Pytorch to TensorRT tutorial

Đặng Anh Quân

# Pipeline:

A diagram of a software development

Description automatically generated

* Step 1: Basic set up for jetson nano:
  + Cập nhật .bashrc file để chứa CUDA nếu chưa có sẵn sử dụng install\_basics.sh file. (.bashrc file – Bash Run Commands là file sẽ được chạy mỗi lần shell mới được bắt đầu, sử dụng để cài đặt biến môi trường – environment variables, …)
* $ mkdir -p ${HOME}/project
* $ cd ${HOME}/project
* $ git clone <https://github.com/jkjung-avt/jetson_nano.git>
* $ cd jetson\_nano/
* $ ./install\_basics.sh
  + Tạo swap hay RAM ảo (sử dụng để hỗ trợ lưu trữ dữ liệu vào bộ nhớ đĩa cứng khi RAM hết, có ích khi hệ thống tràn bộ nhớ vật lý RAM)
    - Kiểm tra Swap: swapon -s
    - Kiểm tra dung lượng đĩa trống: df -h
    - Tạo tệp tin Swap: $ sudo fallocate -l 8G /mnt/8GB.swap (8GB)
    - Tạo phân vùng Swap: $ sudo mkswap /mnt/8GB.swap
    - Kích hoạt: $ sudo swapon /mnt/8GB.swap
    - Kiểm tra lại trạng thái swap: swapon -s
    - Cài đặt để tự động kích hoạt swap mỗi khi boot hệ thống: thêm /mnt/8GB.swap none swap sw 0 0 vào cuối file /etc/fstab
    - Khởi động lại.
  + Cài đặt thư viện TensorRT yêu cầu TensorRT 6.x+:
    - Kiểm tra version của TensorRT bằng:
      * ls /usr/lib/aarch64-linux-gnu/libnvinfer.so\*
    - Kiểm tra hoạt động:
      * $ wget https://gist.githubusercontent.com/jkjung-avt/86b60a7723b97da19f7bfa3cb7d2690e/raw/9561b135c1b40f82afe46afcc1f326a2e913e08c/tegra-cam.py
      * $ python3 tegra-cam.py --usb --vid 0 --width 1280 --height 720
  + Cài đặt openCV:
    - Sử dụng install\_opencv-3.4.6.sh
  + Cài đặt protobuf-3.8.0:
    - Sử dụng install\_protobuf-3.8.0.sh
* Step 2: Install Pycuda using install\_pycuda Bash script:
  + Comment thông tin chung:
    - #!/bin/bash: thể hiện file được viết trong dạng Bash
    - # Reference for installing 'pycuda': https://wiki.tiker.net/PyCuda/Installation/Linux/Ubuntu : thể hiện thông tin mục đích của scripts dùng để cài đặt PyCuda
  + Setup để file thoát:
    - set -e: khiến file thoát ngay lập tức khi 1 command chạy và trả lại non-zero, thể hiện rằng bị lỗi
  + Xác định xem NVCC (NVIDIA CUDA Compiler) có hay chưa:
    - if ! which nvcc > /dev/null; then
    - echo "ERROR: nvcc not found"
    - exit
    - Fi
  + Lưu trữ các thông số máy bằng các biến:
    - arch=$(uname -m): lưu trữ kiến trúc máy (ví dụ: x86\_64)
    - folder=${HOME}/src: set đường dẫn ${HOME}/src trong biến folder
    - mkdir -p $folder: tạo thư mục xác định bởi folder nếu chưa hiện hữu
  + Cài đặt các thư viện cần thiết:
    - echo "\*\* Install requirements"
    - sudo apt-get install -y build-essential python3-dev python3-pip
    - sudo apt-get install -y libboost-python-dev libboost-thread-dev
    - sudo pip3 install setuptools
  + Trích xuất tên của Boost Python library từ đường dẫn hệ thống:
    - boost\_pylib=$(basename /usr/lib/$(arch)-linux-gnu/libboost\_python\*.so)
    - boost\_pylibname=${boost\_pylib%.so}
    - boost\_pyname=${boost\_pylibname/lib/}
  + Download PyCuda version 2019.12 nếu chưa tồn tại:
    - echo "\*\* Download pycuda-2019.1.2 sources"
    - pushd $folder
    - if [ ! -f pycuda-2019.1.2.tar.gz ]; then
    - wget https://files.pythonhosted.org/packages/5e/3f/5658c38579b41866ba21ee1b5020b8225cec86fe717e4b1c5c972de0a33c/pycuda-2019.1.2.tar.gz
    - Fi
  + Build và install pycuda-2019.1.2
    - echo "\*\* Build and install pycuda-2019.1.2"
    - CPU\_CORES=$(nproc) :gán giá trị số cpu cores
    - echo "\*\* cpu cores available: " $CPU\_CORES
    - tar xzvf pycuda-2019.1.2.tar.gz : extract
    - cd pycuda-2019.1.2
  + Config, compile và build:
    - python3 ./configure.py --python-exe=/usr/bin/python3 --cuda-root=/usr/local/cuda --cudadrv-lib-dir=/usr/lib/${arch}-linux-gnu --boost-inc-dir=/usr/include --boost-lib-dir=/usr/lib/${arch}-linux-gnu --boost-python-libname=${boost\_pyname} --boost-thread-libname=boost\_thread --no-use-shipped-boost
    - # if you get --no-use-shipped-boost run follow remove command
    - # rm siteconf.py
    - make -j$CPU\_CORES
    - python3 setup.py build
    - sudo python3 setup.py install
  + Quay lại thư mục ban đầu:
    - Popd
  + Check version:
    - python3 -c "import pycuda; print('pycuda version:', pycuda.VERSION)"
* Step 2: Cài đặt onnx version 1.9.0 tùy thuộc vào protobuf:
  + $ sudo pip3 install onnx==1.9.0

A screenshot of a screen

Description automatically generated

* Step 3: Build “yolo\_layer” plugin
  + $ cd ${HOME}/project/tensorrt\_demos/plugins
  + $ make
* Step 4: tải các mô hình pre-trained yolov3/yolov4 COCO models và convert model sang ONNX rồi sang TensorRT engine.
  + $ cd ${HOME}/project/tensorrt\_demos/yolo
  + $ ./download\_yolo.sh
  + $ python3 yolo\_to\_onnx.py -m “yolo\_models”
  + $ python3 onnx\_to\_tensorrt.py -m “yolo\_models”
* Step 5: Test TensorRT với dog.jpg
  + $ cd ${HOME}/project/tensorrt\_demos
  + $ wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/dog.jpg -O ${HOME}/Pictures/dog.jpg
  + $ python3 trt\_yolo.py --image ${HOME}/Pictures/dog.jpg\ -m yolov4-416
* Step 6 (optional): có thể thử với các định dạng đầu vào khác nhau
  + --image test\_image.jpg: an image file, e.g. jpg or png.
  + --video test\_video.mp4: a video file, e.g. mp4 or ts. An optional --video\_looping flag could be enabled if needed.
  + --usb 0: USB webcam (/dev/video0).
  + --rtsp rtsp://admin:123456@192.168.1.1/live.sdp: RTSP source, e.g. an IP cam. An optional --rtsp\_latency argument could be used to adjust the latency setting in this case.
  + --onboard 0: Jetson onboard camera.
* Step 7: đánh giá mAP sử dụng eval\_yolo.py:
  + Ví dụ: $ python3 eval\_yolo.py -m yolov3-tiny-288

# Train custom YOLOv4 model:

A diagram of a computer program

Description automatically generated

* Các bước để train custom data on yolov4 model (link tham khảo: <https://miai.vn/2020/05/25/yolo-series-train-yolo-v4-train-tren-colab-chi-tiet-va-day-du-a-z/>)
  + Chuẩn bị dữ liệu:
    - Thu thập dữ liệu
    - Sử dụng labellmg để tự gán nhãn (<https://thigiacmaytinh.com/su-dung-tool-labelimg-de-danh-nhan-vat-the-trong-hinh/>)
    - Một số chú ý:
      * Cả ảnh và nhãn đều để chung 1 thư mục
      * Xóa trắng file predifined\_classes.txt trước khi train
  + Chuẩn bị file config
    - Custom config: <https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-custom.cfg>
      * Mấy class cần train
      * Sửa max\_batches = max(<số class>\*2000,6000): số batches tối đa mà quá trình training sẽ chạy
      * Sửa steps = 80% hoặc 90% của max\_batches: xác định ở batches nào thì learning rate nên giảm (giảm dựa theo scales)
      * Sửa toàn bộ các dòng classes = <số class>
      * Sửa toàn bộ filters = <(số class +5)\*3
      * Nếu bị out memory, chuyển subdivisions (chia nhỏ batch thành sub-batches) thành 32 hoặc 64 cách khác là giảm size ảnh
  + Tải backbone – mã nguồn của yolov4:
  + Tạo file YOLO.NAMES chưa tên các class và lưu vào folder darknet:
    - !echo "name\_of\_classes-1" > yolo.names
    - !echo "name\_of\_classes\_2" >> yolo.names
  + Tách thành 2 tập train và validation tỉ lệ 20/80:
  + Tạo file YOLO.DATA chứa tham số train
  + Biên dịch mã nguồn của DARKNET
  + Tải Pretrain weights (optional)
  + Train
  + Theo dõi tham số của quá trình train

# Convert custom model into ONNX then TensorRT engine:

A diagram of a computer program

Description automatically generated with medium confidenceA diagram of a engine

Description automatically generated

* Cần 2 file:
  + File config
  + File weights
* Convert model to ONNX, what does it do step by step:
  + Kiểm tra version của Python (sẽ raises error nếu nhỏ hơn 3)
  + Phân tích dòng lệnh qua hàm parse\_args() (output là cfg\_file\_path, weights\_file\_path và output\_file\_path với định dạng file lần lượt là .cfg; .weights; .onnx
  + Phân tích file cấu hình của backbone darknet sử dụng DarknetParser() (output là layer configuration lưu trong layer\_configs)
  + Xác định số vật thể và anchor sử dụng hàm get\_category\_num (output là số loại lưu trong category\_num)
  + Định nghĩa hình dạng của output tensor:
    - Lấy tên của output convolutional layers trong model sử dụng hàm get\_ouput\_convs(layer\_configs) và lưu vào output\_tensor\_names
    - Tính tổng số giá trị cần dự đoán bởi mỗi anchor và lưu vào biến c
      * c = (category\_num + 5) \* get\_anchor\_num(cfg\_file\_path)
    - Xác định cao và rộng của đầu vào sử dụng hàm get\_h\_and\_w(layer\_configs)
    - Tính toán hình dạng của output tensor dựa vào số của ouput\_tensor\_names. Mỗi hình dạng của tensor là 1 list gồm 3 thông số: số kênh, cao và rộng của ma trận đặc trưng được điều chỉnh dựa trên index của layer
    - Nếu YOLO là kiến trúc PAN (Path Aggregation Network) thì thứ tự của dạng output tensor được ngược lại
    - Cuối cùng output\_tensor\_dims là biến kiểu ordered dictionary được tạo mapping giữa output\_tensor\_names với output\_tensor\_shapes.
  + ONNX Graph Building:
    - Tạo builder là 1 object của GraphBuilderONNX với các tham số như YOLO model name, chiều của output tensor, và maxium batch size
    - Gọi hàm build\_onnx\_graph để xây dựng onnx graph sử dụng darknet layer config và weights file
  + Checking ONNX model
  + Saving ONNX model: Lưu ONNX model vòa ouput\_file\_path
* Convert ONNX to TensorRT engine, what does it do step by step:
  + Phân tích dòng lệnh sử dụng argparse, với một số cờ:
    - -v or --verbose: cho phép xuất ra chi tiết để gỡ lỗi .
    - -c or --category\_num: Số loại.
    - -m or --model: model.
    - --int8: xây dựng INT8 TensorRT engine hay ko.
    - --dla\_core: Xác định ID của nhân DLA (Deep Learning Accelerator) cho suy luận (0 ~ N-1)
  + Xây dựng TensorRT engine sử dụng hàm build\_engine với các tham số:
    - args.model: model.
    - args.int8: xây dựng INT8 TensorRT engine hay ko.
    - args.dla\_core:
    - args.verbose:.
  + Kiểm tra quá trình build engine có thành công hay không
  + Lưu TensorRT engine vào engine\_path

# Runing:

* Sử dụng file trt\_yolo.py để chạy:
  + Phân tích câu lệnh sử dụng hàm parse\_args:
    - desc: mô tả đoạn code sẽ sử dụng làm gì
    - add\_camera\_args(parser): thêm các cờ liên quan đến câu lệnh
    - -c or --category\_num: Số loại.
    - -t or --conf\_thresh: Định nghĩa confidence threshold, defualt là 0.3
    - -m or --model: model
    - -l or --letter\_box: có áp dụng suy luận với letterboxed hay không, defualt là False
  + Kiểm tra số category
  + Kiểm tra sự hiện diện của model file
  + Khởi tạo camera
  + Khởi tạo object:
    - Khởi tạo biến dictionary cls\_dict mapping giữa chỉ số class và tên của class.
    - Khởi tạo đối tượng của lớp BBoxVisualization và TrtYOLO dùng cho visualization và inference
    - Mở cửa sổ visualization
    - Lặp và detect: gọi hàm loop\_and\_detect dùng để liên tục captures frames từ camera và thực thi real-time object detection sử dụng TensorRT YOLO model đồng thời trực quan hóa kết quả
    - Releasing tài nguyên khi không dùng đến.

# Evaluating: