Lab 2: Object Recognition

NOTE: We will be installing essential software dependencies and packages so make sure you have a stable internet connection for your Jetson Nano before proceeding.

Objective: This lab will cover the setup and use of Docker containers, basic object recognition and training modules for the Jetson Nano Developer Kit.

Preface:
Object recognition is the ability of an AI system to recognize and identify objects in images or video data. This is achieved through machine learning algorithms. Some uses for object recognition include:

- enhancing image and video processing
- improved robotics and automation
- advancing autonomous vehicles
- improving security
- facilitating natural language processing

Object recognition enables machines to interact with the world around them and complete more challenging tasks. Consider this idea and use this lab to help facilitate possible concepts for the upcoming project proposal.

Materials Required:
- Jetson Nano Development Kit (basic setup complete)
- IMX 219 or USB Webcam
- ethernet cable / WIFI module
- keyboard and mouse
- display with cables / secondary computer

Assignment Submission Instructions

You will need to turn in the following for full credit (10 points + 6 bonus points) on today's lab:

- Part 1 deliverables (3 pts + 2 bonus)
- screenshot of detectnet with 2 or more objects (2 pts)
- all questions answered (4 pts)
- (Bonus) Part IV (4 pts)
- all the items above turned in as a .zip file (1 pts)

All the items above **must be** turned in as a .zip file, please **do not** use Github for submission

Part I: Getting started with Docker container

If you are not familiar with Docker, please complete the following tutorial.
CS131-Getting-Started-Docker.docx
Dockerfile
mycode.py

(3 points + 2 bonus) Q1.1
Problem Statement:
Create a Docker container to explore and understand system process monitoring and management using `htop` and `top`. The task involves running a background process to observe its impact on system resources in real-time and capturing system process information to a mounted volume

Requirements:
1. Dockerfile Creation and Image Building:
  - Create a Dockerfile using an Ubuntu or Alpine base image.
  - Install *htop*.
    `apt install htop`
  - Build the Docker image and tag it appropriately.

2. Run Docker Container with Interactive Shell:
  - Run the Docker container interactively.
  - Start *htop* inside the container to view real-time process information.

3. Run a Background Program:
  - While *htop* is running, open a new terminal.
  - Run a script or program in the background that noticeably affects CPU or memory usage, such as a continuous loop.
    Example command: `bash -c “while true; do :; done” &`
    - ‘bash -c’ starts a new bash shell and runs the command provided as a string.
    - "while true; do :; done" is an infinite loop with no operation inside it.
    - & runs the command in the background.

  - Observe the impact of this background process in *htop*.
  - Terminate the process
    - ps aux | grep ‘while’
    - kill -9 <PID>

Deliverables:
Dockerfile, commands used to build and run your dockerfile in a README.txt file and screenshots of htop output - before and while running your background script

(Bonus Requirements)
You'll notice that any modifications or data created in interactive mode are lost once the container is stopped. To persist data, we can use docker volumes. In this part of the lab, you are tasked with implementing a Docker volume. Your goal is to effectively mount this volume inside your container, ensuring that it serves as a storage location for capturing and saving process information generated during your session. This approach safeguards your data against container restarts or shutdowns but also provides an essential understanding of managing persistent data in container environments.

4. Setup Volume Mounting:
   - Create a directory on your host machine.
   - Run the Docker container again, this time mounting the created directory to a specific path inside the container (e.g., `/data`).

5. Capture and Save Process Information:
   - Inside the container, use the *top* command in batch mode to capture a snapshot of system processes and redirect the output to a file (create busy_process.txt) in the mounted volume
   - Verify that the file `busy_process.txt` is created in the mounted volume and contains the expected information.

Bonus Deliverables:
Updated Dockerfile that includes both the above parts, commands used to build and run your dockerfile in a README.txt file and your updated busy_process.txt file with the process information.

Part II - Setup Camera Module and DetectNet

Step 1: verify camera module

- attach camera module to relevant port
- **Reboot** the board after you insert the camera on board
- open *Cheese Webcam Booth* to test (Only for USB Webcam), we are using csi camera, so ignore this
- Use command `ls /dev/video*` to detect if the camera is connected, if that returns /dev/video0 or /dev/video1, it means the camera is connected
- Make sure you have upgraded everything
    - sudo apt update
    - sudo apt upgrade

Step 1.5: for CSI cameras (any devices connected to CAM0 or CAM1 ports)

- run commands (lines with # are comments):

```
git clone https://github.com/JetsonHacksNano/CSI-Camera.git
```

```
# Simple Test
#  Ctrl^C to exit
# sensor_id selects the camera: 0 or 1 on Jetson Nano B01
cd CSI-Camera
gst-launch-1.0 nvarguscamerasrc sensor_id=0 ! nvoverlaysink
```

- for verification and troubleshooting issues: https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md

Step 2: download and run container image

- run the following commands:

```
git clone --recursive http://github.com/dusty-nv/jetson-inference/
cd jetson-inference/
git submodule update --init --recursive
sudo docker/run.sh
```

- contents of the container can be verified through jetson-inference/data directory

Step 2.5: docker/run.sh error troubleshooting

NOTE: This issue is particularly common with the Jetson Nano 2GB kit

- if you are unable to run docker/run.sh successfully you will need to build the project from source
- follow these exact steps: https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md

Step 3: launch video-viewer utility

- run the following commands (must be in root):

```
sudo su
# displays the system mount for webcam
ls /dev/video*
cd build/aarch64/bin
# replace /dev/video0 with csi://0 for IMX 219 camera
# ./video-viewer /dev/video0
./video-viewer csi://0
```

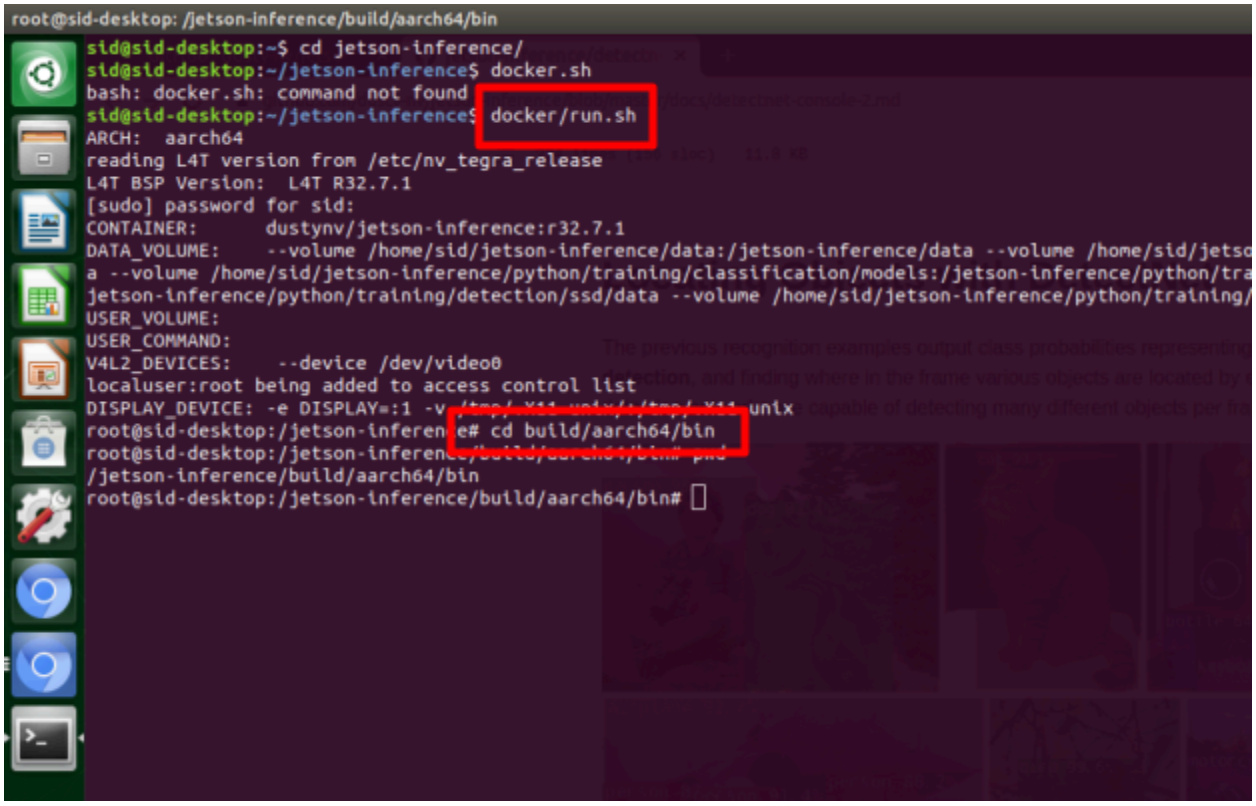
Step 4: basic object detection with detectnet

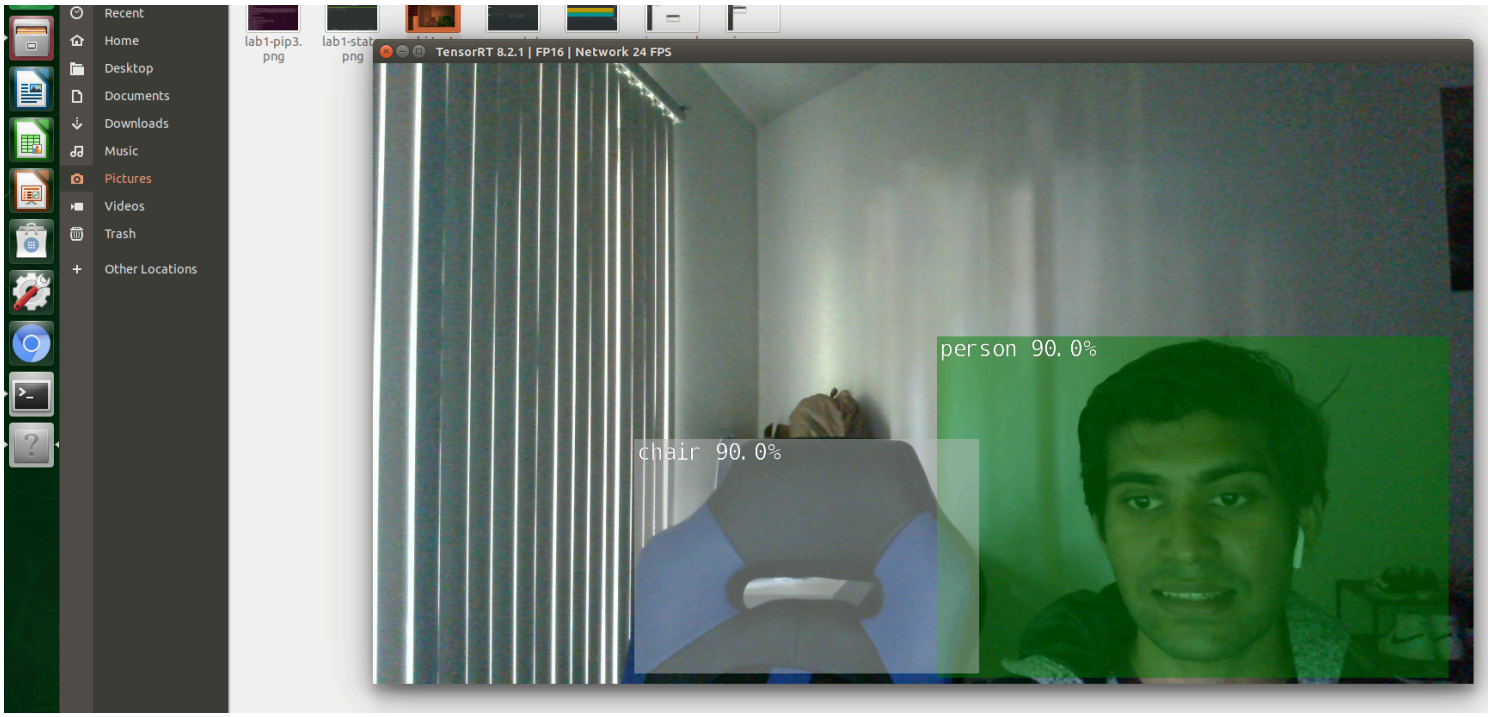
NOTE: This is the default model *SSD-MobileNet-v2* for Jetson Nano

- make sure you are running docker/run.sh
- make sure you are in the following directory: ./jetson-inference/build/aarch64/bin
- run the following command (must be in root):

```
# replace /dev/video0 with csi://0 for IMX 219 camera
# detectnet /dev/video0
detectnet csi://0
```

- test detectnet tool with basic objects





DetectNet is able to recognize a person and chair with 90.0% certainty.

Part III - Testing Object Recognition Database

Step 1: identify sample images in container

- open *File*
- head to the jetson-inference/data/images director

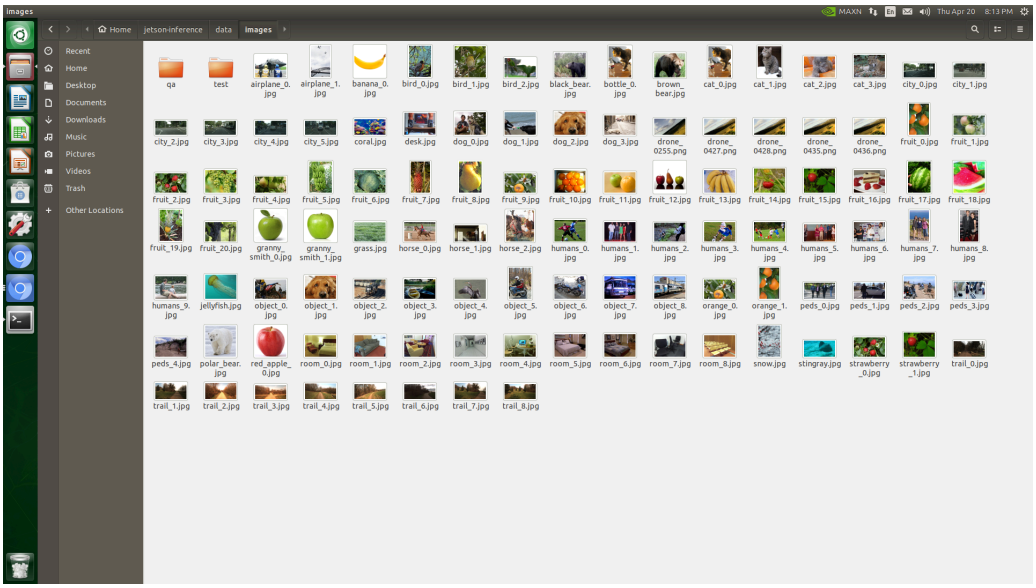
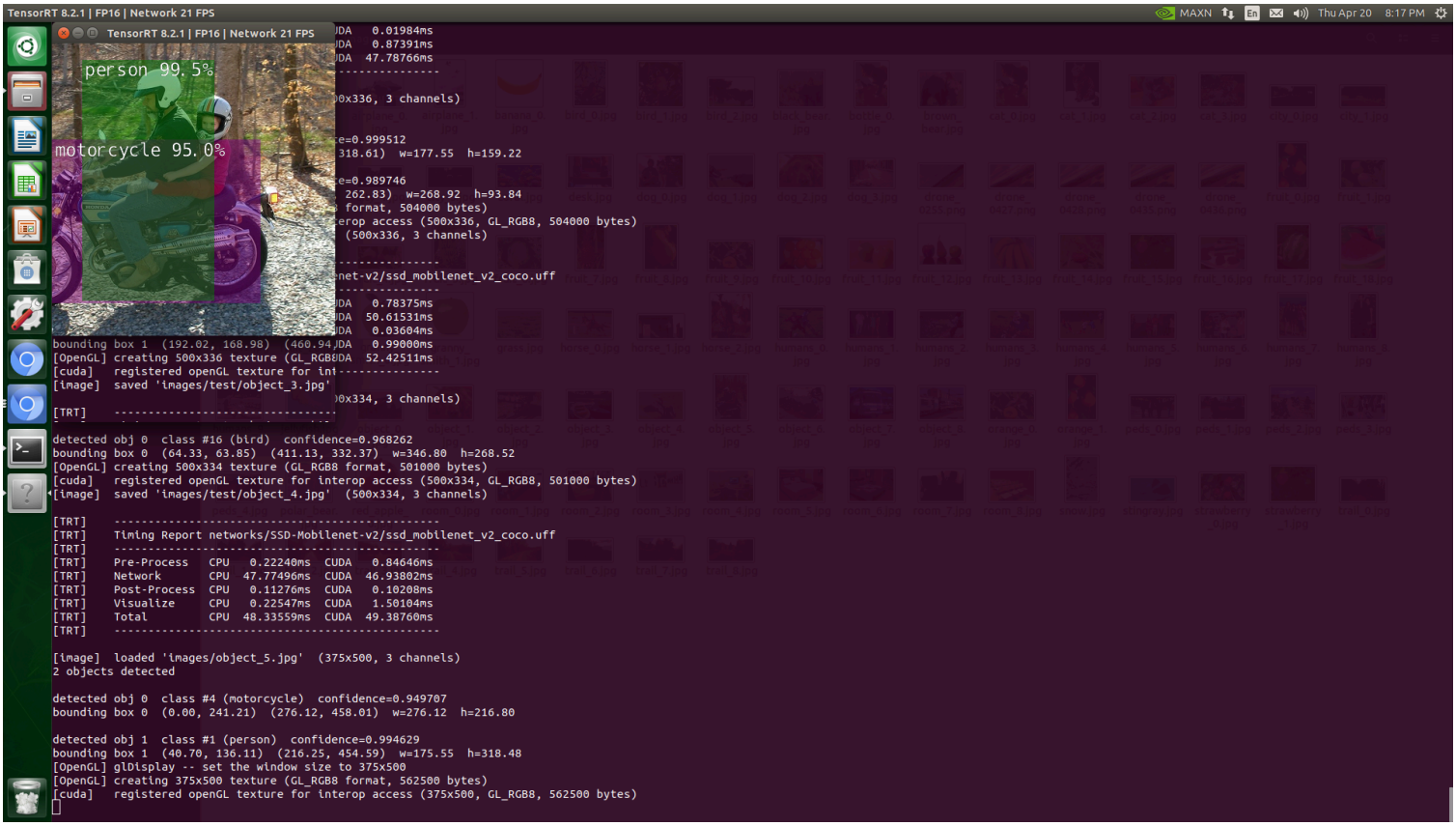


Image repository with default samples provided by jetson-inference container

Step 2: Test detectnet with samples

- run docker/run.sh if needed
- go to relevant directory
- pick class of objects you would like to test (e.g. room_*.jpg)
- run the following command:

```
docker\run.sh
#Go to images directory
cd data/images
# replace object_*.jpg with your chosen class of objects
# You can use ls command to check what images under images folder
detectnet images/object_*.jpg images/test/object_%i.jpg
# make sure to leave the _%i.jpg to receive test data from detectnet
```



detectnet provides bounding boxes and produces image data using console output

Test with a few different image classes, then answer the questions below:

(1 pts)Q2.1: Exactly how long did it take for detectnet to recognize your images? Explain why?

(1 pts)Q2.2: What is the highest and lowest percent confidence for your selected object class?

(1 pts)Q2.3: Out of all the images you tested, which took the longest?

(1 pts)Q2.4: Which component has a greater latency for image processing: CUDA or CPU? Please show the latency and explain why.

(2 pts)Q2.5: Add screenshot of detectnet with 2 or more objects

(Bonus 4 pts) Part IV - Build and Run a different object detection/image recognition model on Jetson Nano using Docker container

Deliverables: Please submit a zip file containing the output files from your object detection/image recognition model, all relevant source code, and your Dockerfile. Additionally, include a README.txt file in the zip, detailing all commands, and steps used in your project, from setup to execution. This README should provide clear instructions for us to replicate your work, including any commands for building and running the Docker container, and descriptions of any other artifacts included in your submission.

Resources

- Jetson Nano + Raspberry Pi Camera (IMX219)
- Object Detection GitHub
- Jetson AI Fundamentals

- Open Images Dataset V7