**Pruebas iniciales cámara Oak -1**



Índice

[I. Primer contacto cámara Oak-1 3](#_Toc197426882)

[II. Depthai viewer 4](#_Toc197426883)

[III. Repositorio fabricante 5](#_Toc197426884)

[IV. General modelo de detección de objetos personalizado 6](#_Toc197426885)

[V. Conversión modelo de detección 7](#_Toc197426886)

[VI. Test del nuevo modelo creado (.blob) desde Depthai demo 10](#_Toc197426887)

[VII. Test del nuevo modelo creado (.blob) usando un script 10](#_Toc197426888)

[VIII. Bibliografía 11](#_Toc197426889)

# Primer contacto cámara Oak-1

La cámara Oak-1 de Luxonis se conecta a través de un cable tipo C directamente al ordenador.



Ilustración . Cámara Oak-1 lite

Para poder ver el contenido de la cámara es necesario tener instalado Python (> 3.9). A continuación, se deben instalar las siguientes librerías para comenzar a visualizar imágenes de la cámara: depthai, numpy y opencv-python. Para ello, y utilizando el software Anaconda Navigator que permite crear entornos virtuales, instalaremos las librerías necesarias en un entorno nuevo.

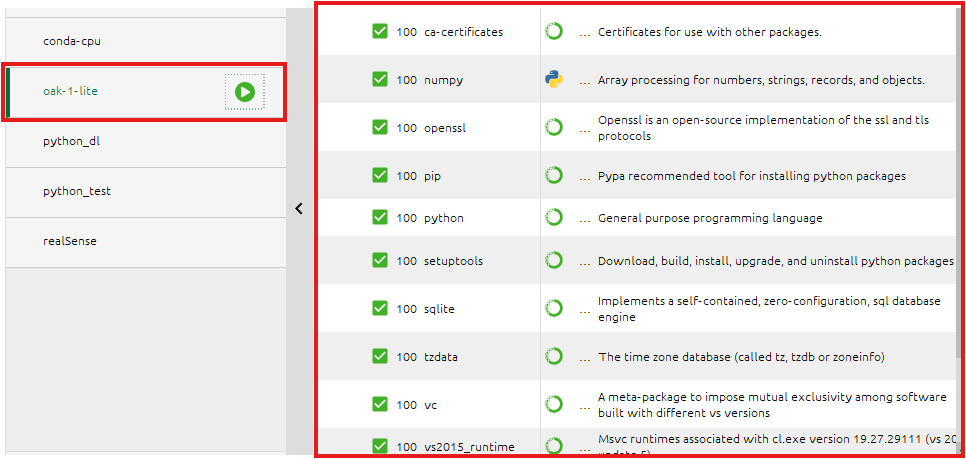


Ilustración . Entorno virtual en Anaconda

Para instalar dichas librerías, simplemente introducimos los siguientes comandos en la terminal:

* **pip install depthai**
* **pip install numpy**
* **pip install opencv-python**

A continuación, ya estamos en disposición de ejecutar un script básico de Python desde Visual Studio Code que permita visualizar la cámara en vivo.

En la carpeta *Code*, se encuentra un script llamado ***testCameraOak1.py*** que permite visualizar la cámara en vivo y realizar una captura pulsando la tecla *s* del teclado.

# Depthai viewer

Una forma rápida de poder visualizar la cámara y usar alguno de los modelos de IA que tiene integrados es usando el visualizador que tiene por defecto llamado Depthai Viewer. Para ello, es necesario instalar una nueva librería en el entorno virtual a través de la consola:

**python -m pip install depthai-viewer**

A continuación, para poder ejecutar el visualizador solamente se debe introducir el siguiente comando en la terminal:

**python -m depthai\_viewer**

Se presenta una captura de pantalla del visualizador empleando en modelo de detección *Yolo V6*.

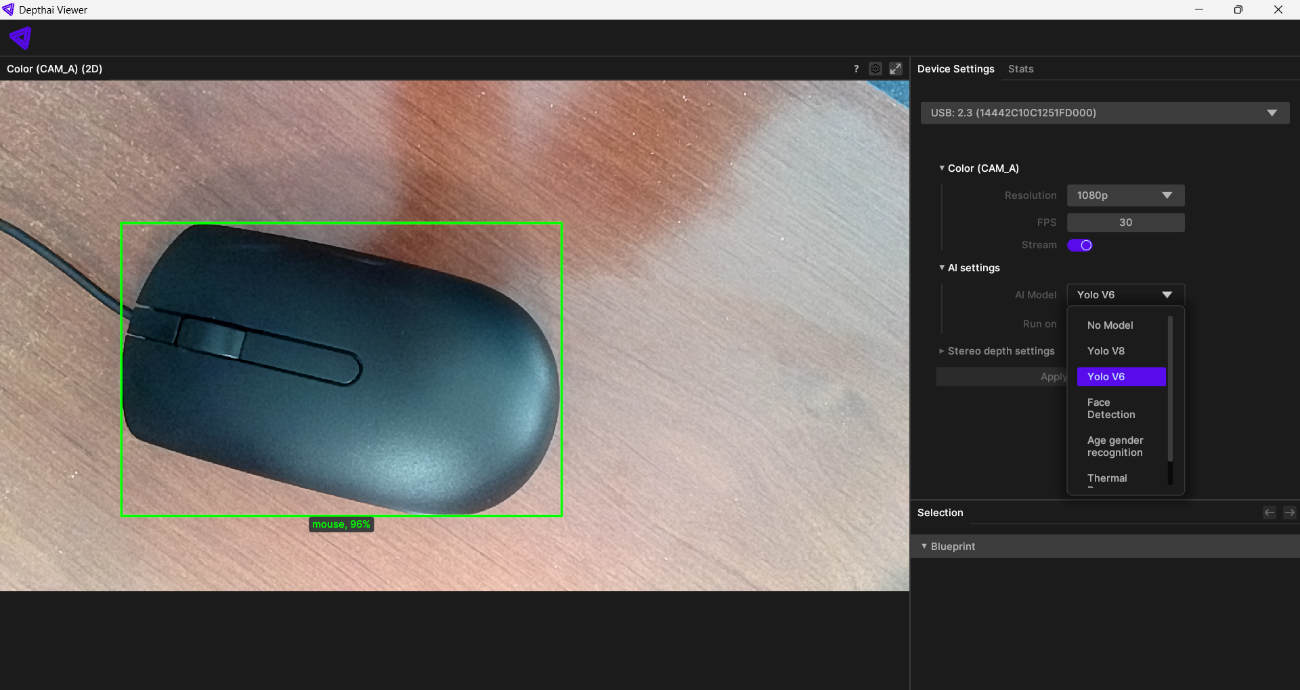


Ilustración . Depthai viewer

Para más información consultar: [DepthAI Viewer](https://docs.luxonis.com/software/tools/dai-viewer/)

# Repositorio fabricante

El fabricante de la cámara tiene un repositorio en [Github](https://github.com/luxonis/depthai) en el que existe bastante documentación de como utilizar la cámara desde Python.

Una vez descargado el repositorio, en primer lugar se debe instalar las librerías que nos requiere. Para ello vamos al directorio *Code\depthai-main\depthai-main* y ejecutamos el script *install\_requirements.py*.

A continuación, se puede probar a ejecutar el archivo *depthai\_demo.py*. Este despliega una interfaz de QT que permite visualizar la aplicación de diferentes modelos de IA en tiempo real, cambiar parámetros de la cámara, etc.

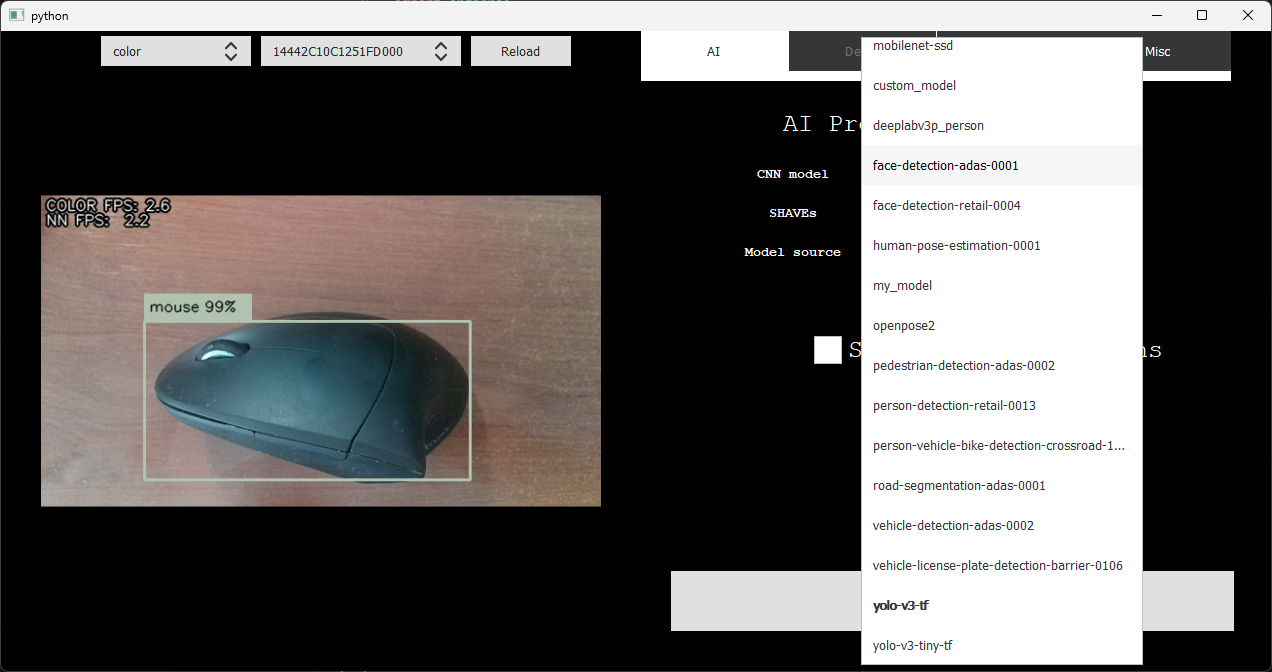


Ilustración . Depthai demo selección de modelo IA

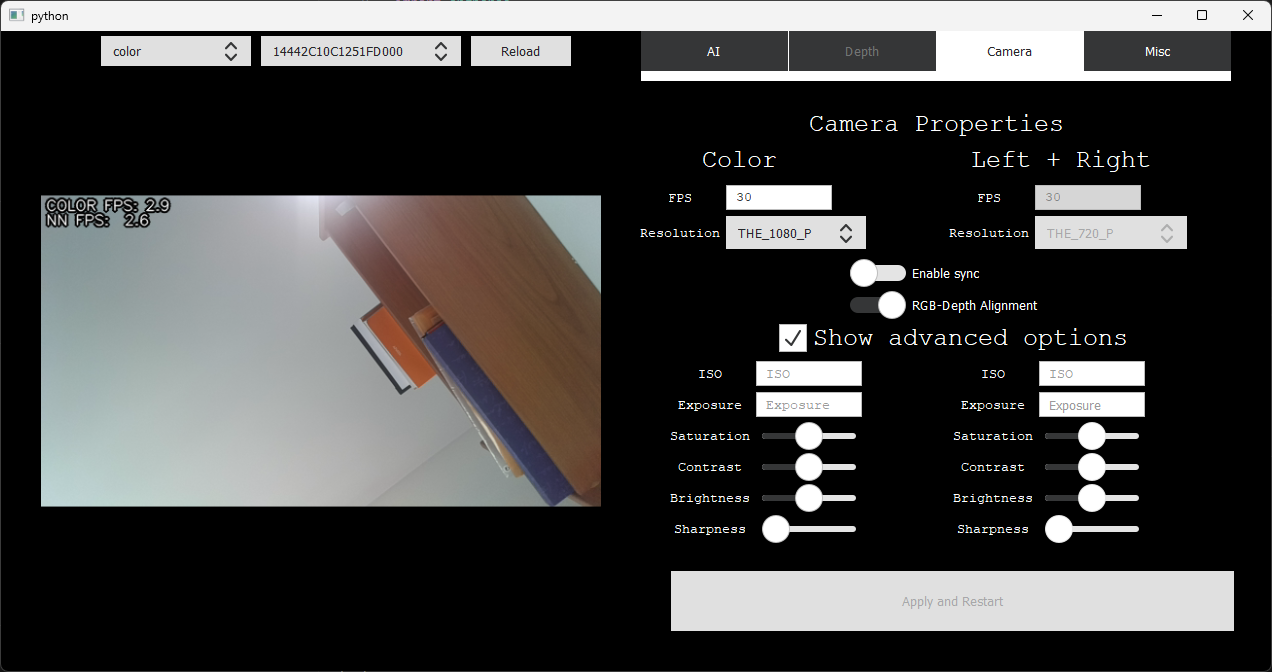


Ilustración . Depthai demo modificación parámetros cámara

# General modelo de detección de objetos personalizado

Para entrenar un modelo de detección de objetos personalizado se sigue el siguiente tutorial dónde se explican detalladamente los pasos a seguir.

[How to Train YOLO Object Detection Models in Google Colab (YOLO11, YOLOv8, YOLOv5) - YouTube](https://www.youtube.com/watch?v=r0RspiLG260)

Se crea un entorno vitual anaconda.

Se instala label studio y se ejecuta (<https://labelstud.io/guide/quick_start>

):

* pip install label-studio
* label-studio start

Es necesario crear una cuenta en la plataforma de label studio. A continuación, se crea un nuevo proyecto del tipo *Object Detection with Bounding* Boxes, se suben y etiquetan las imágenes para finalmente exportar el dataset en formato *“YOLO”*. Se descargará una carpeta comprimida con toda la información relativa al etiquetado del dataset.

El siguiente paso es ejecutar en Colab el código ([Train\_YOLO\_Models.ipynb - Colab](https://colab.research.google.com/github/EdjeElectronics/Train-and-Deploy-YOLO-Models/blob/main/Train_YOLO_Models.ipynb)) para entrenar el modelo siguiendo el tutorial indicado al principio. A destacar que en el apartado 5.2 podemos modificar los siguientes parámetros para que se ajusten más a nuestras necesidades.

!yolo detect train data=/content/data.yaml model=yolo11s.pt epochs=60 imgsz=320

Después del entrenamiento se valida el modelo con algunas imágenes y finalmente se obtiene una carpeta my\_model.zip que contiene el modelo .pt que se utilizará más adelante.

# Conversión modelo de detección

Para realizar la conversión existen dos formas, siendo la primera de ellas quizás más sencilla.

**Opción 1**

Para poder procesar el modelo de detección de objetos dentro de la cámara es necesario pasar el archivo .pt a .blob. Para ello, el fabricante de la cámara dispone de un conversor en línea que permite transformarlo directamente: <https://tools.luxonis.com/>.

Es muy sencillo de usar, simplemente hay que cargar el modelo con extensión .pt e indicar el tamaño de la imagen de entrada. Este último debe ser igual al del entrenamiento del modelo.

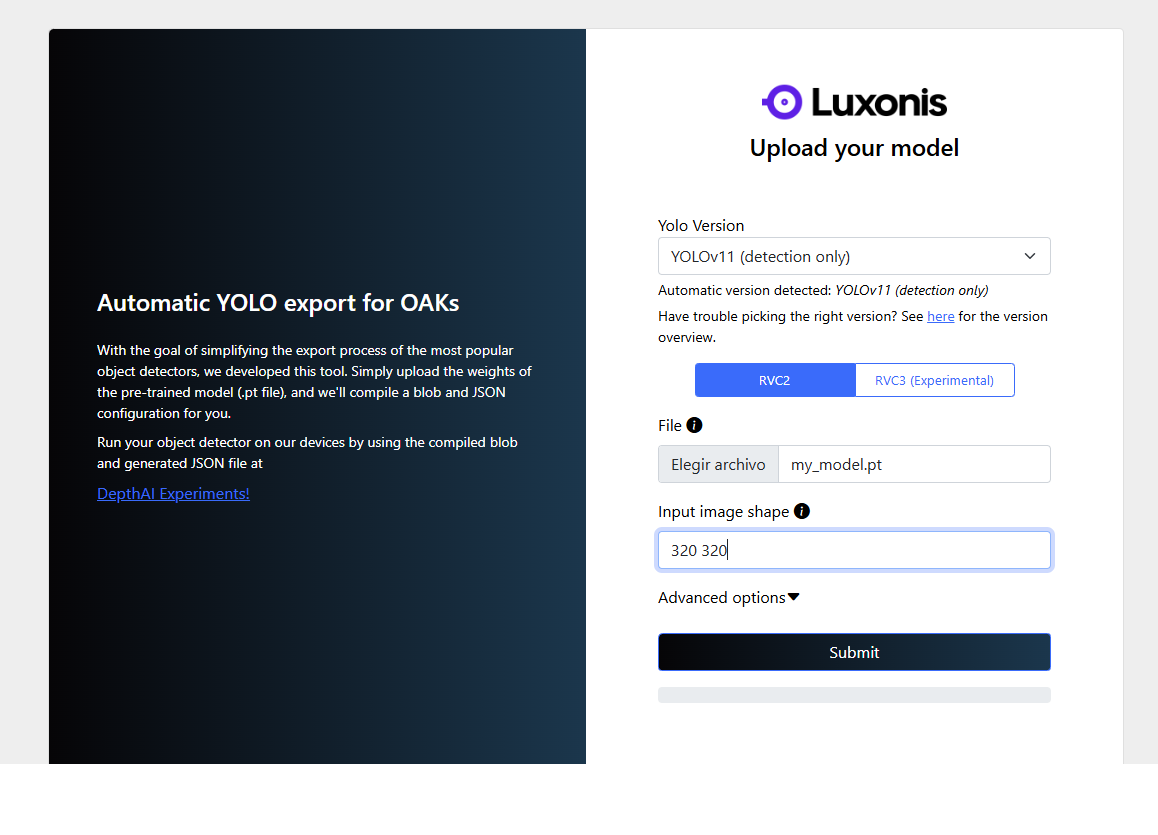


Ilustración . Convertidor online Luxonis

Al terminar la conversión, se descargará un archivo comprimido que contiene 5 archivos incluido el modelo .blob que utilizaremos más adelante.

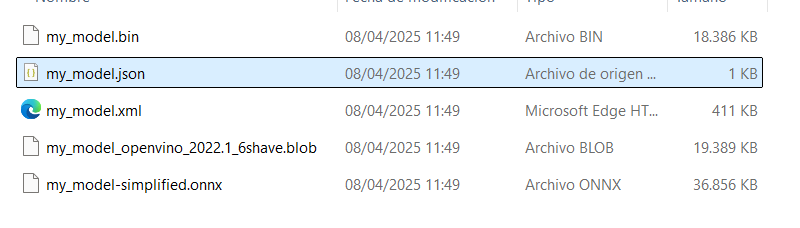


Ilustración . Contenido de la carpeta comprimida

Más información: https://docs.luxonis.com/software/ai-inference/integrations/yolo/

**Opción 2**

En este caso usaremos un convertidor online que solo permite convertir los formatos PyTorch (ONNX), TensorFlow, Caffe y OpenVINO ZOO model, por lo que primero tendremos que hacer otra conversión del formato .pt a alguno de los formatos mencionados.

Utilizando el script convert.py, convertimos el modelo .pt a .onnx.

Ahora, empleando el convertidor online (<https://blobconverter.luxonis.com/> ), seleccionamos “ONNX” y clicamos en continuar. A continuación, cargamos el archivo anterior (.onnx) y clicamos en convertir manteniendo los parámetros que tiene por defecto.

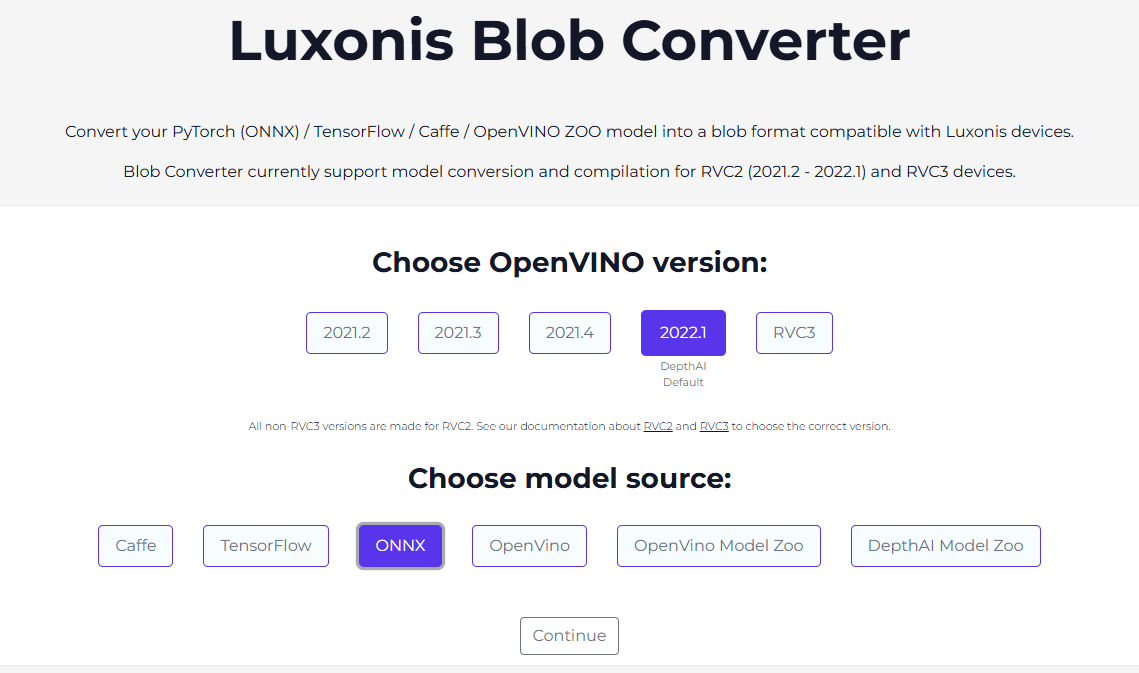


Ilustración . Blob converter, selección tipo de modelo origen

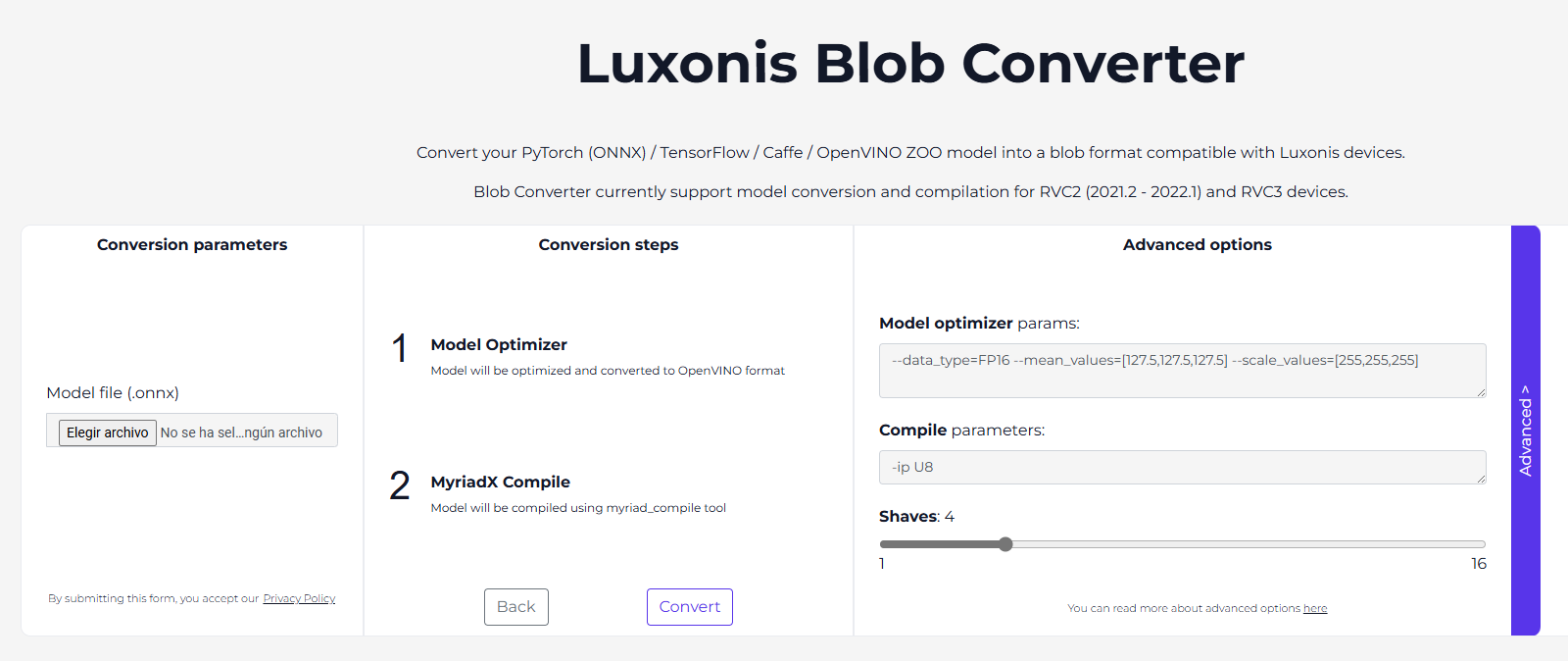


Ilustración . Blob converter, carga de fichero

Al finalizar la conversión, se descargará un fichero . blob del modelo.

Información más detallada en <https://docs.luxonis.com/software/ai-inference/conversion/>.

# Test del nuevo modelo creado (.blob) desde Depthai demo

Para validar el funcionamiento del modelo .blob, utilizaremos el visualizador Depthai\_demo.py del apartado III que nos permite ver en tiempo real el funcionamiento de modelos. Para añadir un nuevo modelo, se debe crear una carpeta con el nombre del modelo (my\_model) en la siguiente ruta *Code\depthai-main\depthai-main\resources\nn*. A continuación, se deben añadir en dicha carpeta:

* Todos los ficheros generados en la opción 1 del apartado IV.
* Una copia del fichero *handler.py* ubicado en *Code\depthai-main\depthai-main\resources\nn\custom\_model*.

Ahora ya estamos en disposición de validar el buen funcionamiento del modelo. Para ello se ejecuta el script *depthai\_demo.py y seleccionar el modelo añadido (my\_model).*

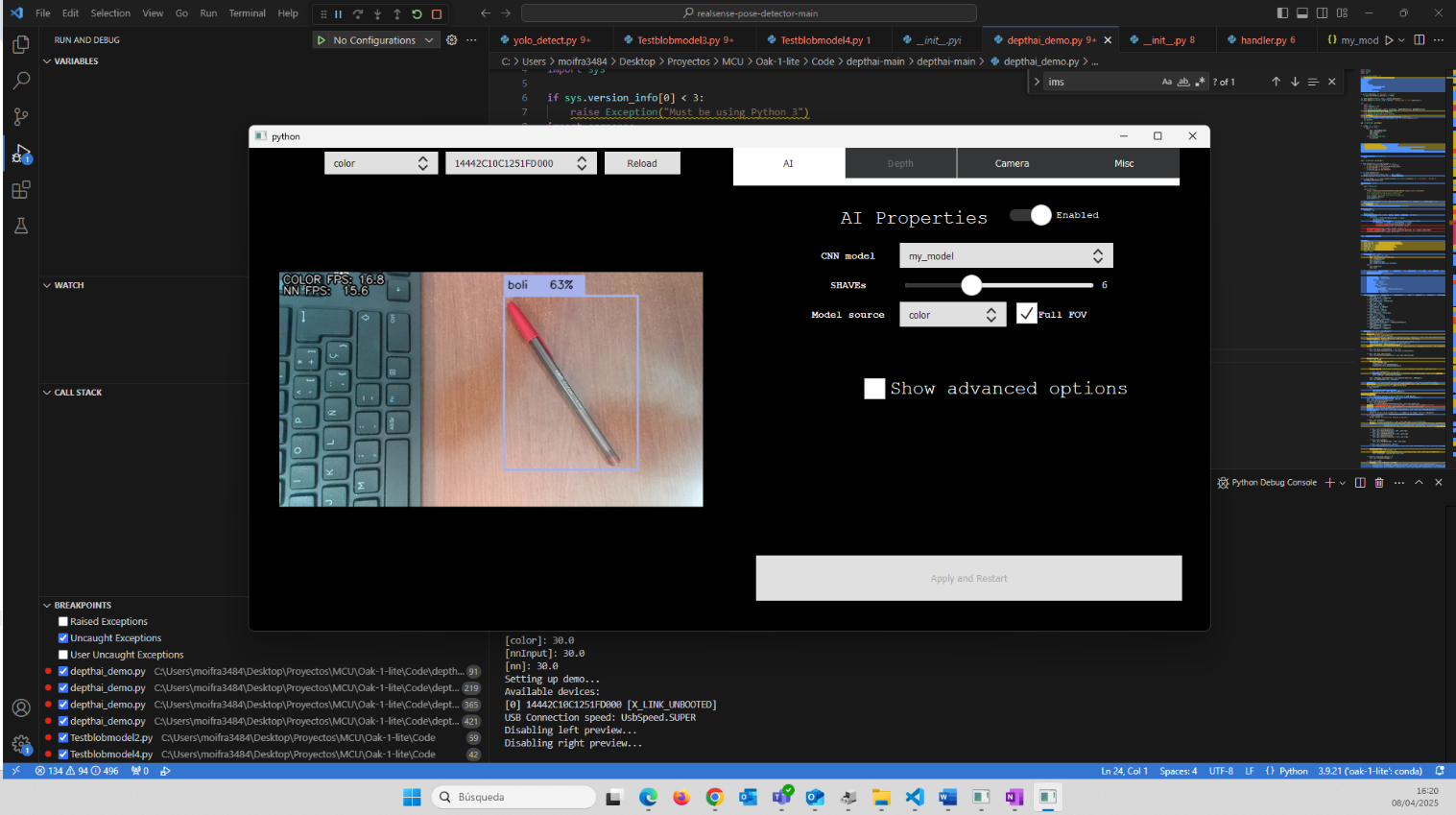
**

Ilustración . Nuevo modelo desde Depthai demo

# Test del nuevo modelo creado (.blob) usando un script

Para comprobar el correcto funcionamiento de dicho modelo se debe ejecutar el script *testCameraOak1BlobModel.py*. Se deben añadir las rutas del fichero .json y .blob, el primero contiene información del modelo entrenado (etiquetas, resolución, etc.) y el segundo tiene el modelo compilado. Ambos archivos se han generado en la opción 1 del apartado IV.

Al lanzar el script, visualizaremos en vivo la cámara Oak-1 capaz de detectar los objetos entrenados.

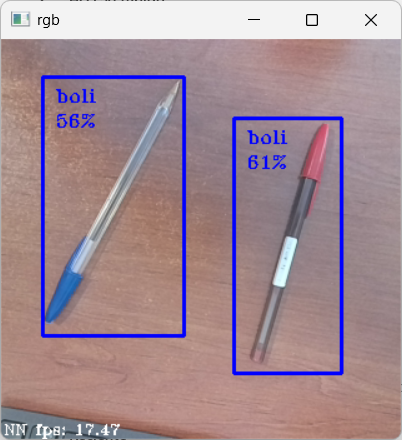


Ilustración . Captura pantalla del test del modelo .blob

# Bibliografía

<https://docs.luxonis.com/software/depthai-components/device/>

<https://github.com/luxonis/depthai>

https://docs.luxonis.com/software/depthai/examples/yolov8\_nano/