12110916 Haoyu Li/
12110644 Sicheng Zhou/
12110649 Chenyang Mao/
12112328 Zubin Zheng /

CS324 · Deep Learing

Project Report

December 27, 2024

1 Introduction

Edge computing has emerged as a crucial paradigm in the artificial intelligence landscape, enabling real-time processing of AI workloads directly on edge devices. This approach significantly reduces latency and bandwidth requirements compared to cloud-based solutions, making it particularly valuable for applications requiring immediate response times. Our project focuses on implementing and demonstrating real-time object detection capabilities on the NVIDIA Jetson Nano platform, a powerful edge computing device designed specifically for AI applications.

The Jetson Nano represents an ideal platform for deploying AI at the edge. It offers a balance between computational power and energy efficiency, making it suitable for real-time AI applications where response times need to be maintained within milliseconds to seconds. In this project, we specifically focus on object detection, a fundamental computer vision task that involves identifying and localizing objects within images or video streams.

2 Method

Our implementation leverages the jetson-inference repository, a comprehensive collection of DNN vision libraries and examples optimized for NVIDIA Jetson platforms. The project implementation followed these key steps.

2.1 Camera Installation

First, unplug the pin of the Nano's CSI interface. Then plug the camera cable, metal facing the heat sink, into the camera port on the Jetson NVIDIA development kit. Close the latch. Connection complete.

We used the code from CSDN to test the camera.

```
import cv2

def gstreamer_pipeline(
    capture_width=1280,
    capture_height=720,
    display_width=1280,
    display_height=720,
    framerate=60,
    flip_method=0,
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, "
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
```

```
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )

def show_camera():
    cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)

    while cap.isOpened():
        flag, img = cap.read()
        cv2.imshow("CSI Camera", img)
        kk = cv2.waitKey(1)

        # do other things

        if kk == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    show_camera()
```

2.2 Building the Project from Source

First, we prepared the development environment for Jetson Nano.

```
# Updated the package list and installed essential dependencies
sudo apt-get update
sudo apt-get install git cmake libpython3-dev python3-numpy

# Installed Python development packages to enable Python bindings for the C++ code
# Ensured all necessary CUDA and TensorRT dependencies were in place through JetPack
```

Then, we built the project from the source following these steps.

```
# Cloned the jetson-inference repository with its submodules:
git clone --recursive https://github.com/dusty-nv/jetson-inference

# Created and configured the build environment using CMake:
cd jetson-inference
mkdir build
cd build
cmake ../

# Compiled the project and installed the libraries:
make -j$(nproc)
```

```
sudo make install
sudo ldconfig
```

The compilation process built several key components:

- Core libraries in `/usr/local/lib`
- Header files in `/usr/local/include`
- Python bindings for jetson.inference and jetson.utils
- Binary executables in `/usr/local/bin`
- Pre-trained neural network models

2.3 DetectNet

DetectNet is a deep learning-based object detection framework specifically designed for efficient and real-time detection of objects within images or video streams. It is a part of NVIDIA's Deep Learning framework ecosystem and works seamlessly with NVIDIA GPUs, including the Jetson platform. DetectNet leverages convolutional neural networks (CNNs) to perform both object localization and classification in one unified architecture, making it highly effective for applications like autonomous vehicles, robotics, and video surveillance.

DetectNet is based on a single-shot detection approach, meaning it simultaneously predicts bounding boxes and classifies objects in one pass through the network. This approach makes it computationally efficient compared to traditional two-stage detectors like Faster R-CNN, which first generate region proposals and then classify those regions.

The decision to use DetectNet for object detection on Jetson Nano is influenced by several factors:

1. **Efficiency on Edge Devices** Jetson Nano is a small, low-power device designed for edge AI applications. DetectNet's lightweight and efficient architecture is well-suited for this platform, as it provides real-time performance without requiring extensive computational resources.
2. **Optimized for NVIDIA GPUs** DetectNet is tailored to leverage NVIDIA GPUs, including the embedded GPU on Jetson Nano. This ensures maximum performance by utilizing CUDA cores for parallel processing and accelerating inference tasks.
3. **Ease of Training and Deployment** DetectNet can be retrained using NVIDIA's Transfer Learning Toolkit, allowing developers to fine-tune the model on specific datasets. It also integrates seamlessly with NVIDIA's TensorRT for optimized inference performance on Jetson Nano.
4. **Real-Time Object Detection** Many applications of Jetson Nano, such as robotics, smart cameras, and IoT devices, require real-time object detection. DetectNet's single-shot detection approach ensures fast and accurate predictions, enabling real-time response.
5. **Flexibility for Custom Applications** DetectNet supports various object detection tasks, such as pedestrian detection, vehicle detection, and more, making it versatile for different use cases.
6. **Community and Ecosystem Support** DetectNet is supported by NVIDIA's rich ecosystem of tools, libraries, and community forums. This ensures access to resources, tutorials, and troubleshooting support, which is crucial for developers working on Jetson Nano.

2.4 Training

The model we used is based on MobileNetv2-SSD and trained on The MS COCO (Microsoft Common Objects in Context) dataset, which is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images of 91 categories.

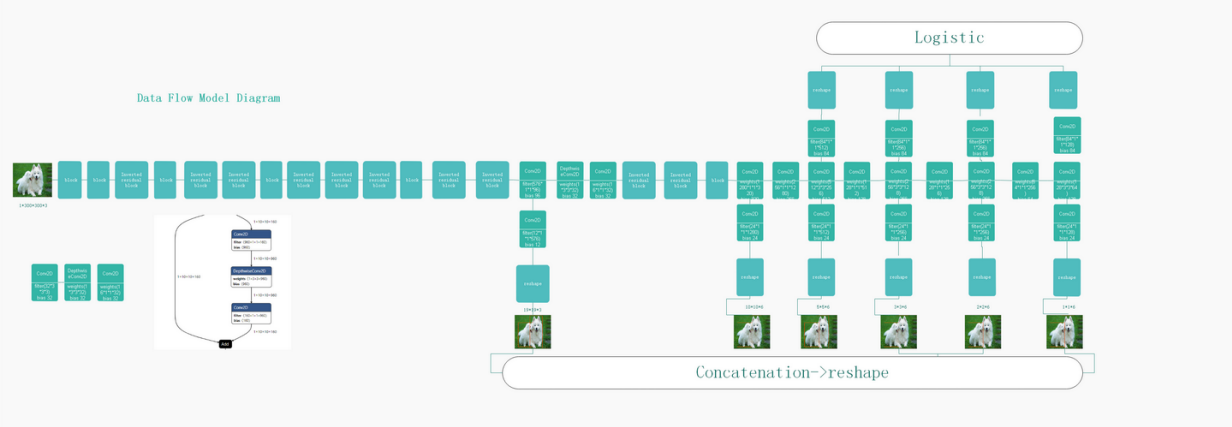

Figure 1: MobileNetv2-SSD consisting of 5 bottleneck residual blocks similar to MobileNetv2 and 9 convolution layers. Extract featuremaps of size 19X19,10x10,5x5,3x3,2x2,1x1 to do object detection, similar to the idea of SSD.

3 Result

3.1 Camera Test Results

The test code and results of our camera are shown in Figure 2.

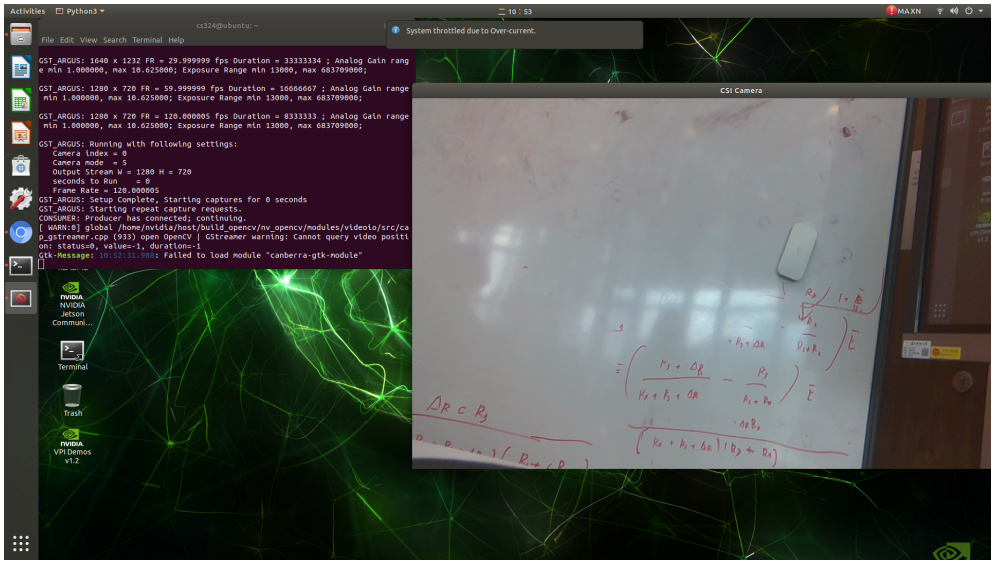

Figure 2: The test code and results of our camera.

3.2 Code Execution Screenshots

The code execution illustration is shown in Figure 3.

3.3 Recognized Objects

We conducted experiments to evaluate the performance of our object detection algorithm across various scenes containing everyday objects. The results, as illustrated in Figure 4, demonstrate the algorithm's capability to identify and classify multiple objects accurately. The detected objects and their corresponding confidence scores are summarized as follows:

- **Keyboard:** Detected with a confidence score of 66.7%.

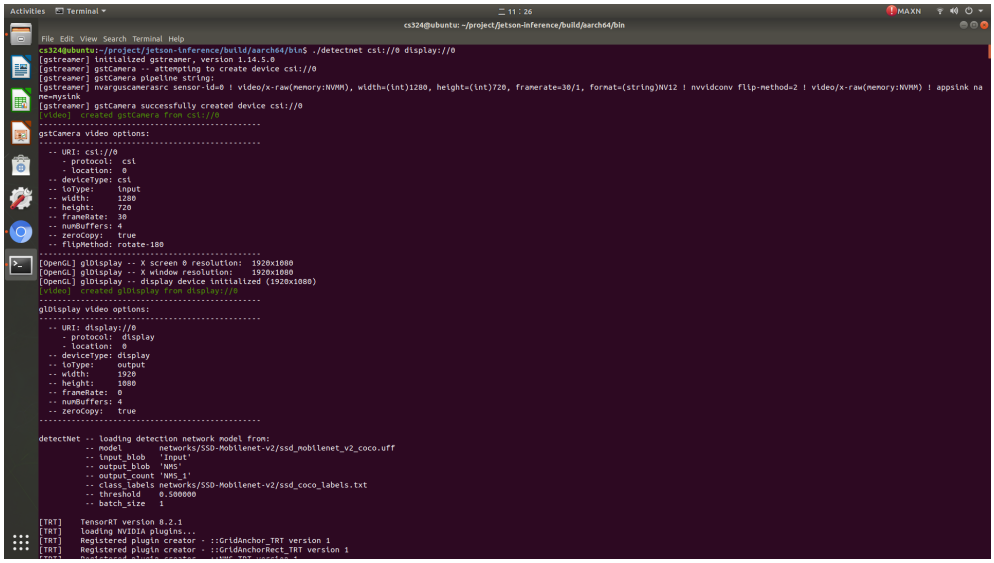

Figure 3: The code execution screenshots.

- **Mouse:** Detected with a high confidence score of 98.9%.
- **Cell Phone:** Achieved a confidence score of 90.6%.
- **Remote Control:** Identified with a confidence score of 66.5%.
- **Chair:** Detected with a confidence score of 87.3%.
- **Backpack and Handbag:** The backpack and handbag were detected with confidence scores of 51.5% and 94.1%, respectively.
- **Television (TV):** Detected with a confidence score of 81.2%.

These results highlight the robustness of our algorithm in recognizing objects under various conditions, including diverse lighting, angles, and partial occlusions. The overall detection performance is satisfactory, particularly for high-priority objects such as electronic devices.

Figure 4: Object detection results from our algorithm applied to various scenes. Detected objects include a keyboard, mouse, cell phone, remote control, chair, backpack, handbag, and television, each annotated with their respective confidence scores. The results demonstrate the algorithm's capability to recognize and classify objects in diverse environments.

3.4 Unrecognized Objects

Sometimes, the object detection algorithm produces inaccurate results. For example, it misclassified a ceiling light in a discussion room as an "airplane." This error likely stems from the training dataset, where most objects located high above or far in the sky are labeled as "airplane." Such biases in the dataset lead to misclassifications.

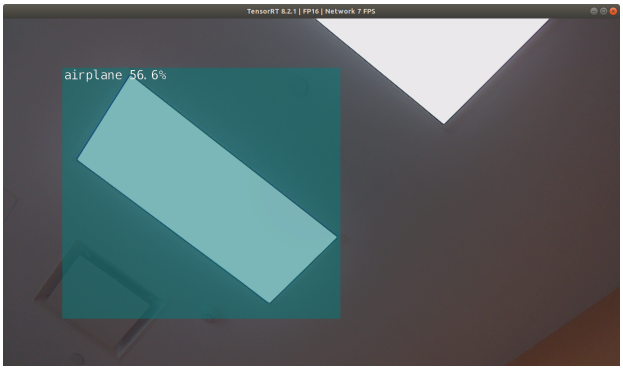

Figure 5: Examples of Poor Recognition

4 Analysis

When deploying the model for inference, we noticed an interesting phenomenon: the first time we ran the model MobileNetv2-SSD, it seemed that TensorRT took a few minutes (i.e., 4 ∼ 6 minutes) to optimize the network. Fortunately, this optimized network file was then cached to disk, so later runs using the model loaded faster (i.e., 20 ∼ 30 seconds). It is worth delving deeper into the reasons TensorRT performed this optimization when deploying the model for the first time.

NVIDIA® TensorRT™is an ecosystem of APIs for high-performance deep learning inference. TensorRT includes an inference runtime and model optimizations that deliver **low latency** and **high throughput** for production applications. The TensorRT ecosystem includes TensorRT, TensorRT-LLM, TensorRT Model Optimizer, and TensorRT Cloud. The following are some benefits of TensorRT:

- **Speed Up Inference by 36X** NVIDIA TensorRT-based applications perform up to 36X faster than CPU-only platforms during inference. TensorRT optimizes neural network models trained on all major frameworks, calibrates them for lower precision with high accuracy, and deploys them to hyperscale data centers, workstations, laptops, and edge devices.
- **Optimize Inference Performance** TensorRT, built on the CUDA parallel programming model, optimizes inference using techniques such as quantization, layer and tensor fusion, and kernel tuning on all types of NVIDIA GPUs, from edge devices to PCs to data centers.

NVIDIA TensorRT Model Optimizer can accelerate AI inference performance via neural network optimization. Specifically, NVIDIA TensorRT Model Optimizer is a unified library of state-of-the-art model optimization techniques, including quantization and sparsity:

- **Post-training quantization (PTQ)** is one of the most popular model compression methods to reduce memory footprint and accelerate inference. Model Optimizer provides advanced calibration algorithms including INT8 SmoothQuant and INT4 AWQ (Activation-aware Weight Quantization).
- **Sparsity** further reduces the size of models by selectively encouraging zero values in model parameters that can then be discarded from storage or computations.

It compresses deep learning models for downstream deployment frameworks like TensorRT to efficiently optimize inference on NVIDIA devices.

In summary, it can typically take around 4 ~ 6 minutes for TensorRT to optimize the network on Jetson Nano (although that time can vary as it is profiling different kernels on the GPU to find the configuration of DNN layers that runs fastest). This should only occur the first time we load a particular model; after it's optimized, the optimized model will be saved to disk and then loaded straight away on further runs.