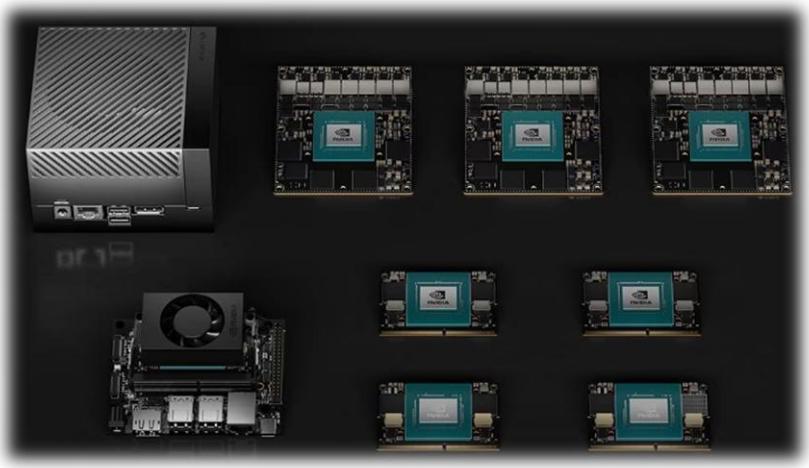
.:: NVIDIA Jetson con Ultralytics YOLO11 ::.

By Freddy Alacarazo | @surflaweb

23 de Abril del 2025



¿Qué es NVIDIA Jetson?

NVIDIA Jetson es una serie de placas de computación integradas diseñadas para llevar la computación acelerada de IA (inteligencia artificial) a los dispositivos periféricos. Estos dispositivos compactos y potentes se basan en la arquitectura GPU de NVIDIA y son capaces de ejecutar complejos algoritmos de IA y modelos de aprendizaje profundo directamente en el dispositivo, sin necesidad de depender de recursos de computación en la nube. Las placas Jetson se utilizan a menudo en robótica, vehículos autónomos, automatización industrial y otras aplicaciones en las que la inferencia de IA debe realizarse localmente con baja latencia y alta eficiencia. Además, estas placas se basan en la arquitectura ARM64 y consumen menos energía que los dispositivos informáticos tradicionales GPU.

¿Qué es NVIDIA JetPack?

El kit de desarrollo de software NVIDIA JetPack que incorpora los módulos Jetson es la solución más completa y proporciona un entorno de desarrollo completo para crear aplicaciones de IA aceleradas de principio a fin y acortar el tiempo de comercialización. JetPack incluye Jetson Linux con gestor de arranque, kernel Linux, entorno de escritorio Ubuntu y un completo conjunto de librerías para aceleración de GPU computing, multimedia, gráficos y visión computerizada. También incluye muestras, documentación y herramientas de desarrollo tanto para el ordenador anfitrión como para el kit de desarrollo, y es compatible con SDK de nivel superior

como DeepStream para análisis de vídeo en streaming, Isaac para robótica y Riva para IA conversacional.

Soporte para JetPack basado en el dispositivo Jetson

La siguiente tabla muestra las versiones de NVIDIA JetPack soportadas por los diferentes dispositivos NVIDIA Jetson:

	JetPack 4	JetPack 5	JetPack 6
Jetson Nano	✅	❌	❌
Jetson TX2	✅	❌	❌
Jetson Xavier NX	✅	✅	❌
Jetson AGX Xavier	✅	✅	❌
Jetson AGX Orin	❌	✅	✅
Jetson Orin NX	❌	✅	✅
Jetson Orin Nano	❌	✅	✅

Ejecutar en JetPack 6.1

Flashear el Jetpack 6.1 al jetson usando imagen iso o bien mediante SDK manager. Vídeo tutorial: https://youtu.be/eonMa5g0n8k

Descarga SDK manager: https://developer.nvidia.com/sdk-manager

Pasos:

1. **Instalar el paquete Ultralytics**

   Aquí instalaremos el paquete Ultralytics en el Jetson con dependencias opcionales para que podamos exportar los PyTorch modelos a otros formatos diferentes. Nos centraremos principalmente en NVIDIA TensorRT exportaciones porque TensorRT se asegurará de que podemos obtener el máximo rendimiento de los dispositivos Jetson.

1.1 Actualizar la lista de paquetes, instalar pip y actualizar a la última versión

```
sudo apt update
sudo apt install python3-pip -y
pip install -U pip
pip install --upgrade pip
```

Con pip instalado, ahora crear enviroment para no instalar todo en la raíz y evitarnos problemas:

```
# Crear el enviroment:

python3 -m venv --system-site-packages venv

# Activar el enviroment:

source venv/bin/activate
```

Proceder con los siguientes pasos.

1.2 Instalar ultralytics vía pip

```
pip install ultralytics==8.3.70
```

1.3 Reiniciar el dispositivo (opcional)

```
sudo reboot
```

2. Instalar PyTorch y Torchvision

La instalación anterior de ultralytics instalará Torch y Torchvision. Sin embargo, estos 2 paquetes instalados a través de pip no son compatibles para ejecutarse en la plataforma Jetson que se basa en la arquitectura ARM64. Por lo tanto, tenemos que desinstalarlos (torch y torchvision) e instalar manualmente PyTorch pip wheel y compilar / instalar Torchvision desde el código fuente.

 # desinstalar torch y torchvision

```
pip uninstall torch torchvision
```

2.1 Instalar torch 2.5.0 y torchvision 0.20 según JP6.1

```
pip install https://github.com/ultralytics/assets/releases/download/v0.0.0/torchvision-0.20.0a0+afc54f7-cp310-cp310-linux_aarch64.whl
pip install https://github.com/ultralytics/assets/releases/download/v0.0.0/torch-2.5.0a0+872d972e41.nv24.08-cp310-cp310-linux_aarch64.whl
```

** Copia de seguridad de estos whl's aquí en el drive de google:

https://drive.google.com/drive/folders/1iouzZSy9YAZ-w7w2WYtAdG3n2tdC1egW?usp=sharing

2.2 Instalar cuSPARSELt para solucionar un problema de dependencia con torch 2.5.0

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/arm64/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get -y install libcusparselt0 libcusparselt-dev
```

** Copia de seguridad de **cuda-keyring_1.1-1_all.deb** ver en el drive:

https://drive.google.com/drive/folders/1iouzZSy9YAZ-w7w2WYtAdG3n2tdC1egW?usp=sharing

2.3 Instalar onnxruntime-gpu

En onnxruntime-gpu alojado en PyPI no tiene aarch64 binarios para la Jetson. Así que tenemos que instalar manualmente este paquete. Este paquete es necesario para algunas de las exportaciones.

Puede encontrar todos los **onnxruntime-gpu** organizados por versión de JetPack, versión de Python y otros detalles de compatibilidad. Matriz de compatibilidad de Jetson Zoo ONNX Runtime. Aquí descargaremos e instalaremos **onnxruntime-gpu 1.20.0** con **Python3.10:**

```
pip install https://github.com/ultralytics/assets/releases/download/v0.0.0/onnxruntime_gpu-1.20.0-cp310-cp310-linux_aarch64.whl
```

** Copia de seguridad de **onnxruntime-gpu** ver en el drive:

https://drive.google.com/drive/folders/1iouzZSy9YAZ-w7w2WYtAdG3n2tdC1egW?usp=sharing

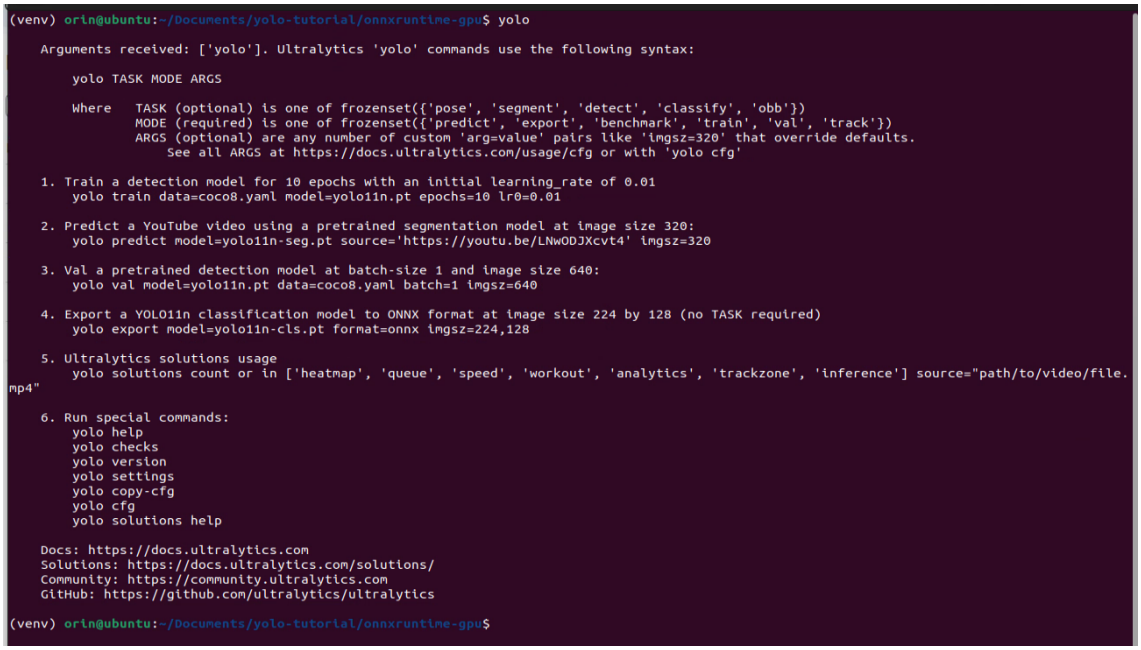
Onnxruntime-gpu revertirá automáticamente la versión de numpy a la última. Así que necesitamos reinstalar **numpy a 1.23.5** para solucionar un problema ejecutando:

```
pip uninstall numpy
```

```
pip install numpy==1.23.5
```

**** Antes de continuar, prueba que el comando "yolo" ya funciona y nos arroja información:**

```
yolo
```



3. Descargar un modelo YOLO listo para usar (off-the-shelf YOLO model)

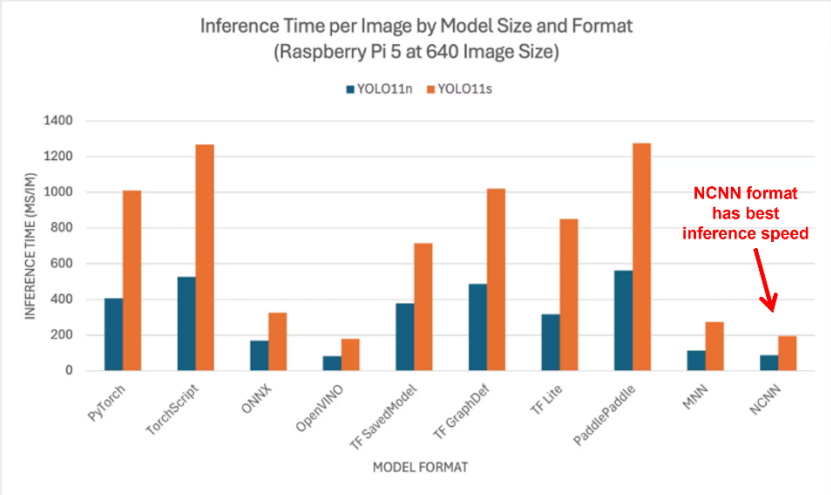
Para ello, usaremos Ultralytics el cual facilita la descarga y el uso de un modelo YOLO estándar. Estos modelos se entrenan con el conjunto de datos COCO y pueden detectar 80 objetos comunes, como "persona", "coche", "silla", etc. Para descargar un modelo de detección YOLO11n, ejecute:

```
yolo detect predict model=yolo11n.pt
```

Esto descargará automáticamente un archivo de modelo "yolo11n.pt" y lo ejecutará en un par de imágenes de prueba. Si deseamos usar un modelo diferente, simplemente reemplazamos "yolo11n.pt" por otro modelo, como "yolo11s.pt" o "yolov8n.pt".

4. **Exportar el modelo YOLO al formato NCNN**

**** Nota:** esto es solo para a acelerar el tiempo de inferencia y que no sea muy lento.



Instalar ncnn a través de pip:

```
$ pip install ncnn
```

**** Nota:** A fecha de este tutorial/guía al inspeccionar la versión instalada de "ncnn" se usó:

ncnn==1.0.20241226

En caso de que en el futuro suceda problemas usen esa versión.

Exportar modelo:

```
# Reemplazar la palabra your_model.pt por el nombre del modelo.

yolo export model=your_model.pt format=ncnn
```

Quedaría así:

```
yolo export model=yolo11n.pt format=ncnn
```

Una vez convertido, usaremos este para la inferencia.

5. **Ejecutar inferencia el modelo Yolo**

Descargar script para probar desde el navegador o bien usando wget:

```
wget https://ejtech.io/code/yolo_detect.py
```

Instalar dependencias del script:

Este script require de tener instalado opencv y numpy, sin embargo estas dos dependencias ya se instalaron antes. El opencv se instalo automáticamente con Ultralytics y también antes instalamos numpy.

Parámetros para ejecutar el script:

- **--model**: Path to the trained model weights (e.g. “runs/detect/train/weights/yolo11n_ncnn_model”)

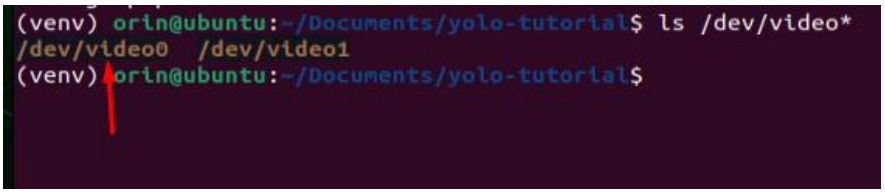
- **--source**: Path to an image file (“test_img.jpg”), a folder of images (“img_dir”), a video file (“test_vid.mp4”), or the index for a connected USB camera (“usb0”).

- **--resolution** (optional): Resolution in WxH to display inference results at. If not specified, the program will match the source resolution.

Ejecutar el siguiente comando para ejecutar la detección con el modelo "yolo11n_ncnn_model" usando una cámara USB conectada con una resolución de 640x480. Si usa un modelo personalizado, reemplace "yolo11n_ncnn_model" con el nombre de la carpeta del modelo NCNN personalizado (por ejemplo, "custom_ncnn_model").

Conectar webcam y verificar si está en la lista de cámaras:

```
$ ls /dev/video*
```



Ejecución:

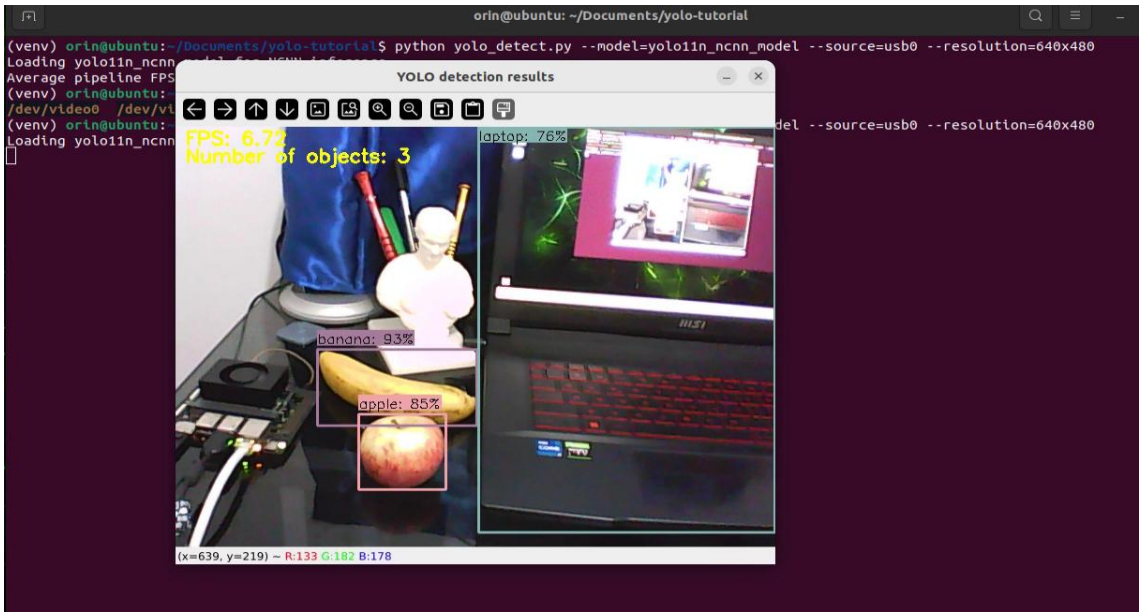
```
# Ejecutar a una resolución de 640x480

python yolo_detect.py --model=yolo11n_ncnn_model --source=usb0 --
resolution=640x480


# Ejecutar usando una resolución de 1280x720

python yolo_detect.py --model=yolo11n_ncnn_model --source=usb0 --
resolution=1280x720
```

Resultado:



A continuación, se muestran otros ejemplos de comandos que muestran cómo ejecutar el script en un archivo de vídeo o una imagen:

```
# Run on video file named “test_video.mp4”
python yolo_detect.py --model=yolo11n_ncnn_model --source=test_video.mp4

# Run on folder of images named “img_dir” en un modelo custom.
python yolo_detect.py --model=custom_ncnn_model --source=img_dir
```

6. Fuentes

- https://docs.ultralytics.com/es/guides/nvidia-jetson/#install-onnxruntime-gpu
- https://www.ejtech.io/learn/yolo-on-raspberry-pi
- https://www.youtube.com/watch?v=z70ZrSZNi-8