channels): 3

Default (image channels passed in sequence as input channels)

Advanced (manually select which input image channel is assigned to each model input)

Run

Detección de objetos con Modelos Existentes

Imagen procesada. Observar capa vectorial nueva = 99 objetos

Aparece un nuevo grupo de capas: "model_output" con la clase car

Navegador

Capas

Administrador de mar...

Nombre

Berisso
capa car detection
celdaGrilla5
Ensenada 100000
Ensenada 1500
nordelta 1:1606 14 piletas
nordelta_1000

Deepness

Processing parameters

NOTE: These options may be a fixed value for some models

Resolution [cm/px]: 10,00

Tile size [px]: 640

Batch size: 1

Process using local cache

Tiles overlap:

[%] 10 [px] 0

Detection parameters

NOTE: Applicable only if a detection model is used

Detector type: YOLO_v5_or_v7_default

Inverted output shape: False
Skipped objectness: False
Ignore objectness: False

Confidence: 0,30

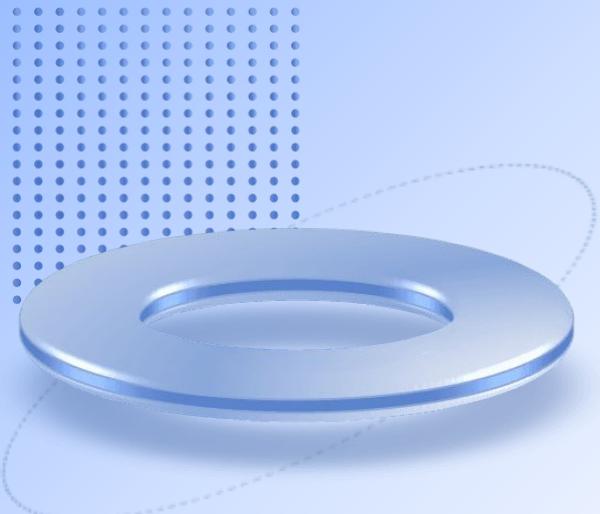
IoU threshold: 0,80

Training data export

Run

Altera el estado de edición de la capa act! Coordenada -34,543737° -58,496043° Escala 1:1000 Amplificador 100% Rotación 0,0 ° Representar EPSG:4326

Etapa 2: Desarrollar un modelo propio



El proceso completo

Desarrollar un modelo propio

Qué es una red neuronal: Modelo computacional inspirado en neuronas biológicas que aprende patrones a través de capas de procesamiento. En visión por computadora, detecta objetos identificando características visuales (bordes, texturas, formas) cada vez más complejas

Qué es Transfer Learning : No reinventar la rueda. Utilizar modelos preentrenados ya optimizados en millones de imágenes, como por ejemplo el modelo Yolo v.11:
<https://docs.ultralytics.com/es/models/yolo11/> .

- **Tiempo:** Entrenamiento en horas vs. semanas.
- **Datos:** Requiere menos imágenes etiquetadas. Datasets más chico.
- **Precisión:** Resultados más confiables desde el principio.

Enfoque Práctico: YOLO 11 es ideal para adaptar a casos específicos (detectar objetos particulares) con ajuste fino (fine-tuning) del modelo preentrenado.

La Base de Todo: Transfer Learning - Dataset

Comprendiendo y construyendo Datasets para Deep Learning Geoespacial en QGIS.

Transfer Learning como Punto de Partida

- ❖ **Definición:** Técnica de ML que reutiliza un modelo ya entrenado para una tarea general como base (*fine-tuning*) para resolver una tarea diferente pero relacionada.
- ❖ **Ventaja:** Aprovecha el conocimiento visual preexistente del modelo.
- ❖ **Ejemplo Práctico:** Para detectar piletas, se parte de un modelo robusto como **YOLOv11**.
- ❖ **Resultado:** Se ajusta (re-entrena) el modelo base con un **dataset propio** de piletas, logrando alta precisión con menos datos y tiempo.

Definición y Requisitos de un Dataset robusto



Materia Prima



Cantidad



Calidad

El Dataset es la base del Deep Learning. Se compone de imágenes satelitales y sus **anotaciones** (etiquetas).

Se necesitan muchos ejemplos en diversas condiciones (tamaños, formas, colores, iluminación, entornos).

Implica **Consistencia** (mismo criterio de etiquetado), **Cobertura** (todos los objetos) y **Diversidad** (contextos).

Preparación y Exportación de Imágenes en QGIS

Preparación (Gridding) en QGIS

Se utiliza una capa base (ej. Google Satellite) sobre el área de interés (ej. Nordelta).

Se debe crear una **cuadrícula (grilla)** con celdas de ancho y alto definidos. En QGIS Ingresar por “Vectorial”, “Herramientas de investigación”, “Crear cuadrícula”.

La escala debe permitir ver los objetos (ej. 1:850 para piletas).

Por ejemplo usar dimensiones de 250m x 160m usadas en la herramienta "Crear cuadrícula"

Exportación de Imágenes

Las celdas de la cuadrícula se exportan como imágenes individuales usando **Atlas del Compositor de Impresión** de QGIS.

Se exporta en formato PNG.

Se tilda la opción de georreferenciación. Esto crea un archivo **.PGW** adicional, que contiene 6 líneas con parámetros espaciales, el cual si bien no es estrictamente necesario para el entrenamiento del modelo, es muy útil para verificaciones.

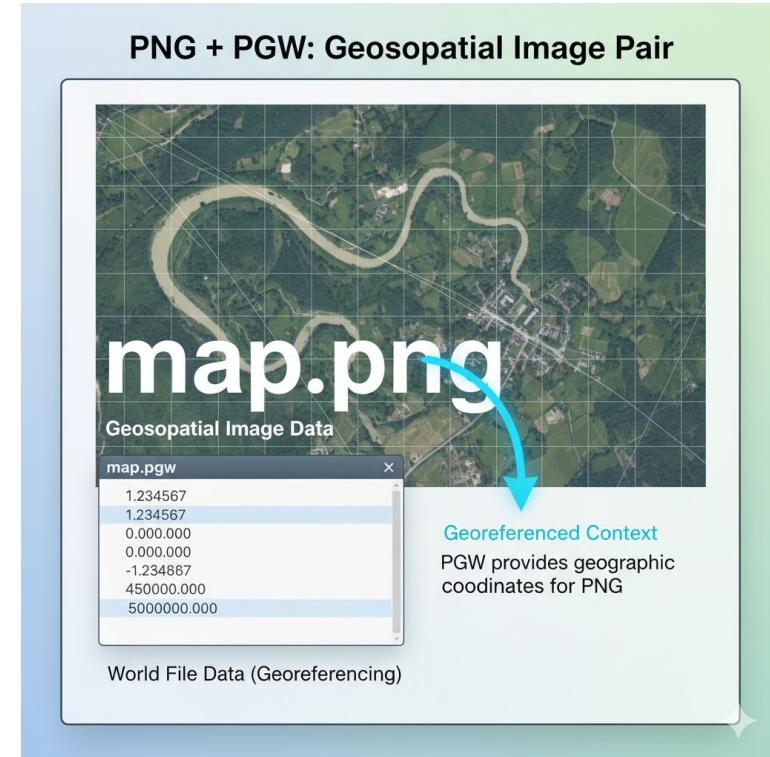
Preparación y Exportación de Imágenes en QGIS

Archivos .pgw

Un archivo **.pgw** es un world file asociado a una imagen PNG.

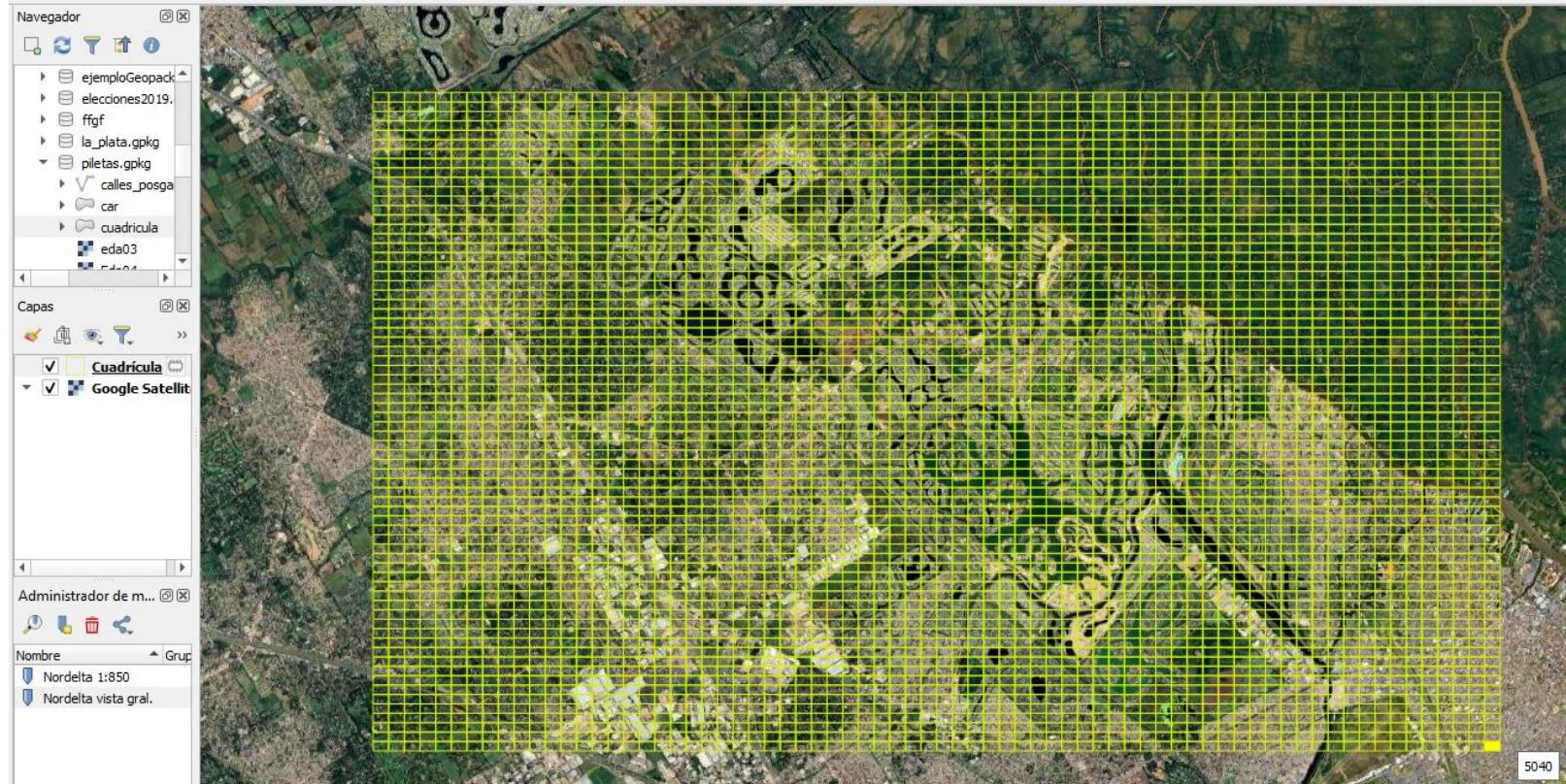
Este archivo contiene la información de georreferenciación de la imagen: Los parámetros necesarios para ubicarla correctamente en un sistema de coordenadas geográficas. El nombre "pgw" proviene de "PNG World File", pero el formato es el mismo para otros tipos de imágenes (como **.jgw** para JPEG o **.tfw** para TIFF).

El archivo **.pgw** permite que la imagen PNG sea reconocida como georreferenciada en software GIS como QGIS. Sin este archivo, la imagen sería solo un gráfico sin ubicación espacial.



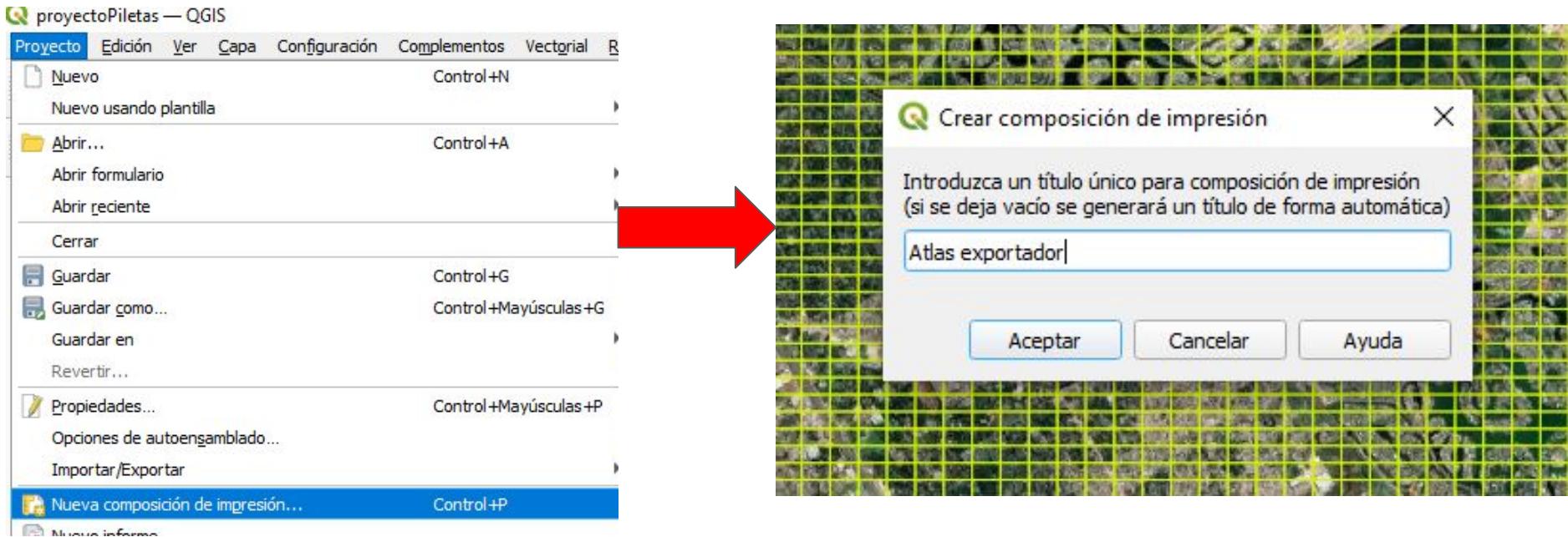
La cuadrícula se ve como una “grilla” de elementos de tipo polígono

En casos reales ajustar el área de cobertura de la grilla para cubrir los objetos de interés y algunas zonas donde NO se encuentren los objetos de interés, para enriquecer el dataset



Para exportar las imágenes de cada celda armar un Atlas.

En QGIS seleccionar menú superior “Proyecto”, “Nueva composición de impresión” e ingresar un nombre para el Atlas y click en “Aceptar”



Etiquetar las imágenes con Roboflow

Una vez exportadas las imágenes, se debe comenzar con el proceso de “etiquetado” o “anotación”. Esto es identificar los objetos de interés en la imagen con un bounding box para que el modelo aprenda y vea que estamos buscando. Para esta tarea se sugiere la aplicación Roboflow <https://roboflow.com/>

Anotación y Formato YOLO:

Las Anotaciones consisten en dibujar un Bounding Box (caja delimitadora) alrededor de cada objeto.

El formato YOLO usa archivos de tipo .txt. Un archivo por imagen. En este archivo, cada línea es un objeto.

Todas las coordenadas están normalizadas en valores entre 0 y 1.



Proceso de Etiquetado y División de Datos

- **Proceso de etiquetado:** En **Roboflow** se usa la herramienta “Bounding box tool” haciendo ajustes finos al rectángulo por medio de los grips de los vértices.
- **Contexto:** Si un árbol interfiere con la pileta, la caja debe agrandarse para dar contexto y variedad al dataset.
- **Imagenes nulas:** Las imágenes que carecen del objeto de interés deben ser etiquetadas como “Null” para enriquecer el dataset.
- **División Train/Valid:** El universo de datos se divide en Train (entrenamiento) (~80%) y Valid (validación) (~20%) para verificar el funcionamiento del modelo.

Roboflow <https://roboflow.com/>:

Es la aplicación web especializada para el etiquetado. Permite gestionar el proyecto, subir imágenes y ofrece herramientas potentes (anotación asistida, balance de datos). Roboflow ofrece herramientas avanzadas como anotación asistida por IA y balance de datos, mejorando la calidad y consistencia del dataset.

Elección de Modelo y Entorno de trabajo para el entrenamiento



Modelo: **YOLOv11**

Para el Transfer Learning (Fine-Tuning), se selecciona **YOLOv11** de Ultralytics, conocido por su alta precisión, eficiencia y versatilidad.



Entorno: **Google Colab**

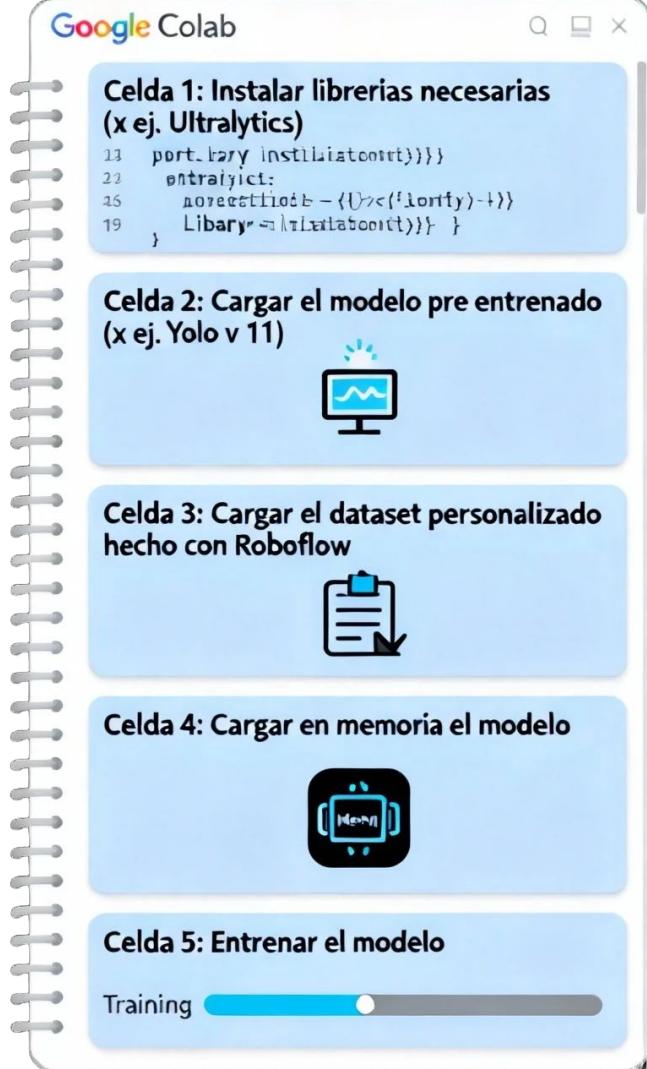
Herramienta gratuita en la nube que proporciona acceso a hardware potente (GPUs o TPUs) para ejecutar código Python.



Hardware: **GPU T4**

Es la opción gratuita altamente recomendada en Colab. Usar una CPU estándar no es recomendable (entrenamiento extremadamente lento).

Entrenamiento del modelo con Google Colab



Ajuste del Archivo data.yaml

Este archivo, descargado de Roboflow, contiene la configuración esencial (rutas y clases). Aparece cuando se descomprime el zip del dataset.

Es necesario modificarlo en Colab para reflejar las rutas correctas del entorno temporal:

- Cambiar el ítem train a:
"/content/train"
- Cambiar el ítem val a:
"/content/val"
- Cambiar el ítem names a: "[‘pileta’]"

Conceptos Clave del Entrenamiento

Época (Epoch)

Un ciclo completo de procesamiento a través de todo el conjunto de datos de entrenamiento.

Se necesitan múltiples épocas para que el modelo aprenda eficazmente los patrones.

Un batch pequeño permite actualizaciones frecuentes de pesos, acelerando la convergencia del modelo durante el entrenamiento

Tamaño del lote (batch size)

Define cuantos datos de entrenamiento se procesan en una sola iteración antes de que se actualicen los parámetros del modelo.

Ejemplo: 1000 imágenes con un batch size de 200 = 5 iteraciones por época.

Desarrollar un modelo propio.

Ejecutar las celdas definidas en Google Colab para entrenar el modelo:

Esto llevará más o menos tiempo dependiendo de la capacidad del hardware disponible y los parámetros seleccionados (cantidad de épocas y batch size principalmente)

Desarrollar un modelo propio - Evaluar el modelo

Para evaluar en forma rápida el modelo vamos a agregar una celda en Google Colab:

En el panel lateral izquierdo de Google Colab “Archivos” arrastrar y soltar una imagen en formato png y utilizar el siguiente código en una nueva celda:

```
# Hacer predicciones en una imagen  
results = model.predict(source='/content/nordelta03.png', save=True)
```

El resultado de esa celda:

```
...  
image 1/1 /content/nordelta03.png: 512x640 7 piletazas, 13.4ms  
Speed: 2.5ms preprocess, 13.4ms inference, 1.4ms postprocess per image at shape (1, 3, 512, 640)  
Results saved to runs/detect/yolo11l_datsetv10124
```

Entrenamiento, Conversión y Evaluación Final

Entrenamiento y Conversión

Una vez finalizado el entrenamiento (ej. 10 épocas), el modelo se guarda como “**best.pt**” en una carpeta de Google Colab, panel lateral izquierdo “Archivos”, en la ubicación “runs/detect/nombreDelModelo/weights/”.

Para usarlo en DeepNess (QGIS), debe ser convertido del formato .pt al formato estándar ONNX.

Para esto se usa la sentencia: “**model.export(format='ONNX')**”. El archivo se descarga y utiliza en DeepNess.

Entrenamiento, Conversión y Evaluación Final

Entrenamiento y Conversión

Es muy importante descargar este archivo al finalizar el entrenamiento, porque reside en un contexto volátil, que se perderá con la desconexión de Colab al finalizar las tareas. De todas maneras es aconsejable descargarlo por cualquier eventual desconexión del entorno de trabajo.

Conclusiones

Etapa 1 (DeepNess)

Es posible usar Deep Learning en QGIS de forma **inmediata** y sencilla para tareas comunes (segmentación, detección) utilizando modelos pre-entrenados del "Model ZOO".

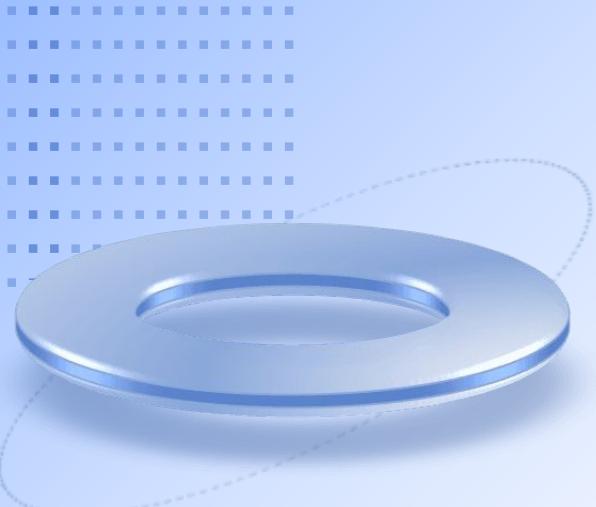
Etapa 2 (Modelo Propio)

Crear un modelo personalizado es un proceso **complejo pero potente** que requiere un pipeline estructurado (QGIS → Roboflow → Colab → QGIS).

El **Transfer Learning** permite obtener resultados de alta precisión sin necesidad de millones de imágenes.

¿Preguntas?

Gracias por su atención.



GRACIAS!

Marcelo Hosan