Razvoj aplikacij AI na robu v ekosistemu NVIDIA Jetson

zbral in uredil: *Simon Pogorelčnik univ. dipl. inž.*

NVIDIA Jetson Orin

Razvoj aplikacij AI na platformi NVIDIA Jetson (kot je Orin Nano) se močno razlikuje od običajnega razvoja aplikacij na podobnih razvojnih mikro-računalnikih (npr. Arduino, Raspbery,…), saj vključuje optimizacijo modelov in izkoriščanje posebnosti strojne opreme.

Poskušal bom opisati potek razvoja Edge AI aplikacij na platformi Jetson in podati praktične napotke za prve korake. Najprej naj opišem sam postopek, ki je razdeljen na šest ključnih korakov:

Koraki razvoja Edge AI aplikacij na NVIDIA Jetson

1. Korak: Nastavitev okolja (JetPack SDK)

Prvi korak je priprava same strojne opreme (Jetson Orin Nano) in namestitev potrebnega programskega ogrodja.

- **Namestitev JetPack SDK:** To je temeljni programski paket podjetja NVIDIA, ki vključuje:
  - ✓ **L4T (Linux for Tegra):** Optimiziran operacijski sistem (Ubuntu) za Jetson.
  - ✓ **CUDA Toolkit:** Omogoča uporabo GPU-ja za splošne izračune.
  - ✓ **cuDNN:** Knjižnica za pospeševanje nevronskih mrež (globoko učenje).
  - ✓ **TensorRT in VPI:** Orodja za optimizacijo in inferenco modelov (glej korak 4).
- Nastavitev senzorjev in I/O: Povezava kamere, senzorjev (GPIO, MIPI CSI-2), diska in drugih perifernih naprav, s katerimi bo aplikacija delovala.

2. Korak: Pridobivanje in Treniranje AI Modela (Oblak ali Zunaj)

AI model se skoraj nikoli ne trenira neposredno na Jetson napravi, saj ta nima dovolj virov za hitro in učinkovito treniranje velikih modelov.

- **Izbira modela:** Izbira ustrezne arhitekture (npr. YOLO za prepoznavanje objektov, ResNet za klasifikacijo, manjši LLM za obdelavo jezika).
- **Trening modela:** Trening poteka na zmogljivih **Cloud/PC GPU strežnikih** z uporabo ogrodij, kot so PyTorch ali TensorFlow.
- **Prenos in prilagoditev:** Po treniranju se model shrani in prenese na Jetson napravo v enem od standardnih formatov (npr. ONNX).

3. Korak: Optimizacija modela za Jetson (TensorRT)

To je kritičen korak, ki Edge AI loči od Cloud AI, saj omogoča, da se model izvede izjemno hitro in energetsko učinkovito.

- **Pretvorba v TensorRT format:** Uporaba orodja **NVIDIA TensorRT**, ki analizira model in ga **optimizira** za specifično strojno opremo (Tensor jedra) na Jetsonu.
- **Kvantizacija (Quantization):** Zmanjšanje natančnosti številk znotraj modela (npr. iz 32-bitne plavajoče vejice na 8-bitno celo število - **INT8**). To bistveno poveča hitrost inference in zmanjša porabo moči, z minimalnim vplivom na natančnost.
- **Model Pruning (Obrezovanje):** Odstranjevanje nepotrebnih povezav in uteži, kar dodatno zmanjša velikost modela.

4. Korak: Razvoj Aplikacije (Inference Pipeline)

Po optimizaciji je treba model vključiti v delujočo aplikacijo, ki obdeluje podatke v realnem času.

- **Pridobivanje podatkov:** Uporaba knjižnic za zajem podatkov iz senzorjev ali video tokov (npr. OpenCV).
- **Predprocesiranje:** Priprava surovih podatkov za vnos v model (npr. spreminjanje velikosti slike, normalizacija).
- **Inference: Izvajanje** optimiziranega modela (TensorRT) za pridobivanje napovedi.
- **Postprocesiranje in odločanje:** Interpretacija rezultatov modela in sprožitev dejanske akcije (npr. "Če je napoved 'neznan objekt', ustavi robota," ali "Zaznana je okvara, pošlji opozorilo.").
- **NVIDIA DeepStream:** Za aplikacije računalniškega vida se pogosto uporablja ogrodje DeepStream, ki izjemno pospeši izgradnjo video analitičnih cevi.

5. Korak: Testiranje in Profiliranje

Aplikacijo je treba temeljito testirati na sami napravi, kjer bo delovala.

- **Testiranje v realnem času:** Preverjanje, ali aplikacija dosega zahtevano hitrost (FPS - sličice na sekundo) in nizko latenco.
- **Merjenje porabe virov:** Uporaba orodij za preverjanje obremenitve GPU/CPU in porabe pomnilnika (Memory Usage).
- **Energetska učinkovitost:** Pri napravah, ki delujejo na baterijo (kot so droni), je ključno preverjanje in optimizacija porabe energije.

6. Korak: Uvajanje in Posodabljanje (Deployment & OTA)

Zadnji korak je uvedba aplikacije v produkcijo.

- **Deployment na rob:** Namestitev aplikacije na končno napravo, pogosto v obliki DockeR kontejnerjev za poenostavitev upravljanja odvisnosti.
- **Upravljanje flote naprav:** Pri razvoju za več naprav je potrebno centralno orodje (npr. NVIDIA Fleet Command) za daljinsko spremljanje in odpravljanje napak.
- **Posodobitve (OTA - Over The Air):** Priprava mehanizmov za oddaljeno posodabljanje programske opreme in AI modelov. To omogoča stalno izboljševanje modela, ko se v oblaku natrenira nova, natančnejša različica.

Platforma Jetson Orin Nano

Jetson Orin Nano predstavlja odlično platformo za razvoj umetne inteligence na robu (edge AI). Jetson Orin Nano 8GB z JetPack 6.2 ponuja izjemno moč, še posebej z novimi optimizacijami za generativno umetno inteligenco (AI).

Ker imamo 8 GB pomnilnika (RAM/VRAM), so za **trening** (zlasti transferno učenje ali fine-tuning) in **inferenco** (izvajanje) primerni predvsem **manjši in optimizirani modeli**.

Tukaj so najbolj priporočljive kategorije in primeri modelov za vašo strojno opremo:

1. Modeli za računalniški vid (Computer Vision)

To je ključno področje za Jetson. Osredotočite se na arhitekture, ki so optimizirane za hitro delovanje na omejenih virih:

Naloga	Priporočeni Modeli	Opombe
Zaznavanje objektov (Object Detection)	**YOLO** (npr. YOLOv5n, YOLOv8n, YOLOv8s), SSD, Tiny-YOLO	Majhne različice zagotavljajo odlično hitrost in natančnost.
Klasifikacija slik (Image Classification)	**MobileNet** (vse različice), **EfficientNet** (manjše različice), **ResNet** (manjše različice)	Zasnovani za mobilne in vgrajene sisteme.
Segmentacija	**NanoOWL**, **SAM** (Segment Anything Model - manjše različice)	Uporabni za naprednejše aplikacije, kot so robotika in avtonomna vozila.
Vision Transformers (ViT)	**DINOv2** (osnovne različice), **CLIP** (manjše različice)	JetPack 6.2 ima izboljšave tudi za te novejše arhitekture.

2. Mali jezikovni modeli (Small Language Models - SLM)

Generativna AI je zdaj dosegljiva tudi na Orin Nano! Priporočeni so modeli z manj kot 10 milijardami parametrov, ki so pogosto **kvantizirani** (zmanjšana natančnost za manjšo porabo pomnilnika in večjo hitrost).

Model	Priporočena Velikost	Opombe
Llama 3.2	**1B do 3B** (milijard parametrov), lahko tudi **8B** z močno kvantizacijo.	Ena izmed najbolj zmogljivih serij, dobro optimizirana.
Gemma 2	**2B**	Google-ov odprtokodni model, ki je zasnovan za učinkovitost.
Phi-3.5	**3.8B (Mini)**	Kompaktni model, primeren za "chat" aplikacije.
Qwen2	Manjše različice (npr. 7B)	Dosega dobro hitrost in kakovost izpisa.
SmolLM2	**1.7B**	Posebej majhen in učinkovit model.

Ključni dejavniki za uspešen razvoj

- **Optimizacija (Inferenca):** Vedno uporabite **NVIDIA TensorRT** za pretvorbo in optimizacijo modelov, preden jih izvajate (inferenca). TensorRT dramatično pospeši izvajanje modelov na GPU-ju.
- **Kvantizacija (Quantization):** Uporaba formatov, kot so INT8, FP16 ali celo INT4, je ključna za zmanjšanje porabe pomnilnika in hitrejše delovanje (še posebej pri LLM-jih).
- **Transferno učenje (Fine-Tuning):** Namesto treniranja od začetka (ki je zahtevno z 8GB), uporabite metode, kot je **LoRA** za prilagoditev obstoječih pred-treniranih modelov vašim podatkom.
- **NVIDIA TAO Toolkit:** Razmislite o uporabi TAO, ki poenostavi *fine-tuning* in optimizacijo standardnih modelov (YOLO, SSD, UNet, itd.) in jih samodejno pripravi za TensorRT.

Skratka, Jetson Orin Nano je idealen za **aplikacije računalniškega vida v realnem času** in za **interaktivne aplikacije, ki temeljijo na malih jezikovnih modelih (SLM)**.

Treniranje modela

Pri razvoju AI aplikacij, zlasti na Edge napravah, je faza treniranja modela ključna, čeprav poteka na zmogljivejši strojni opremi (ne na Jetsonu).

Tukaj je opis nekaterih orodij, in njihove vloge v ekosistemu strojnega učenja.

Orodja za treniranje in pripravo modelov AI

Roboflow (Poudarek na podatkih in pripravi)

Roboflow je platforma, ki se osredotoča na **pripravo, označevanje in upravljanje naborov podatkov** za računalniški vid (Computer Vision).

Roboflow ne trenira naborov globokega učenja (DL) na lastni strojni opremi, ampak močno **pospeši in poenostavi kritične korake pred treningom** in **poenostavi izvoz** za trening.

Upravljanje naborov podatkov	Centralizirano shranjevanje in sodelovanje pri naborih podatkov.
Označevanje (Labeling)	Intuitivno spletno orodje za označevanje slik (bounding boxi, segmentacija).
Priprava podatkov	Avtomatsko pretvarjanje naborov v različne formate (npr. YOLO, COCO, format PyTorch).
Povečanje podatkov (Augmentation)	Samodejno ustvarjanje novih podatkovnih primerov (z vrtenjem, spreminjanjem barve, zamegljenostjo), kar izboljša robustnost modela.
Roboflow Universe	Ogromno skladišče že označenih, odprtokodnih naborov podatkov, ki jih lahko takoj uporabite.

Ultralytics (Poudarek na arhitekturi modela)

Ultralytics je podjetje, ki je najbolj znano po razvoju in vzdrževanju serije modelov **YOLO (You Only Look Once)**, trenutno zlasti **YOLOv8**. YOLO je vodilna arhitektura za **prepoznavanje objektov v realnem času**.

Ultralytics zagotavlja **lahko dostopno in optimizirano ogrodje** za dejansko treniranje AI modelov za računalniški vid.

Arhitektura YOLO	Zelo hitri in natančni modeli, ki so idealni za **Edge AI** (nizka latenca).
Enostaven vmesnik	Zelo preprosti ukazni vmesnik (CLI) in Python API za zagon treninga na vaših podatkih.
Podpora za različne naloge	YOLOv8 podpira ne le prepoznavanje objektov, ampak tudi segmentacijo in pozo (pose estimation).
Treniranje in izvoz	Omogoča enostavno treniranje iz nič in izvoz modela v formate, optimizirane za inferenco (npr. ONNX), kar je ključen korak pred implementacijo na Jetson.

Kaggle (Poudarek na podatkih, kodeksih in skupnosti)

Kaggle je največja **spletna skupnost za podatkovne znanstvenike in inženirje strojnega učenja**. Je ekosistem, ki združuje podatkovne nabore, orodja, tekmovanja in okolja za sodelovanje.

Kaggle ponuja **brezplačno okolje za izvajanje kode** in **dostop do podatkov in predlog**.

Notebooks (Jupyter/Colab)	Vgrajeno okolje v brskalniku (Kaggle Notebooks) z brezplačnim dostopom do zmogljivih **GPU-jev** (zlasti za tekmovanja), kar je idealno za dejansko izvajanje treninga.
Nabori podatkov (Datasets)	Ogromna zbirka javno dostopnih naborov podatkov, ki jih lahko prenesete ali uporabite neposredno v svojem Notebooku.
Tekmovanja	Tekmovanja v podatkovni znanosti, ki ponujajo zapletene probleme in preizkušene rešitve (kodo) najboljših na svetu.
Skupnost in učenje	Možnost ogleda, učenja in kloniranja kode drugih uporabnikov, kar izjemno pospeši razvoj.

Povzetek

	fokus	uporaba pri **Jetsonom Orin Nano**
Roboflow	Priprava podatkov, označevanje, povečanje.	Pripravi podatke, ki jih nato trenirate (običajno z Ultralytics), in jih pripravi za izvoz za inferenco.
Ultralytics	Trening modela, zlasti YOLO arhitektur.	Zagotovi optimiziran, natreniran AI model, ki ga nato s **TensorRT** (na Jetsonu) pretvorite in zaženete.
Kaggle	Brezplačno okolje za trening, podatki, predloge kode.	Služi kot okolje, kjer dejansko poteka GPU trening modela, preden ga prenesete in optimizirate za Jetson.

Nasveti za treniranje modelov računalniškega vida

Trening zmogljivega modela je eden najpomembnejših korakov v celotnem procesu računalniškega vida. Tukaj je nekaj nasvetov, da zagotovite pravilen začetek.

1. Izberite pravi tip modela

Vsak tip projekta ustreza določeni vrsti modela. Priporočamo, da izberete **najenostavnejši tip modela, ki bo rešil vaš problem**. Enostavnejši modeli, kot je **klasifikacija**, so hitri za označevanje in izvajanje (inferenco), medtem ko bolj kompleksni modeli, kot je **segmentacija primerka** (*instance segmentation*), zahtevajo več časa za označevanje in so počasnejši pri inferenci. Za številne naloge zaznavanja je **zaznavanje objektov (omejevalna polja)** odlična začetna točka, saj ponuja dobro ravnovesje med kompleksnostjo in zmogljivostjo.

2. Uporabite podatke, ki ustrezajo produkcijskim podatkom

Modeli računalniškega vida poskušajo ujemati vzorce, ki jih »vidijo« v vaših učnih podatkih. Ena največjih napak je, da se označujejo podatki, ki so popolnoma drugačni od tistih, ki jih model pričakuje v produkciji. Na primer, če nameravate zaznati živali na fotografijah, posnetih z mobilnimi telefoni, ne uporabite rezultatov iskanja »mačka in pes« z Google Slik kot učnih podatkov. Vedno zbirajte slike iz **dejanskega okolja, s svetlobnimi pogoji in pod zornimi koti**, s katerimi se bo srečal vaš nameščeni model.

3. Začnite z majhnim naborom (Hitra iteracija)

Bolje je začeti z **majhnim naborom učnih podatkov (~50–100 slik)** in se osredotočiti na zagotavljanje, da so vaši podatki raznoliki. Če označujete okvirje iz videoposnetka, je najbolje vzeti eno sliko na 5 sekund (ali več) in vzorčiti iz več videoposnetkov. Manjši nabori podatkov omogočajo hitrejšo analizo rezultatov treninga in so hitrejši za ponovno označevanje. Ta pristop – **majhen, raznolik nabor podatkov** – vam omogoča hitro vzpostavitev močne izhodiščne točke.

4. Trenirajte pogosto in zgodaj

Številni uporabniki predolgo čakajo, preden trenirajo svoj prvi model. Bolje je **trenirati zgodaj in pogosto**. Uporabite lahko orodja za evalvacijo modelov, da vidite, kje je vaš model šibek, kar vam pomaga določiti prednost prihodnjega dela. Zgodnji trening skoraj takoj razkrije težave pri zbiranju ali strategiji označevanja podatkov.

5. Osredotočite se na podatke, ne na hiperparametre

Prilagajanje hiperparametrov (npr. hitrost učenja, velikost serije) in dodajanje številnih razširitev (*augmentations*) lahko močnemu modelu prinese nekaj dodatnih točk natančnosti, vendar slabega modela ne bo izboljšalo. Če želite izboljšati mAP modela za **10–20%**, se osredotočite na **podatke**. Verjetno imate napake pri označevanju (ali je vsaka slika označena točno tako, kot želite, da se model odzove?) ali pa potrebujete bolj raznolik nabor učnih podatkov. **Smeti noter, smeti ven (GIGO)** ostaja glavno pravilo umetne inteligence.

6. Popravite napake pri označevanju

Redno pregledujte svoje oznake. Tudi majhen odstotek napačnih ali nedoslednih oznak lahko znatno omeji delovanje modela. Orodja, ki vam omogočajo vizualizacijo napovedi modela poleg človeških oznak, lahko hitro prepoznajo področja, kjer so vaše oznake pomanjkljive ali dvoumnne.

Srečno pri ustvarjanju!

Prvi koraki

Vzemi si čas za Linux, Python in spoznavanje obstoječih primerov. Ko boš videl, da lahko s par vrsticami kode tvoja kamera "prepozna psa", bo motivacija ogromna.

1. Najprej osnove

- **Linux**: Jetson Orin Nano uporablja Ubuntu (večinoma različico 20.04 ali novejšo). Če nisi vešč Linuxa, se najprej nauči osnov: delo z datotekami, terminal, nameščanje paketov, uporaba apt.
- **Python**: To je najpogosteje uporabljen jezik za AI aplikacije. Osnove Python programiranja so skoraj nujne.

---

2. Postavitev sistema

- Prenesi in namesti **JetPack SDK** (na voljo v Nvidia SDK Manager). JetPack vključuje gonilnike, CUDA, TensorRT in knjižnice, ki so osnova za AI razvoj.
- Prepričaj se, da imaš delujoč dostop do interneta, kamere (če jo boš uporabljal), tipkovnico in zaslon.

---

3. Prvi poskusi z AI

Za začetek ni treba trenirati svojih modelov – lahko uporabiš že pripravljene modele:

- **Nvidia NGC** (Nvidia GPU Cloud) ponuja že pripravljene modele za prepoznavo slik, govora, video analitiko …
- Poskusi z **jetson-inference** knjižnico (https://github.com/dusty-nv/jetson-inference). Ima enostavne primere:
    - Prepoznava predmetov s kamero (*object detection*).
    - Razvrščanje slik (*image classification*).
    - Segmentacija slike.

---

4. Orodja za učenje

- **Python + PyTorch**: glavno orodje za učenje in izvajanje AI modelov. Na Jetsonu lahko uporabljaš optimizirano različico.
- **TensorRT**: knjižnica, ki modele pohitri za delovanje na Jetson platformi.
- **Jupyter Notebook**: zelo uporabno za interaktivno učenje in testiranje.

---

5. Tvoja prva aplikacija

Predlagam en preprost projekt:

- Priključi USB kamero ali CSI kamero.
- Naloži *jetson-inference*.
- Naredi aplikacijo, ki v živo prepoznava predmete pred kamero (npr. ljudi, živali, predmete).
- Kasneje lahko dodaš:
- štetje ljudi v prostoru,
- zaznavanje gibanja,
    - glasovno povratno informacijo.

"Prepoznavanje predmetov v živo s kamero na Jetson Orin Nano".

To je najlažji uvod, ker združuje osnovne korake: delo z Jetsonom, Python kodo in uporabo že pripravljenih AI modelov.

🚀 Vodič: Prvi projekt na Jetson Orin Nano

1. Priprava okolja

1. Zaženi Jetson Orin Nano z nameščenim **JetPack SDK**
   (če še ni, ga namestiš prek Nvidia SDK Manager na PC-ju).
2. Posodobi sistem:

```
sudo apt update && sudo apt upgrade -y
```

3. Namesti osnovne pakete:

```
sudo apt install git python3-pip cmake -y
```

---

2. Namestitev *jetson-inference*

To je Nvidia-jev demo paket z vnaprej naučenimi modeli.

```
cd ~
git clone --recursive https://github.com/dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ..
make -j$(nproc)
sudo make install
sudo ldconfig
```

Med nameščanjem boš vprašan, ali želiš prenesti pred-pripravljene modele – izberi **DA** (to bo olajšalo prvi zagon).

---

3. Preizkus primera (brez lastne kode)

Za testiranje, da vse deluje, zaženi:

```
cd ~/jetson-inference/build/aarch64/bin
./imagenet.py images/jellyfish.jpg images/test_output.jpg
```

- To vzame sliko jellyfish.jpg in jo razvrsti v kategorijo.
- Če boš odprl images/test_output.jpg, boš videl napoved z oznako.

---

4. Prepoznavanje predmetov v živo s kamero

Če imaš priključeno USB ali CSI kamero, zaženi:

```
./detectnet.py --camera=/dev/video0
```

To bo odprlo okno in prikazovalo video v živo s prepoznanimi predmeti
(npr. oseba, steklenica, pes …).

Če imaš **CSI kamero** (tista na MIPI konektorju), uporabi:

```
./detectnet.py --camera=csi://0
```

5. Napiši svojo Python kodo

Če želiš zagnati preprosto kodo za prepoznavo predmetov:

```
import jetson.inference
import jetson.utils

# naloži model
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

# uporabi kamero (0 = /dev/video0)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

while display.IsStreaming():
    img = camera.Capture()
    detections = net.Detect(img)
    print("Zaznani predmeti:")
    for d in detections:
        print(net.GetClassDesc(d.ClassID))
    display.Render(img)
```

Shrani kot `my_detection.py` in zaženi:

```
python3 my_detection.py
```

---

6. Razširi projekt

Ko to deluje, lahko dodaš:

- štetje ljudi v kadru,
- shranjevanje zaznanih slik,
- glasovno obvestilo ob zaznavi (z espeak).

Razširitev projekta

to bo že tvoj prvi pravi AI projekt na Jetson Orin Nano:

kamera bo zaznavala predmete in posebej štela ljudi v kadru. Ta projekt ti bo pokazal:

- kako zagnati AI model na Jetsonu,
- kako brati podatke iz kamere,
- kako jih prikazati in obdelati v Pythonu.

Koda: štetje ljudi pred kamero: people_counter.py

```
import jetson.inference
import jetson.utils
import time

# Naložimo model za detekcijo (SSD-Mobilenet-V2)
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

# Kamera (0 = USB kamera na /dev/video0; za CSI uporabi "csi://0")
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

while display.IsStreaming():
    # zajemi sliko
    img = camera.Capture()

    # detekcija predmetov
    detections = net.Detect(img)

    # števec ljudi
    people_count = 0
    for d in detections:
        class_name = net.GetClassDesc(d.ClassID)
        if class_name.lower() == "person":
            people_count += 1

    # izpis v terminal
    print(f"👤 Število ljudi v kadru: {people_count}")

    # izris slike in števca
    display.Render(img)
    display.SetStatus(f"People Counter | Detected: {people_count}")

    # počakaj malo (da ne preobremeni sistema)
    time.sleep(0.1)
```

▶ Zagon

```
python3 people_counter.py
```

V terminalu boš videl:

👤 Število ljudi v kadru: 0

👤 Število ljudi v kadru: 1

👤 Število ljudi v kadru: 2

V video oknu bo pisalo število zaznanih oseb.

Možne nadgradnje

- Shrani sliko, kadar zazna več kot X ljudi (npr. nad 3).
- Dodaj glasovno opozorilo z espeak:

```
sudo apt install espeak -y
```

in v pyton kodi:

```
import os
if people_count > 0:
    os.system(f'espeak "Detected {people_count} people"')
```

Razširitev projekta

Vključi štetje tudi za druge objekte (npr. pse, avtomobile).

To je že uporaben mini-sistem za **štetje ljudi** – lahko ga uporabiš v učilnici, pisarni ali celo kot osnova za pametni varnostni sistem. Tu je **nadgradnja kode**, kjer bo kamera ob zaznavi človeka **shranila sliko** v mapo (npr. detections/)?

```
import jetson.inference
import jetson.utils
import time
import os

# Naložimo model za detekcijo
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

# Kamera (USB kamera = /dev/video0, CSI kamera = csi://0)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

# Ustvari mapo za shranjevanje slik (če še ne obstaja)
save_dir = "detections"
os.makedirs(save_dir, exist_ok=True)

while display.IsStreaming():
    # zajemi sliko
```

```
img = camera.Capture()

# detekcija predmetov
detections = net.Detect(img)

# števec ljudi
people_count = 0
for d in detections:
    class_name = net.GetClassDesc(d.ClassID)
    if class_name.lower() == "person":
        people_count += 1

# izpis v terminal
print(f"👤 Število ljudi v kadru: {people_count}")

# če je zaznana oseba, shrani sliko
if people_count > 0:
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    filename = os.path.join(save_dir, f"people_{timestamp}.jpg")
    jetson.utils.saveImageRGBA(filename, img, img.width, img.height)
    print(f"💾 Slika shranjena: {filename}")

# izris slike in status
display.Render(img)
display.SetStatus(f"People Counter | Detected: {people_count}")

time.sleep(0.1)
```

▶ Zagon

```
python3 people_counter_save.py
```

- V terminalu boš videl štetje ljudi.
- Če kamera zazna osebo, boš dobil obvestilo:
- 💾 Slika shranjena: **detections/people_20250927-153210.jpg**
- Slike se bodo kopičile v mapi **detections/.**

Nadgradnje (če te bo zamikalo):

- Dodaj omejitev – shrani samo vsako 5. sekundo, da ne bo preveč slik.
- Naredi, da shrani **samo, če je zaznan nov človek** (ne vsako sličico).
- Dodaj e-poštno obvestilo ob novi zaznavi (malo zahtevnejši korak).

Koda: People Counter + Shranjevanje slik (vsakih 5 sekund)

Shranjeno v datoteki *people_counter_save_interval.py*

```
import jetson.inference
import jetson.utils
import time
import os

# Naložimo model za detekcijo
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

# Kamera (USB kamera = /dev/video0, CSI kamera = csi://0)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

# Ustvari mapo za shranjevanje slik (če še ne obstaja)
save_dir = "detections"
os.makedirs(save_dir, exist_ok=True)

# Shrani vsaj vsakih X sekund
SAVE_INTERVAL = 5
last_save_time = 0

while display.IsStreaming():
    # zajemi sliko
    img = camera.Capture()

    # detekcija predmetov
    detections = net.Detect(img)

    # števec ljudi
    people_count = 0
    for d in detections:
        class_name = net.GetClassDesc(d.ClassID)
        if class_name.lower() == "person":
            people_count += 1

    # izpis v terminal
    print(f"👤 Število ljudi v kadru: {people_count}")

    # shrani samo, če je oseba in je minilo dovolj časa
    current_time = time.time()
    if people_count > 0 and (current_time - last_save_time) >= SAVE_INTERVAL:
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = os.path.join(save_dir, f"people_{timestamp}.jpg")
        jetson.utils.saveImageRGBA(filename, img, img.width, img.height)
        print(f"💾 Slika shranjena: {filename}")
        last_save_time = current_time

    # izris slike in status
```

```
    display.Render(img)
    display.SetStatus(f"People Counter | Detected: {people_count}")

    time.sleep(0.1)
```

▶ Zagon

```
python3 people_counter_save_interval.py
```

- Če kamera zazna človeka, bo shranila **največ eno sliko vsakih 5 sekund**.
- Če v kadru ni nikogar, se slike ne shranjujejo.

bolj pametnobi bilo, da shrani sliko **samo takrat, ko se spremeni število ljudi v kadru**.
Tako ne boš imel nepotrebnih ponavljajočih slik.

Razširitev projekta

People Counter + Shranjevanje samo ob spremembi

Koda v datoteki: **`people_counter_save_change.py`**

```
import jetson.inference
import jetson.utils
import time
import os

# Naložimo model za detekcijo
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

# Kamera (USB kamera = /dev/video0, CSI kamera = csi://0)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

# Ustvari mapo za shranjevanje slik (če še ne obstaja)
save_dir = "detections"
os.makedirs(save_dir, exist_ok=True)

# Prejšnje število ljudi
last_people_count = -1

while display.IsStreaming():
    # zajemi sliko
    img = camera.Capture()

    # detekcija predmetov
    detections = net.Detect(img)

    # števec ljudi
    people_count = 0
    for d in detections:
        class_name = net.GetClassDesc(d.ClassID)
```

```
        if class_name.lower() == "person":
            people_count += 1

    # izpis v terminal
    print(f"👤 Število ljudi v kadru: {people_count}")

    # shrani sliko samo, če se je spremenilo število ljudi
    if people_count != last_people_count:
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = os.path.join(save_dir, f"people_{people_count}_{timestamp}.jpg")
        jetson.utils.saveImageRGBA(filename, img, img.width, img.height)
        print(f"💾 Sprememba zaznana ({last_people_count} → {people_count}). Slika
shranjena: {filename}")
        last_people_count = people_count

    # izris slike in status
    display.Render(img)
    display.SetStatus(f"People Counter | Detected: {people_count}")

    time.sleep(0.1)
```

▶ Zagon

```
python3 people_counter_save_change.py
```

- Če se v kadru spremeni število ljudi (npr. 0 → 1, 1 → 3, 3 → 0), se shrani nova slika.
- Če ostane enako (npr. ves čas 1 oseba), ne bo shranjevalo več slik.

Prednosti te različice

- Manj nepotrebnih slik.
- Dobro za primerjavo, kako se **spreminja dogajanje** (npr. prihod ali odhod ljudi).
- Imena datotek imajo zapisano tudi novo število ljudi (npr. people_2_20250927-174215.jpg).

Računalniški vid (Computer Vision)

CILJ: Prepoznavanje objektov v videu (v realnem času)

1. Osnova: Jetson + JetPack

Jetson Orin Nano mora imeti nameščen **JetPack SDK** (prek Nvidia SDK Manager).

Ta paket že vsebuje:

- **CUDA** (za paralelno računanje na GPU),
- **TensorRT** (za pohitritev AI modelov),
- **DeepStream SDK** (za video obdelavo in zaznavo objektov),
- **OpenCV**,
- in podporo za **kamere**.

Če imaš JetPack že nameščen (kar verjetno imaš, glede na prejšnje korake), si pripravljen.

---

2. Izbira modela za prepoznavanje objektov

Najpogosteje uporabljen je eden od teh modelov:

Model	Opis	Hitrost na Jetsonu
SSD-Mobilenet-v2	Hiter, lahek, prepoznava osnovne objekte.	⚡⚡⚡ (zelo hitro)
YOLOv5 / YOLOv8 / YOLOv11	Sodobni modeli za detekcijo, dobra natančnost.	⚡⚡ (hitro z TensorRT)
DetectNet v2 (NVIDIA)	Nvidia-jev model, optimiziran za Jetson.	⚡⚡⚡
Custom YOLO model	Tvoj model, naučen na tvojih slikah.	⚡ (odvisno od kompleksnosti)

Za začetek priporočam **SSD-Mobilenet-v2** ali **YOLOv8n (nano)**.

---

3. Praktični primer – *Real-time Object Detection*

Če želiš preprost zagon:

```
cd ~/jetson-inference/build/aarch64/bin
./detectnet.py --camera=/dev/video0
```

To bo v živo prikazovalo, kaj kamera "vidi", in prepoznavalo objekte (oseba, pes, stol, avto …).
Deluje s privzetim modelom *ssd-mobilenet-v2.*

Če želiš uporabiti YOLO model:

```
./detectnet.py --network=yolov8n --camera=/dev/video0
```

4. Uporabi Python API za lastne projekte

Znotraj Python-a lahko zgradiš svojo aplikacijo (kot si že delal prej):

```
import jetson.inference
import jetson.utils

net = jetson.inference.detectNet("yolov8n", threshold=0.5)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

while display.IsStreaming():
    img = camera.Capture()
    detections = net.Detect(img)
    for d in detections:
        print(net.GetClassDesc(d.ClassID), d.Confidence)
    display.Render(img)
```

To ti omogoča:

- da v realnem času prepoznavaš objekte,
- beležiš rezultate (npr. šteješ avtomobile, ljudi),
- in jih po želji shraniš ali kombiniraš z drugimi senzorji.

---

5. Če želiš resno obdelavo videa (več kamer, tokov, IP kamere)

Uporabi **Nvidia DeepStream SDK** – to je uradni "video analytics" sistem:

- Podpira RTSP kamere, USB, CSI.
- Lahko obdeluje več video tokov hkrati.
- Podpira **multi-model pipeline** (npr. detekcija + sledenje + razpoznavanje obrazov).
- Ima grafični prikaz in ogromno primerov.

Za namestitev:

```
sudo apt install deepstream-6.4
```

Za zagon primera:

```
deepstream-app -c /opt/nvidia/deepstream/deepstream/samples/configs/deepstream-app/source1_primary_detector.txt
```

---

6. Napredne možnosti (ko boš želel napredovati)

Ko osvojiš osnove, lahko:

- **Treniraš svoj YOLO model** (na PC-ju) za specifične objekte (npr. učence, avtomobile, šolske predmete).
- Model preneseš na Jetson in ga optimiziraš z **TensorRT**.
- Dodajaš funkcije: sledenje, štetje, alarm, zapis v bazo ipd.

Namen	Orodje / Model
Hitri začetek	jetson-inference (SSD-Mobilenet)
Večja natančnost	YOLOv8 (TensorRT optimiziran)
Industrijska rešitev	DeepStream SDK
Tvoj lasten model	YOLOv8 ali DetectNet, treniran na PC
Analiza videa v Pythonu	jetson.inference + jetson.utils

YOLOv8 (TensorRT optimiziran)

To bo tvoj prvi korak k "profesionalnemu" računalniškemu vidu.

Gremo korak za korakom skozi namestitev in zagon YOLOv8 na tvojem Jetson Orin Nano, da boš imel natančno prepoznavanje objektov v videu.

CILJ

Uporabiti **YOLOv8 (Ultralytics)** model za **real-time prepoznavanje objektov** preko kamere, neposredno na Jetson Orin Nano.

---

1. Priprava okolja

Najprej posodobi sistem:

```
sudo apt update && sudo apt upgrade -y
```

Namesti osnovne knjižnice:

```
sudo apt install python3-pip python3-opencv git -y
```

Posodobi pip:

```
python3 -m pip install --upgrade pip
```

---

2. Namesti YOLOv8 (Ultralytics)

Na Jetsonu lahko brez težav namestiš Python knjižnico YOLOv8:

```
pip install ultralytics
```

Po uspešni namestitvi preveri:

```
yolo version
```

Če dobiš izpis različice (npr. Ultralytics YOLOv8.1.2), si pripravljen.

---

3. Test: prepoznava objektov na sliki

Prenesi testno sliko:

```
wget https://ultralytics.com/images/zidane.jpg -O test.jpg
```

Zaženi YOLOv8:

```
yolo predict model=yolov8n.pt source=test.jpg show=True
```

Model yolov8n.pt je **nano različica** (najmanjša in najhitrejša).

V oknu se ti pokaže slika z označenimi objekti.

Če to deluje – odlično!

---

4. Prepoznava objektov z živo kamero

Za USB kamero:

```
yolo predict model=yolov8n.pt source=0 show=True
```

Za CSI kamero (tista na MIPI priključku):

```
yolo predict model=yolov8n.pt source="csi://0" show=True
```

To bo odprlo video okno in prikazovalo zaznane objekte (osebe, avtomobile, pse …) v realnem času.

---

5. Pohitritev z GPU (TensorRT / CUDA)

YOLOv8 že uporablja **PyTorch + CUDA** na Jetsonu, če je JetPack pravilno nameščen.
Preveri, da se CUDA uporablja:

```
python3 -c "import torch; print(torch.cuda.is_available())"
```

Če dobiš True, tvoj GPU deluje.
Za hitrejše delovanje lahko model **pretvoriš v TensorRT**:

```
yolo export model=yolov8n.pt format=engine
```

To ustvari datoteko yolov8n.engine — to je TensorRT različica, ki deluje *do 3–5× hitreje* na Jetsonu.

Nato jo uporabiš:

```
yolo predict model=yolov8n.engine source=0 show=True
```

6. Shranjevanje rezultatov

Če želiš, da YOLO shrani video z zaznavami:

```
yolo predict model=yolov8n.engine source=0 save=True
```

To ustvari mapo `runs/detect/predict/` s posnetkom.

7. (Neobvezno) – uporaba v tvoji Python kodi

Če želiš vgraditi YOLOv8 v svoj Python projekt:

```
from ultralytics import YOLO

model = YOLO("yolov8n.engine")  # ali .pt, če še nisi pretvoril

results = model.predict(source=0, show=True)  # kamera 0
```

YOLO ti vrne seznam zaznav (objekti, koordinate, zaupanja vrednost), ki jih lahko nato analiziraš, shraniš ali povežeš z drugimi funkcijami (štetje, alarm, sledenje itd.).

8. Nasveti za zmogljivost

Da YOLOv8 deluje gladko na Orin Nano:

Nastavitev	**Priporočilo**
Uporabi model yolov8n.pt ali yolov8s.pt	večji modeli (m, l, x) so pretežki
Zmanjšaj ločljivost videa	npr. 640×480 ali manj
Pretvori model v TensorRT (.engine)	največji pospešek
Onemogoči prikaz (show=False) za hitrost	če ne rabiš videa v živo
Uporabi "headless" način	za avtomatsko obdelavo brez GUI

Treniranje lastnega modela YOLOv8

Nadaljujmo iz "uporabe vnaprej naučenega modela" → v **treniranje svojega YOLOv8 modela**.

To pomeni, da bomo lahko naučili Jetsona (ali PC) **prepoznavati točno tiste objekte**, ki jih želimo recimo: učence, učitelje, šolske predmete, živali, vozila, karkoli.

CILJ

Trenirati lasten YOLOv8 model (npr. *"prepoznaj zvezek, svinčnik in tablico"*) in ga nato uporabiti na Jetson Orin Nano.

Koraki postopka:

1. zbereš slike,
2. označiš,
3. treniraš YOLOv8,
4. preneseš model na Jetson,
5. uporabljaš za zaznavo v realnem času.

1. KJE TRENIRAŠ MODEL?

Treniranje je zelo zahtevno za Jetson Orin Nano — zato priporočam:

- Treniraj na PC-ju ali prenosniku z Nvidia GPU (če ga imaš).
- Nato prenesi naučen model (.pt ali .engine) na Jetson za izvajanje (*inference*).

Jetson je popoln za izvajanje modelov, ne pa za dolgotrajno treniranje.

Če nimaš močnejšega PC-ja, obstajajo možnosti v oblaku (Google Colab, Kaggle ipd.), lahko ti pripravim Colab skripto kasneje.

2. ZBIRKA PODATKOV (dataset)

Za YOLO potrebuješ slike + označbe.

a) Zberi slike:

- Posnemi slike tistih predmetov, ki jih želiš prepoznavati (npr. 200 slik zvezkov, 200 svinčnikov ...).
- Shrani jih v mapo, npr.:

```
dataset/
    ├── images/
        ├── train/
        ├── val/
        └── test/
```

b) Oznake (labels):

Vsaka slika mora imeti **.txt** datoteko z oznakami, kjer je zapisano:

<class_id> <x_center> <y_center> <width> <height>

Vse koordinate so v relativnih vrednostih (0–1).
Lahko jih ustvariš ročno z orodji, kot so:

- **LabelImg**

```
pip install labelImg
labelImg
```

- **Roboflow** (spletno orodje – brezplačno za manjše projekte)
- **Makesense.ai** (https://www.makesense.ai)

Vsak predmet dobi svojo številko (npr. 0=zvezek, 1=svinčnik, 2=tablica).

3. Konfiguracija datoteke data.yaml

Ustvari datoteko data.yaml (npr. v mapi projekta):

```
train: dataset/images/train
val: dataset/images/val
test: dataset/images/test

nc: 3
names: ['zvezek', 'svinčnik', 'tablica']
```

4. Treniranje YOLOv8 modela

V terminalu (na PC ali strežniku z GPU):

```
pip install ultralytics
yolo train model=yolov8n.pt data=data.yaml epochs=50 imgsz=640
```

Pomen parametrov:

- model=yolov8n.pt → osnovni "nano" model
- epochs=50 → koliko krogov učenja
- imgsz=640 → velikost vhodne slike
- data=data.yaml → datoteka s potmi do podatkov

YOLO bo samodejno ustvaril mapo runs/detect/train/ z rezultati in najboljšim modelom:

```
runs/detect/train/weights/best.pt
```

5. Testiranje modela

Ko treniranje končaš:

```
yolo predict model=runs/detect/train/weights/best.pt source=dataset/images/val
```

Preveri rezultate (označeni objekti naj se ujemajo s tvojimi pričakovanji).

6. Prenos modela na Jetson

Na Jetson Orin Nano:

1. Kopiraj model best.pt (prek USB, SCP ali mreže).
2. Namesti YOLOv8 (če še ni):

```
pip install ultralytics
```

3. Zaženi zaznavo:

```
yolo predict model=best.pt source=0 show=True
```

7. (Neobvezno) Pretvori v TensorRT za hitrost

Na Jetsonu lahko svoj model pospešiš:

```
yolo export model=best.pt format=engine
```

In nato:

```
yolo predict model=best.engine source=0 show=True
```

To bo delovalo do 3–5× hitreje.

8. Nasveti iz prakse

	Priporočilo
Število slik	vsaj 200–300 na razred
Kakovost slik	raznolike (svetloba, kot, ozadje)
Ločljivost	640×640 ali 416×416
Preverjanje	10–20 % slik naj bo za validacijo
Pospešitev	TensorRT in yolov8n (nano) model
Shranjevanje	dodaj save=True za posnetke zaznav

Uporaba Google Colab za treniranje modela YOLOv8

To je naslednji **logični in praktični korak** — uporabili bomo **Google Colab**, da boš lahko treniral svoj YOLOv8 model v oblaku (brez potrebe po močnem računalniku).

CILJ

Uporabiti **Google Colab** za treniranje lastnega YOLOv8 modela, potem pa **prenesti naučen model (best.pt)** na tvoj **Jetson Orin Nano**.

1. KAJ POTREBUJEŠ

Preden začnemo, pripravi:

1. Google račun (za dostop do Google Colab).
2. Zbirko slik (.jpg, .png) in oznak (.txt) v YOLO formatu.
3. Datoteko data.yaml, kot prej:

```
train: /content/dataset/images/train
val: /content/dataset/images/val
nc: 3
names: ['zvezek', 'svinčnik', 'tablica']
```

---

2. USTVARI NOVO NOTEBOOK DATOTEKO

Odpri https://colab.research.google.com
in klikni **"New Notebook"**.

---

3. VSEBINA CELOTNE SKRIPTE (kopiraj v Colab)

```
# --- 1. Preveri GPU ---
!nvidia-smi

# --- 2. Namesti YOLOv8 (Ultralytics) ---
!pip install ultralytics==8.2.50

from ultralytics import YOLO
import os

# --- 3. Naloži svoj dataset (ZIP datoteka) ---
# Npr. če imaš ZIP datoteko 'dataset.zip' v Google Drive:
from google.colab import drive
drive.mount('/content/drive')

# Kopiraj dataset v Colab okolje
!cp /content/drive/MyDrive/dataset.zip /content/

# Razpakiraj
!unzip -q dataset.zip -d /content/dataset
```

```
# --- 4. Preveri strukturo ---
!ls /content/dataset

# --- 5. Ustvari YOLOv8 model in treniraj ---
model = YOLO("yolov8n.pt")  # Nano model (najmanjši in najhitrejši)

results = model.train(
    data="/content/dataset/data.yaml",  # pot do tvoje YAML datoteke
    epochs=50,                          # število epoch (lahko spremeniš)
    imgsz=640,                          # velikost slik
    batch=16,                           # velikost batcha
    project="runs/train",               # izhodna mapa
    name="my_model",                    # ime projekta
)

# --- 6. Preveri rezultate ---
!ls runs/train/my_model/weights/

# --- 7. Prenesi naučen model ---
from google.colab import files
files.download('runs/train/my_model/weights/best.pt')
```

🗁 4. STRUKTURA DATASETA (ZIP datoteka)

Tvoj .zip mora imeti takšno strukturo:

dataset.zip

```
├── data.yaml
├── images/
│   ├── train/
│   │   ├── img1.jpg
│   │   ├── img2.jpg
│   └── val/
│       ├── val1.jpg
│       ├── val2.jpg
├── labels/
│   ├── train/
│   │   ├── img1.txt
│   ├── val/
│       ├── val1.txt
```

⚠ Pazljivo: Imena .txt datotek morajo biti enaka imenom slik.

5. PO TRENIRANJU

- V mapi runs/train/my_model/weights/ dobiš:
    - best.pt – **tvoj naučen model**

- last.pt – model iz zadnje epohe

Klikneš v Colabu na *download arrow* ali uporabiš:

files.download('runs/train/my_model/weights/best.pt')

---

6. UPORABA NA JETSONU

Ko best.pt preneseš na Jetson:

```
yolo predict model=best.pt source=0 show=True
```

Če želiš večjo hitrost:

```
yolo export model=best.pt format=engine
yolo predict model=best.engine source=0 show=True
```

NASVETI IZ PRAKSE

	Priporočilo
GPU v Colabu	Nastavi: **Runtime → Change runtime type → GPU (T4 ali A100)**
Slike	Vsaj 100–200 na razred
Epochs	50–100 za stabilen model
Velikost	yolov8n za Jetson Nano
Validacija	Uporabi 10–20 % slik za val
Shrani dataset v Drive	Lažje ga naložiš večkrat

Treniranje modela na vnaprej pripravljenih podatkih (datasets)

Kako trenirat model na že vnaprej pripravljenih podatkih? Aha vidi se, da že razumeš, **kaj iščeš**: majhen, a uporaben **dataset za učenje YOLO modela** na vsakdanjih predmetih, brez da bi moral prenesti celoten (ogromen) COCO.

In ja — točno za to obstajajo **lahke različice COCO**, kot je **COCO8**, ki jo Ultralytics že vključuje kot učni primer.

CILJ

Uporabiti **enostaven dataset (COCO8)** ali podoben, za treniranje YOLOv8 modela, ki zna prepoznati vsakdanje objekte (npr. laptop, knjigo, telefon ...).

Kaj je COCO8

COCO8 je **mini različica** slavnega COCO dataseta:

- vsebuje **8 slik**,
- ima večino **klas iz originalnega COCO (80 razredov)**,
- namenjen je **testiranju, učenju in razumevanju procesa**, ne natančnosti.

To pomeni, da ga lahko uporabiš za:

- hitro preizkušnjo treniranja,
- preverjanje, da tvoj YOLO sistem deluje,
- demonstracijo delovanja na Jetsonu.

Kako uporabiti COCO8 v Colab ali lokalno

Če si že v **Google Colab** ali imaš YOLOv8 nameščen lokalno, zadošča ena vrstica:

```
yolo train model=yolov8n.pt data=coco8.yaml epochs=10 imgsz=640
```

Ultralytics samodejno prenese COCO8 dataset (v /content/datasets/coco8).

Struktura COCO8 dataseta

Ko ga YOLO prenese, izgleda tako:

```
datasets/coco8/
├── images/
│   ├── train/
│   └── val/
├── labels/
│   ├── train/
│   └── val/
└── data.yaml
```

Datoteka data.yaml vsebuje:

```
train: ./datasets/coco8/images/train
val: ./datasets/coco8/images/val

nc: 80
names:
  0: person
  1: bicycle
```

```
  2: car
  3: motorcycle
  ...
  63: laptop
  64: mouse
  65: remote
  66: keyboard
  67: cell phone
  68: microwave
  69: oven
  70: toaster
  71: sink
  72: refrigerator
  73: book
  74: clock
  75: vase
  76: scissors
  77: teddy bear
  78: hair drier
  79: toothbrush
```

Torej, vse razrede, ki si jih navedel, **vsebuje**.

Primer: treniranje s COCO8 v Colab

Če želiš preprosto trenirati na COCO8 in videti rezultate, v Colab vpiši:

```
!pip install ultralytics
from ultralytics import YOLO

# Prenesi in treniraj COCO8 dataset
model = YOLO('yolov8n.pt')
model.train(data='coco8.yaml', epochs=10, imgsz=640)
```

To bo preneslo dataset in izvedlo hitro treniranje z nekaj slikami, zgolj za učenje procesa.

Testiranje modela

Ko se treniranje konča, lahko testiraš z eno vrstico:

```
model.predict(source='https://ultralytics.com/images/zidane.jpg', show=True)
```

Ali pa z lokalno kamero (na Jetsonu):

```
yolo predict model=runs/train/exp/weights/best.pt source=0 show=True
```

Če želiš nekoliko večji dataset (še vedno lahek)

Če želiš nekaj bolj realnega od COCO8, a še vedno majhno:

Dataset	Opis	Velikost
COCO128	128 slik iz COCO z 80 razredi	~180 MB
VOC	Klasični PASCAL VOC (20 razredov)	~2 GB
Objects365 mini	600 kategorij, izrezani primeri	~1 GB
Roboflow "Everyday Objects"	zbirke uporabnikov, lahko izbereš razrede	po meri

COCO128 lahko naložiš enako preprosto:

```
yolo train model=yolov8n.pt data=coco128.yaml epochs=20 imgsz=640
```

Povzetek

Dataset	Namen	Vsebuje tvoje razrede?	Priporočeno za
COCO8	Testiranje, učenje	✅ Da	Začetek
COCO128	Hitra praksa	✅ Da	Kratek trening
COCO2017	Polni dataset (~25GB)	✅ Da	Resno treniranje
Roboflow	Po meri	✅ Ti izbereš	Vaje ali specifične projekte

Ollama server:

zagon kontainerja

sh

```
jetson-containers run $(autotag ollama) ollama run gemma3
```

sh

```
ollama run gemma3
```

nekaj docker ukazov:

```
docker ps -a
docker compose ps
docker inspect <id-docker>
npr.: docker inspect ollama
docker stop ollama
docker stats
docker system df
free -m

echo $container   - pove v katerem kontejnerju si
```