**DeepStream Inference Pipeline Documentation**

**1. Introduction**

This document provides step-by-step instructions for setting up and running inference on videos using the DeepStream framework with ONNX models inside a Docker container.

**Pipeline Overview**

1. **Input Data:**
   * **JSON Files** (stage-2-detections): These contain object detection results with bounding boxes and category labels.
   * **Video Files** (deepstream-videos): Corresponding videos where the detected objects appear.
   * **ONNX Model** (person\_fp\_classifier\_alexnet.onnx): A pre-trained **AlexNet** model for object classification.
2. **Processing Steps:**
   * Load the ONNX classification model.
   * Loop through each JSON file to extract object detection information (bounding boxes and category labels).
   * Find the corresponding video file.
   * Read the video frame by frame.
   * For each frame:
     + Extract the detected objects using bounding box coordinates.
     + Resize and normalize the object images to match **AlexNet** input requirements.
     + Pass them through the ONNX classification model.
     + Overlay inference results on the video.
   * Save the output video with the classification results.

**2. Prerequisites**

* **Operating System**: Linux (Ubuntu recommended)
* **Docker Installed**: Verify with docker --version
* **NVIDIA GPU & Drivers Installed**: Verify with nvidia-smi
* **DeepStream SDK** (v7.1)
* **ONNX Model** (e.g., person\_fp\_classifier\_alexnet.onnx)
* **JSON Detection Files** (e.g., event20241218104215119.det.json)
* **AVI Video Files** (e.g., event20241218104215119.avi)

**3. Setting Up the Environment**

**Step 1: Pull and Run DeepStream Docker Container**

sudo docker run --gpus all -it --name deepstream-container \

-v ~/deepstream\_project:/mnt/deepstream\_project \

deepstream-custom:v1 /bin/bash

**Step 2: Install Dependencies (If Required)**

Inside the container, install required packages:

pip install onnxruntime numpy opencv-python

**4. Running the Inference**

**Step 1: Navigate to the Config Folder**

cd /mnt/deepstream\_project/deepstream-configs

### ****Step 2: Verify Model and Data Availability****

Ensure the ONNX model and input data exist inside the container:

ls -lh /mnt/deepstream\_project/deepstream-configs/person\_fp\_classifier\_alexnet.onnx

ls -lh /mnt/deepstream\_project/deepstream-videos/

ls -lh /mnt/deepstream\_project/stage-2-detections/

**Step 3: Run the Inference Script**

python3 run\_inference.py

**5. Extracting Output Files**

**Step 1: Copy Output Videos to Local Machine**

docker cp deepstream-container:/mnt/deepstream\_project/output-videos ./output-videos

**Step 2: Change Ownership of Extracted Files**

sudo chown -R $USER:$USER ./output-videos

**Step 3: Play Output Video**

vlc ./output-videos/event20241230205933117\_output.avi

**6. Performance Benchmarking**

The script logs the following performance metrics:

* **Total Frames Processed**
* **Total Inference Time (seconds)**
* **Average Inference FPS**

To view these details:

tail -n 20 run\_inference.log

**7. Troubleshooting**

**Issue: Permission Denied for Video Files**

**Solution:**

sudo chown -R $USER:$USER ./output-videos

**Issue: No Display Available in Docker**

**Solution:** Run with xhost + to allow GUI applications inside Docker:

xhost +

Then start the container with:

docker run --gpus all -it --rm -e DISPLAY=$DISPLAY --network=host deepstream-custom:v1 /bin/bash

**Issue: Missing Dependencies**

**Solution:** Inside the container, install missing packages using:

pip install -r requirements.txt

**To re-enter an already running Docker container after exiting, use the following command:**

### ****If the container is stopped, start it first:****

1. **Check container status**

docker ps -a | grep deepstream-container

If the container is in "Exited" status, restart it:

1. **Restart the container**

docker start deepstream-container

1. **Re-enter the container**

docker exec -it deepstream-container /bin/bash

**8. Conclusion**

This document outlines the complete pipeline for running inference on DeepStream, managing video processing, and optimizing performance. Follow these steps to ensure smooth execution.

For further debugging, refer to DeepStream documentation or NVIDIA developer forums.

## ****Additional Commands****

### ****List Running Containers****

docker ps

### ****List All Containers (Including Stopped)****

docker ps -a

### ****Remove a Container****

docker rm deepstream-container

### ****Remove an Image****

docker rmi deepstream-custom:v1

### ****Delete All Containers****

docker rm $(docker ps -aq)

### ****Delete All Images****

docker rmi $(docker images -q)

### **Restart exited container**

**resume** your previously exited container with everything as-is — it **retains GPU access**, mounted volumes, and settings **from the original docker run**.

docker start -ai f663dead6823

### ****check if display access is working inside your container**,**

echo $DISPLAY

xeyes

To **generate an updated requirements.txt** file including both **pip and mim installed packages**, follow these steps:

### ****Step 1: Generate**** requirements.txt ****with pip****

Run the following command inside the container:

pip freeze > /mnt/deepstream\_project/requirements.txt

This will save all installed Python packages to requirements.txt.

### ****Step 2: Append MIM Installed Packages****

MIM (Model Installer Manager) is used for OpenMMLab repositories. Run the following:

mim list > /mnt/deepstream\_project/mim\_packages.txt

Now, manually append the required packages to requirements.txt:

cat /mnt/deepstream\_project/mim\_packages.txt >> /mnt/deepstream\_project/requirements.txt

To **update the existing deepstream-custom:v1 image** with the new changes made inside the running container, follow these steps:

**1️⃣ Commit Changes to the Same Image (v1)**

If you want to **update deepstream-custom:v1** with your latest changes, run:

docker commit deepstream-container deepstream-custom:v1

🔹 This will overwrite the deepstream-custom:v1 image with the current state of the container.

**2️⃣ Verify the Image Update**

To check if the update was applied, list all Docker images:

docker images

You should see deepstream-custom:v1 with a new updated timestamp.

**3️⃣ Restart the Container from Updated Image**

If you need to **stop and restart** the container from the updated v1 image:

docker stop deepstream-container

docker rm deepstream-container

docker run --gpus all -it --name deepstream-container \

-v ~/deepstream\_project:/mnt/deepstream\_project \

deepstream-custom:v1 /bin/bash

🔹 This ensures you are using the latest version of the deepstream-custom:v1 image.

**4️⃣ Verify Libraries are Installed**

Once inside the container, check if the new dependencies are available:

python3 -c "import torch; import mmyolo; import mmengine; print('All dependencies installed')"

### Steps to Run and Enter the Existing Container:

1. **Restart the Container**  
   Run the following command to start the container:

bash

CopyEdit

docker start deepstream-container

1. **Enter the Container**  
   Once the container is running, you can enter it using:

bash

CopyEdit

docker exec -it deepstream-container /bin/bash

or, if it's an Ubuntu-based container:

bash

CopyEdit

docker exec -it deepstream-container bash

If you face any issues, check the container logs for debugging:

bash

CopyEdit

docker logs deepstream-container

docker start wonderful\_solomon

docker exec -it wonderful\_solomon

# Running deepstream

## convert onnx to tensorRT

Command   
/usr/src/tensorrt/bin/trtexec \

--onnx=/mnt/deepstream\_project/deepstream-configs/person\_fp\_classifier\_alexnet.onnx \

--saveEngine=/mnt/deepstream\_project/deepstream-configs/person\_fp\_classifier\_alexnet.trt \

--fp16 --shapes=input:4x3x224x224

You can print all ONNX layers inside your container:

python3 -c "import onnx; model = onnx.load('nexus-safh-2\_2025-01-31-13-27-45\_yolo8-epoch\_800.pth.FP32\_b1\_jacob-pc.onnx'); print([n.op\_type for n in model.graph.node])"

* If the output contains **unsupported layers** like Resize, GatherND, or NonMaxSuppression, TensorRT **might not support them**.

**MMDeploy container**

docker run -it \

-v ~/Downloads/michael-mmyolo-playground:/workspace \

--gpus all \

--name mmdeploy\_test \

wonderful\_solomon:v1 bash

docker run -it \

-v ~/Downloads/michael-mmyolo-playground:/workspace \

--gpus all \

--name mmdeploy\_test \

wonderful\_solomon:v1 bash

python3 /workspace/deepstream\_project/Scratchpad/export\_tensorrt/mmdeploy/tools/deploy.py \

/workspace/deepstream\_project/Scratchpad/export\_tensorrt/michael-mmyolo-playground/deploy\_trt.py \

/workspace/deepstream\_project/Scratchpad/export\_tensorrt/michael-mmyolo-playground/custom\_train.py \

/workspace/deepstream\_project/Scratchpad/export\_tensorrt/training-output/best\_coco\_bbox\_mAP\_epoch\_788.pth \

/workspace/deepstream\_project/Scratchpad/export\_tensorrt/test.jpg \

--work-dir /workspace/deepstream\_project/Scratchpad/export\_tensorrt/training-output/exports \

--device cuda:0 \

--log-level INFO \

--show \

--dump-info

# Converting YOLOv8 & AlexNet ONNX Models to TensorRT for DeepStream

This guide covers the entire process of:

* Setting up a **Docker container with DeepStream**
* Installing **Conda, PyTorch, and TensorRT**
* **Converting YOLOv8 and AlexNet models** from PyTorch .pth to ONNX and then to TensorRT .engine
* Running **DeepStream inference** on the optimized models

## ****1. Basic Docker Commands****

### ****Check if Docker is Installed****

docker --version

### ****List Running Containers****

docker ps

### ****List All Containers (Running & Stopped)****

docker ps -a

### ****List Docker Images****

docker images

## ****2. Pull & Run the DeepStream Docker Container****

### ****Pull DeepStream 6.4****

docker pull nvcr.io/nvidia/deepstream:6.4-triton-multiarch

### ****Start the Container with GPU Access****

xhost +local:docker

docker run --gpus all -it \

-e DISPLAY=$DISPLAY \

-v /tmp/.X11-unix:/tmp/.X11-unix \

-v /home/prakash/deepstream6.4:/working \

nvcr.io/nvidia/deepstream:6.4-triton-multiarch /bin/bash

✅ Inside the container, your **host directory /home/prakash/deepstream6.4/** is mapped to /working/.

## ****3. Setting Up Conda (Python Environment)****

### ****Install Miniconda****

wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86\_64.sh

bash Miniconda3-latest-Linux-x86\_64.sh

Then, restart the terminal and **activate Conda**:

source ~/.bashrc

### ****Create a Conda Environment for PyTorch****

conda create --name torch1.9 python=3.8 -y

conda activate torch1.9

## ****4. Install PyTorch & Dependencies****

### ****Install PyTorch 1.13 with CUDA 11.7****

pip install torch==1.13.0 torchvision==0.14.0 torchaudio==0.13.0 -f https://download.pytorch.org/whl/cu117

### ****Install MMDeploy & MMCV****

pip install mmdeploy==1.3.1

pip install mmcv-full==2.0.0 -f https://download.openmmlab.com/mmcv/dist/cu117/torch1.13/index.html

## ****5. Convert YOLOv8 Model to ONNX****

### ****Convert**** .pth ****to**** .onnx

python3 export.py --weights /working/training-output/best\_coco\_bbox\_mAP\_epoch\_788.pth --img 640 --batch 1 --dynamic --simplify --opset 11 --device cuda

✅ Output file:

/working/training-output/best\_coco\_bbox\_mAP\_epoch\_788.onnx

## ****6. Convert YOLOv8 ONNX to TensorRT****

### ****Convert FP32 Engine****

trtexec --onnx=/working/training-output/best\_coco\_bbox\_mAP\_epoch\_788.onnx \

--saveEngine=/working/training-output/yolov8\_fp32.engine \

--workspace=4096 --verbose

### ****Convert FP16 Engine****

trtexec --onnx=/working/training-output/best\_coco\_bbox\_mAP\_epoch\_788.onnx \

--saveEngine=/working/training-output/yolov8\_fp16.engine \

--workspace=4096 --verbose --fp16

✅ Output:

/working/training-output/yolov8\_fp32.engine

/working/training-output/yolov8\_fp16.engine

## ****7. Convert AlexNet ONNX to TensorRT****

### ****Convert FP32 AlexNet Engine****

trtexec --onnx=/working/training-output/person\_fp\_classifier\_alexnet.onnx \

--saveEngine=/working/training-output/person\_fp\_classifier\_fp32.engine \

--workspace=4096 --verbose

### ****Convert FP16 AlexNet Engine****

trtexec --onnx=/working/training-output/person\_fp\_classifier\_alexnet.onnx \

--saveEngine=/working/training-output/person\_fp\_classifier\_fp16.engine \

--workspace=4096 --verbose --fp16

✅ Output:

/working/training-output/person\_fp\_classifier\_fp32.engine

/working/training-output/person\_fp\_classifier\_fp16.engine

## ****8. Verify the TensorRT Engine Files****

trtexec --loadEngine=/working/ training-output/exports/end2end.engine --verbose

trtexec --loadEngine=/working/training-output/yolov8\_fp16.engine --verbose

trtexec --loadEngine=/working/training-output/person\_fp\_classifier\_fp32.engine --verbose

trtexec --loadEngine=/working/training-output/person\_fp\_classifier\_fp16.engine –verbose

/home/prakash/deepstream6.4/training-output/exports/end2end.engine

## ****9. Save the Docker Container Before Exiting****

### ****Find the Running Container ID****

docker ps

Example:

CONTAINER ID IMAGE COMMAND STATUS NAMES

efa340e929d6 nvcr.io/nvidia/deepstream "/bin/bash" Up 10 hours deepstream\_container

✅ Container ID: **efa340e929d6**

### ****Commit the Container****

docker commit efa340e929d6 my\_deepstream\_with\_conda

✅ Now, the saved image **my\_deepstream\_with\_conda** contains all installed packages.

## ****10. Restart the Saved Container Later****

docker run --gpus all -it \

-e DISPLAY=$DISPLAY \

-v /tmp/.X11-unix:/tmp/.X11-unix \

-v /home/prakash/deepstream6.4:/working \

my\_deepstream\_with\_conda /bin/bash

## ****🎯 Summary****

✔️ **Set up DeepStream in Docker**  
✔️ **Installed Conda, PyTorch, TensorRT**  
✔️ **Converted YOLOv8 & AlexNet to ONNX & TensorRT**  
✔️ **Saved the Docker container**

**My\_deepstream**

xhost +local:docker

docker run --gpus all -it \

-e DISPLAY=$DISPLAY \

-v /tmp/.X11-unix:/tmp/.X11-unix \

-v /home/prakash/deepstream6.4:/working \

my\_deepstream /bin/bash  
  
# Set Correct Paths

DEPLOY\_CONFIG=/working/michael-mmyolo-playground/deploy\_trt.py

MODEL\_CONFIG=/working/michael-mmyolo-playground/custom\_train.py

MODEL=/working/training-output/best\_coco\_bbox\_mAP\_epoch\_788.pth

DEPLOY\_SCRIPT=/working/mmdeploy/tools/deploy.py

TEST\_IMAGE=/working/test.jpg

**My\_deepstream\_conda**

xhost +local:docker

docker run --gpus all -it \

-e DISPLAY=$DISPLAY \

-v /tmp/.X11-unix:/tmp/.X11-unix \

-v /home/prakash/deepstream6.4:/workspace \

my\_deepstream\_with\_conda /bin/bash  
  
# Set Correct Paths

DEPLOY\_CONFIG=/workspace/michael-mmyolo-playground/deploy\_trt.py

MODEL\_CONFIG=/workspace/michael-mmyolo-playground/custom\_train.py

MODEL=/workspace/training-output/best\_coco\_bbox\_mAP\_epoch\_788.pth

DEPLOY\_SCRIPT=/workspace/mmdeploy/tools/deploy.py

TEST\_IMAGE=/workspace/test.jpg

# Run Deployment

conda activate torch1.9 # only for conda

python3 $DEPLOY\_SCRIPT \

$DEPLOY\_CONFIG \

$MODEL\_CONFIG \

$MODEL \

$TEST\_IMAGE \

--work-dir /workspace/training-output/exports \

--device cuda:0 \

--log-level INFO \

--show \

--dump-info

After conversion

conda deactivate