HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH
KHOA VIỄN THÔNG II

ĐỀ TÀI
THỰC TẬP TỐT NGHIỆP

CHUYÊN NGÀNH: ĐIỆN TỬ - TRUYỀN THÔNG
HỆ: ĐẠI HỌC CHÍNH QUY
NIÊN KHÓA: 2015-2020

Đề tài:

Hệ thống nhận dạng vật thể dựa trên mô hình YOLO sử dụng board mạch NVIDIA Jetson TX2

Sinh viên thực hiện:	NGUYỄN HOÀI NAM
MSSV:	N15DCVT036
Lớp:	Đ15CQVT01
Giáo viên hướng dẫn:	PGS.TS. VÕ NGUYỄN QUỐC BẢO
	TS. NGUYỄN ĐỨC PHÚC

Tháng 07 năm 2019
TP.HCM - 2019

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH
KHOA VIỄN THÔNG II

ĐỀ TÀI
THỰC TẬP TỐT NGHIỆP

CHUYÊN NGÀNH: ĐIỆN TỬ - TRUYỀN THÔNG
HỆ: ĐẠI HỌC CHÍNH QUY
NIÊN KHÓA: 2015-2020

Đề tài:

Hệ thống nhận dạng vật thể dựa trên mô hình YOLO sử dụng board mạch NVIDIA Jetson TX2

Sinh viên thực hiện:	NGUYỄN HOÀI NAM
MSSV:	N14DCVT036
Lớp:	Đ15CQVT01
Giáo viên hướng dẫn:	PGS.TS. VÕ NGUYỄN QUỐC BẢO
	TS. NGUYỄN ĐỨC PHÚC

Tháng 07 năm 2019
TP.HCM - 2019

NHẬN XÉT CỦA GIẢNG VIÊN

ABSTRACT

The NVIDIA Jetson TX2 was known as an ideal embedded GPU system. For the inspection the abilities of this system. Combine with deep-learning algorithms win YOLO recognition models. After a brief of object recognition was presented basically. This project deployed an object recognition applicaiton by using YOLO on Jetson TX2. Processes was tracked that involve in figure of hardware elements. The experimental results show that Jetson TX2 with Max-N has a good performance with acceptable statistics. Furthermore, proposal for enhancing factors forward to better result in reality applications.

ACKNOWLEDGEMENT

This work was done during an internship at Posts and Telecommunications Institute of Technology. I would like to specifically acknowledge Prof. Vo Nguyen Quoc Bao, Dr. Nguyen Duc Phuc for providing precious comments within a project time. Acknowledgements are also due to NVIDIA GPU Grant. Finally, I thank PTIT, and IoT Systems Lab for supporting me during my internship project.

LỜI CẢM ƠN

Học tập là quá trình tìm tòi kiến thức và rèn luyện bản thân không ngừng. Trong suốt quá trình học tập và hoàn thành đề tài thực tập, em đã nhận được sự giúp đỡ từ các thầy cô và mọi người xung quanh. Để bày tỏ lòng biết ơn của mình, em xin chân thành gửi lời cảm ơn đến thầy hướng dẫn **PGS.TS. Võ Nguyễn Quốc Bảo** và thầy **TS. Nguyễn Đức Phúc** đã hướng dẫn tận tình, và có những lời đóng góp ý kiến cho đề tài của em.

Em chân thành gửi lời cảm ơn quý thầy cô khoa Viễn Thông 2 đã giúp đỡ em trong quá trình học tập và rèn luyện bản thân trong suốt những năm qua.

Em chân thành gửi lời cảm ơn đến các bạn sinh viên thuộc IoTsLab đã giúp đỡ trong suốt thời gian sinh hoạt tại phòng lab.

Em chân thành gửi lời cảm ơn đến gia đình và bạn bè đã luôn ở bên cạnh và động viên em trong suốt thời gian qua.

Hồ Chí Minh, ngày 10 tháng 07 năm 2019

Sinh viên thực hiện

Nguyễn Hoài Nam

Mục lục

THUẬT NGỮ VIẾT TẮT

Từ Viết Tắt	**Tiếng Anh**
AI	Artificial Intelligence
WLAN	Wireless Local Area Network
UAV	Unnamed Aerial Vehicle FFT
Fast Fourier Transform	
SAE	Society of Automotive Engineers
FIFO	First In First Out
DL	Deep Learning
RCNN	Region Convolutional Neural Network
SSE	Sum Square Error
L4T	Linux for Tegra
RGB	Red Green Blue

Danh sách hình vẽ

Danh sách bảng

LỜI MỞ ĐẦU

Cách mạng công nghệ 4.0 đã mở ra các xu hướng công nghệ mới như Internet of Things, drone, machine learning và AI. Việc ứng dụng các công nghệ mới vào trong cuộc sống sẽ giúp gia tăng hiệu quả và cải thiện cuộc sống của chúng ta.

Trong cuộc sống ngày nay, trí tuệ nhân tạo được ứng dụng rộng rãi, dần len lõi vào cuộc sống của chúng ta. AI hỗ trợ cuộc sống của con người trong hầu hết các ứng dụng thường ngày. Từ đó, xuất hiện số lượng lớn các mô hình phần mềm cũng như phần cứng góp phần vào sự phát triển của các ứng dụng trí tuệ nhân tạo. Mô hình Yolo là mô hình phổ biến trong nhận dạng vật thể cho các ứng dụng học sâu. Song song theo đó, các hệ thống nhúng hỗ trợ mạnh mẽ trong việc xử lý các tác vụ liên quan đến học máy, tiêu biểu là mạch phát triển NVIDIA Jetson TX2.

Để có một cái nhìn tổng quan và cụ thể nhất, trong đề tài này, em sẽ tập trung nghiên cứu mô hình nhận diện vật thể Yolo, đồng thời tìm hiểu phương thức hoạt động của mô hình này trên mạch phát triển Jetson TX2, kiểm thử mô hình nhận dạng YOLO trên mạch NVIDIA Jetson TX2.

Nội dung đề tài bao gồm 4 chương:

- **Chương 1:** Nhận dạng vật thể
- **Chương 2:** Tổng quan về Jetson TX2
- **Chương 3:** Tổng quan về mô hình nhận diện YOLO
- **Chương 4:** Xây dựng mô hình Yolo trên Jetson TX2

Chương 1

Nhận dạng vật thể

1.1 Bài toán nhận dạng vật thể

Nhận dạng vật thể là một công nghệ trong khoa học máy tính với mục đích nhận dạng đối tượng trong hình ảnh hoặc video[11]. Nhận dạng vật thể là một ngõ ra chính của các thuật toán trong học máy cũng như học sâu. Mắt người khi quan sát một tấm ảnh hay một video, có thể nhanh chóng nhận ra các đối tượng cụ thể hay chi tiết hơn. Với mục tiêu dạy cho máy nhận biết được như con người, đạt được mức hiểu biết về đối tượng có trong ảnh.

Nhận dạng vật thể là một công nghệ chính trong ứng dụng xe không người lái (self-driving car). Ứng dụng rộng rãi trong nhận dạng trong công nghiệp, y tế và robot.

1.2 Bài toán dò tìm vật thể

Dò tìm vật thể và nhận dạng vật thể là hai kỹ thuật tương tự nhau, tuy nhiên độ phức tạp trong xử lý lại khác nhau. Dò tìm vật thể là một tiến trình trong việc tìm kiếm trong trường hợp vật thể có trong ảnh. Đối với học sâu, dò tìm vật thể là một tiến trình con của nhận dạng vật thể, không những nhận dạng đối tượng mà còn là vị trí của đối tượng này. Cho phép nhiều đối tượng được nhận dạng và nhiều vị trí của đối tượng trong cùng một ảnh.

1.3 Tiến trình nhận dạng

Chúng ta nghe nhiều đến Machine Learning và Deep Learning, kỹ thuật nhận dạng vật thể trong hai khái niệm này cũng khác nhau.

- Đối với Machine Learning, bắt đầu với tập các hình ảnh, tiến hành trích xuất những đặc tính nổi bật. Các đặc tính được đưa vào mô hình học máy, tại đây, các đặc tính được phân loại được sử dụng để phân loại và nhận dạng. Có nhiều thuật toán cho việc trích xuất đặc tính đối tượng tùy vào mục đích của các ứng dụng.

- Deep Learning được biết đến với mạng thần kinh (Neural network) được dùng để tự động học kết hợp với các đặc tính để xác định được đối tượng. Có 2 hướng tiếp cận với kỹ thuật nhận dạng trong Deep Learning gồm: Huấn luyện mô hình từ ban đầu và sử dụng mô hình huấn luyện sẵn.

Khi kỹ thuật nhận dạng trong Machine Learning được sử dụng phổ biến cho các ứng dụng đơn giản khi dễ dàng biết được đặc tính các đối tượng. Trong khi đó, đối với Deep Learning thì kỹ thuật này yêu cầu cấu hình đồ họa (GPU) mạnh mẽ trong huấn luyện.

1.4 Các kỹ thuật nhận dạng khác

Template matching và Image segmentation là hai kỹ thuật cơ bản trong nhận diện vật thể. Template matching sử dụng hình ảnh kích thước nhỏ để nhận dạng thông qua khu vực trong ảnh lớn. Image segmentation sử dụng những đặc tính cơ bản về kích thước, màu và hình dạng để nhận dạng.

Các thư viện thường được sử dụng chính trong các mô hình nhận dạng có thể kể đến gồm OpenCV trong các ứng dụng Machine Learning và Tensorflow trong Deep Learning. Các thư viện hỗ trợ công cụ và các chức năng chuyển đổi trong các mô hình đặc trưng cho từng kỹ thuật.

1.5 Phạm vi ứng dụng

Nhận dạng vật thể được ứng dụng trong nhiều lĩnh vực quan trọng:

- Tiến trình quan trọng trong ứng dụng xe tự hành.
- Ứng dụng nhận diện khuôn mặt trong bảo mật, an ninh.
- Công nghệ Robot nhận diện.
- Ứng dụng phân loại đối tượng.
- Nhận dạng ngôn ngữ chữ viết.

Chương 2

Tổng quan về Jetson TX2

2.1 Giới thiệu về Jetson TX2

Jetson TX2 [5] là một board phát triển, đây là máy tính nhúng AI tốc độ cao, tiêu thụ năng lượng thấp. Siêu máy tính này có mức tiêu thụ điện năng 7.5 W tính toán trên AI, cho phép mạch chạy với các hệ thống thần kinh nhân tạo sâu hơn và lớn hơn, giúp cho các thiết bị sẽ thông minh hơn với sự chính xác cao hơn và thời gian đáp ứng nhanh hơn trong các tác vụ phân loại hình ảnh, dẫn đường hay nhận dạng tiếng nói. Jetson TX2 thuộc dòng điện toán nhúng (embedded computing), đây là một nền tảng mở cho phép khả năng phát triển các giải pháp AI hiện đại ở edge – từ các công ty kinh doanh, khởi nghiệp đến việc nghiên cứu, học tập.

Thông số phần cứng của Jetson TX2 gồm:

- Xây dựng trên NVIDIA PASCAL - họ GPU với 256 nhân CUDA [16][24].
- Cấu trúc NVIDIA Denver 64-bit (dual-core) 2GHz và CPU ARM Cortex-A57 (quad-core) 2GHz.
- 8 GB RAM LPDDR4 với băng thông 59.7 Gb/s, 32 GB bộ nhớ Flash.
- Hỗ trợ WLAN chuẩn 802.11ac và Bluetooth 4.1, Ethernet 1 Gb.

Trái tim của Jetson TX2 Developer Kit chính là bộ xử lý di động Tegra X2, là chip di động 256 nhân của NVIDIA, được xây dựng dựa trên kiến trúc NVIDIA Kepler, là GPU hiệu quả về điện năng nhất hiện nay. Toàn bộ 256 nhân có thể lập trình được của Tegra X2 mang lại năng lực tính toán và xử lý đồ hoạ mạnh mẽ cho các thiết bị di động. Bộ xử lý Tegra X2 được xây dựng trên kiến trúc Kepler, cũng là kiến trúc dùng trong các siêu máy tính.

Nền tảng Jetson TX2 hỗ trợ bộ công cụ NVIDIA VisionWorks, là bộ công cụ có tập thuật toán xử lý hình ảnh rất mạnh và phong phú, giúp người dùng lập trình có thể nhanh chóng tạo được ứng dụng. Bộ công cụ này có những tính năng hỗ trợ công nghệ tính toán CUDA cho các lĩnh vực như robot, thực tế ảo, nhiếp ảnh điện toán, tương tác người-máy và các hệ thống xe tự lái.

Jetson TX2 Developer Kit hoàn toàn hỗ trợ cho bộ công cụ phát triển CUDA 7.0, gồm công cụ nhận diện lỗi (debuggers) và công cụ tạo profile để phát triển nhiều ứng dụng xử lý song song quy mô lớn. CUDA 7.0 cũng mang nền tảng ARM vào các thư viện tăng tốc của NVIDIA cho FFT, đại số tuyến tính, ma trận rời rạc hay xử lý hình ảnh và video.

Hình 2.1: Board mạch nhúng NVIDIA Jetson TX2

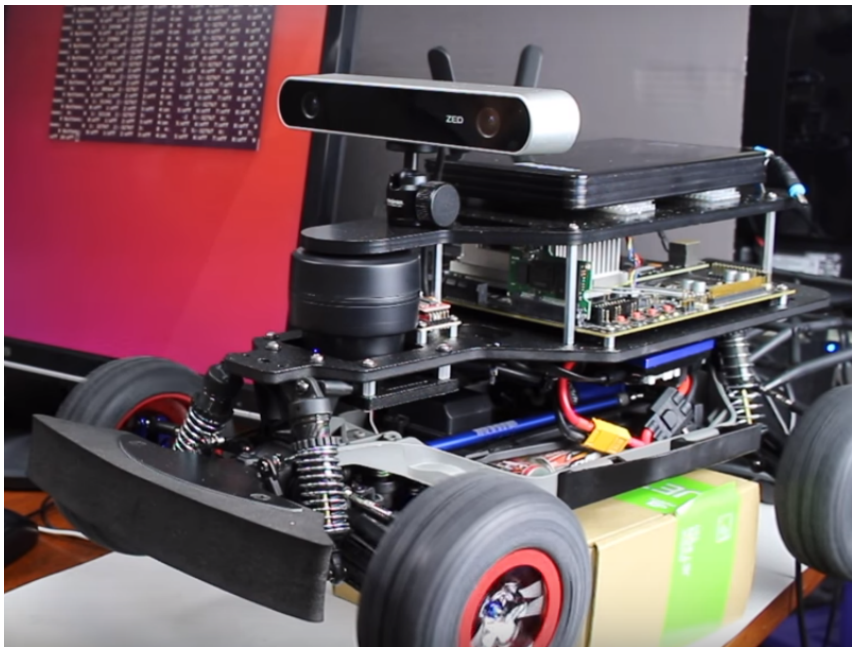
2.2 Phạm vi ứng dụng

NVIDIA Jetson Developer Kit cho các nhà phát triển nhiều công cụ trong các ứng dụng robot, xe tự hành, thiết bị bay không người lái (UAV) các tác vụ xử lý hình ảnh có liên quan đến học máy hay trí tuệ nhân tạo. Do những đặc tính mạnh mẽ về phần cứng thích hợp trong các ứng dụng hỗ trợ AI.

Jetson được sử dụng rộng rãi trong thiết bị bay tự động, drone giao hàng tự động. Thông qua camera được tích hợp trong trên board và bộ xử lý AI mạnh mẽ, thiết bị sẽ nhận diện đường đi, chướng ngại vật, nhận diện người nhận hàng hay mối nguy hiểm xung quanh.

Hình 2.2: Ứng dụng trong máy bay không người lái [19]

Mô hình xe tự lái tìm đường và tránh vật cản sử dụng board Jetson cũng được tạo ra nhằm có thể ứng dụng trong chế tạo robot thăm dò tìm kiếm cứu nạn trên những địa hình khó khăn, đồng thời có thể tự vận hành, thu thập dữ liệu.



Hình 2.3: Ứng dụng trong mô hình xe [12][6]

2.3 Tình hình phát triển trong nước

FPT là đơn vị duy nhất tại Việt Nam nghiên cứu và phát triển hệ thống phần mềm cho xe tự hành từ đầu năm 2016 và đã thử nghiệm thành công vào cuối tháng 10/2017. Trong dự án này, họ sử dụng mạch xử lý trung tâm là NVIDIA Jetson TX2, ngoài ra còn có ZED Stereo Camera, thiết bị thu phát laser Velodyne Lidar, Arduino UNO, Arduino UNO và CAN Shield. Hiện tại, năng lực xe tự hành của FPT đang hướng tới cấp độ 2 trên 5 cấp độ theo SAE.

Hình 2.4: Xe tự hành của FPT [4][3]

Bên cạnh đó, FPT còn tổ chức cuộc thi "Cuộc đua số" – sân chơi công nghệ dành cho sinh viên tại Việt Nam. Đây là cơ hội cho các bạn sinh viên được nghiên cứu, sáng tạo, trải nghiệm công nghệ mới nhất trong lĩnh vực tự hành, đồng thời được tham gia các khóa đào tạo với những chuyên gia hàng đầu của FPT. Theo đó, thí sinh tham gia sẽ được tiếp xúc với công nghệ mới nhất trên thế giới, giúp sinh viên có những định hướng xu thế phù hợp trong tương lai.

Hình 2.5: Cuộc thi "Xe tự hành" [3]

2.4 Tình hình phát triển trên thế giới

Mặc dù có những ưu điểm nổi bật trong khả năng xử lý AI, tuy nhiên việc ứng dụng Jetson TX2 vào trong câc ứng dụng thực tế là khá hạn chế. Các ứng dụng chỉ dừng lại ở các mẫu (Prototype) trong phòng thí nghiệm để giải quyết một phần tác vụ trong một hệ thống lớn, ngoài ra có những nghiên cứu hướng đến cấu trúc bên trong phần cứng nhằm đánh giá hiệu quả áp dụng cho từng ứng dụng cụ thể. Hoặc chỉ dừng lại ở dự án cá nhân, đồ án của nhóm sinh viên.

- Nghiên cứu kiểm chứng quá trình lập lịch trong Jetson TX2 [1] để đánh giá khả năng ứng dụng trong các hệ thống tự động của nhóm nghiên cứu đại học Bắc Carolina. Nghiên cứu này đánh giá bộ lập lịch GPU của Jetson TX2 được định hướng theo kiểu FIFO, phân tích theo lịch trình thời gian thực. Tuy nhiên, việc lập lịch này của GPU thường hay xuất hiện những khoảng thời gian trễ, điều này gây trở ngại rất nhiều đến việc ứng dụng các mô hình trong thực tế. Nhóm nghiên cứu hứa hẹn sẽ loại bỏ những khoảng thời gian trễ này trong tương lai bằng cách kết hợp thay đổi các chế độ lập lịch trên GPU.

- Bài nghiên cứu về tính toán năng lượng tiêu thụ trong quá trình thực hiện tác vụ học sâu (DL) của mạch Jetson TX2 [23]. Thông qua mô hình nhận dạng vật thể YOLO bài nghiên cứu đã đánh giá được những ưu thế về mức tiêu thụ năng lượng thấp, đồng thời là giá trị FPS trong lúc chạy chương trình nhận diện. Nghiên cứu được thực hiện cùng với NVIDIA Jetson TX1 và NVIDIA Tesla P40.

Bảng 2.1: Kết quả so sánh trên 3 đối tượng [22]

		Jetson TX1	Jetson TX2	Xeon w/P40
Energy	Normal	0.65472	0.5800	-
	Max clock	0.78400	0.50667	-
mAP/Energy	Normal	0.48908	0.62568	-
	Max clock	0.38998	0.62737	-
FPS	Normal	0.42233	3.02742	41.7037
	Max clock	2.82583	3.34584	-

Nghiên cứu được thực hiện trên cùng mô hình huấn luyện học sâu như nhau. Tesla P40 có chỉ số FPS cao hơn Jetson TX1 và TX2 tuy nhiên năng lượng mà mạch này tiêu thụ là rất lớn. Theo đó Jetson TX2 với chế độ Max - Q tiêu thụ năng lượng ít nhất, do đó được đánh giá là mạch nhận diện với độ chính xác cao và tiêu thụ năng lượng thấp.

- Cuộc thi NVIDIA AI City Challenge [14][25] được tổ chức thường xuyên kể từ năm 2017. Cuộc thi đưa ra những yêu cầu về xử lý nhận diện những vấn đề giao thông trong thành phố. Từ nhận diện, định hướng phương tiện, đếm số lượng, phân vùng và xây dựng mô hình 3D trên Jetson TX2.

Hình 2.6: Cuộc thi NVIDIA AI City Challenge [14]

Chương 3

Tổng quan mô hình YOLO

3.1 YOLO - You Only Look Once

YOLO [22] là mô hình nhận dạng vật thể. Với điểm mạnh là dễ dàng nhận biết được khái quát về đại diện của mỗi đối tượng. Qua đó có thể phát hiện và phân loại chính xác vật thể trong thời gian thực. Với những đặc tính gọn nhẹ, mã nguồn mở, xử lý với tốc độ nhanh YOLO trở thành mô hình tiêu biểu, đặc trưng trong các ứng dụng nhận dạng vật thể.

Không như những mô hình nhận dạng khác (như Fast RCNN) thực hiện việc phát hiện trên các khu vực đề xuất (Region proposal), theo đó sẽ phải thực hiện dự đoán nhiều lần cho các region khác nhau, scale khác nhau trong một ảnh. YOLO sử dụng duy nhất một neural network cho toàn bộ ảnh. Kiến trúc của YOLO tương tự như FCNN (Full Convolution Neural Network), hình ảnh (kích thước n x n) chỉ được truyền qua FCNN một lần duy nhất, sau đó sẽ trả về ở ngõ ra là ảnh có kích thước m x m. Ảnh ở đầu vào sẽ được chia thành các ô lưới (grid cell), và dự đoán các bounding box cũng như xác suất phân loại cho mỗi grid cell. Các bounding box này được đánh trọng số theo xác suất dự đoán.

Một số khái niệm sử dụng trong YOLO:

Grid cell: Trong mô hình YOLO ảnh được chia thành các grid cell, mỗi grid chịu trách nhiệm nhận diện một đối tượng. Cụ thể, mỗi grid cell dự đoán số bounding box cố định đồng thời cũng xác định vị trí. Tuy nhiên, YOLO bị giới hạn khi các vật thể ở quá gần nhau thì khả năng xác định sẽ bị giảm đi. Trình tự nhận dạng của mỗi grid cell cụ thể như sau:

1. Dự đoán đường bao ngoài và mỗi đường bao có một giá trị tin tưởng
2. Grid cell nhận dạng một vật thể bất kể có bao nhiêu đường viền quanh vật thể đó.
3. Dự đoán xác xuất của vật thể.

Boundary box (Bounding box: mỗi boundary box chứa 5 thành phần: giá trị tọa độ `x, y, w, h` và giá trị tin tưởng [7]. Giá trị tin tưởng (confidence score) ánh xạ tính

khách quan của vật thể trong đường viền và độ chính xác của boundary đó. Giá trị `w` và `h` lần lượt là chiều rộng và chiều cao của boundary, `x, y` lần lượt là tọa độ điểm phiá trên bên trái của boundary. **Unified Detection**: YOLO thống nhất toàn bộ các thành phần riêng biệt trong object detecion vào một mạng neural duy nhất, sử dụng các feature từ toàn bức ảnh để dự đoán mỗi bounding box với tất cả các lớp. Bức ảnh được chia thành S x S grid, nếu tâm của object nằm trong một grid cell, grid cell đó sẽ chịu trách nhiệm phát hiện ra object này.

Mỗi grid cell dự đoán B bounding box và điểm số tin cậy (confidence score) tương ứng mỗi box. Confidence score phản ánh độ tin cậy của việc object nằm trong box, đồng thời là độ chính xác, đô khớp của box so với object. Độ tin cậy được định nghĩa bởi công thức:

$$Pr(obj) * IOU_{pred}^{truth} \tag{3.1}$$

Với:

$Pr(obj)$ giá trị trong khoảng [0, 1] thể hiện khả năng box có chứa object.

IOU_{pred}^{truth} là giá trị Intersection Over Union, thể hiện độ khớp của prediction box so với ground-truth box.

Trong trường hợp không tồn tại object nào trong grid cell, các confidence score có giá trị bằng không.

Mỗi grid cell đồng thời cũng dự đoán xác suất có điều kiện của class cho trước, $Pr(Class_i|Object)$. Xác suất này phụ thuộc việc tồn tại object ở grid cell, và chỉ dự đoán xác suất tương ứng mỗi grid cell, không cần quan tâm đến bounding box.

Khái niệm chính của YOLO là xây dựng mạng CNN để dự đoán tensor (7,7,30). Sử dụng mạng CNN để giảm kích thước ma trận 7 x 7 với 1024 ngõ ra. YOLO sử dụng thuật toán hồi quy tuyến tính [13] sử dụng 2 lớp kết nối để tạo ra dự doán đường viền. Để tiến hành dự đoán cuối cùng, dựa vào chỉ số tin tưởng đáp ứng ngưỡng cho trước sẽ thu được kết quả nhận diện.

Confidence score của lớp cho mỗi dự đoán được tính bằng tích của giá trị box confidence score và condition class probaility.

3.2 Hàm mất mát

YOLO dự đoán nhiều đường viền trên mỗi grid cell. Để tính toán sự mất mát, cần lựa chọn giá trị IoU (Intersection over union) là cao nhất. Phương pháp này giúp việc dự đoán trở nên tốt hơn ngay cả ở các kích cỡ hoặc tỉ lệ của đường viền khác nhau.

YOLO sử dụng SSE (Sum square error) [26] để tính toán sự mất mát, hàm mất mát bao gồm: mất mát phân loại (classification loss), mất mát bên trong (localization loss) và confidence loss.

- Classification loss:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \epsilon classes} \left(p_i\left(c\right) - \hat{p}_i\left(c\right)\right)^2 \tag{3.2}$$

  Với:

  - $\mathbb{1}_i^{obj} = 1$ nếu vật thể có trong cell thứ i, ngược lại thì bằng 0.
  - $\hat{p}_i\left(c\right)$ biểu thị xác xuất điều kiện cho lớp c trong cell i.

- Localization loss: Giá trị mất mát localization tính toán lỗi trong dự đoán boundary về vị trí và kích cỡ.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(x_i - \hat{x_i}\right)^2 + \left(y_i - \hat{y_i}\right)^2\right]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{\omega_i} - \sqrt{\hat{\omega_i}}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h_i}}\right)^2\right] \tag{3.3}$$

  Với:

  - $\mathbb{1}_i^{obj} = 1$ nếu đường viền thứ j trong cell i có chứa vật thể, ngược lại thì bằng 0.
  - λ_{coord} gia tăng giá trị mất mát trong hệ tọa độ của boundary.

- Confidence loss: Khi vật thể được nhận diện, giá trị mất mát confidence đánh giá sự tương quan của vật thể trong boundary

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_i^{obj}\left(C_i-\hat{C}_i\right)^2 \tag{3.4}$$

Với:

- $\hat{C}_i$ là chỉ số confidence của boundary j trong cell i.
- $\mathbb{1}_{ij}^{obj}=1$ khi đường viền thứ j trong cell i có chứa vật thể, ngược lại thì bằng 0.

Nếu vật thể không thể tìm thấy, mất mát confidence là:

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_i^{noobj}\left(C_i-\hat{C}_i\right)^2 \tag{3.5}$$

- Hàm mất mát của toàn bộ quá trình sẽ là tổng của mất mát localizationm, mất mát confidence, mất mát classification.

3.3 Đặc điểm

Mô hình có nhiều điểm vượt trội so với các hệ thống phân loại cơ bản (classifier-based). Tại giai đoạn nhận diện, YOLO sẽ "nhìn" toàn bộ bức ảnh (thay vì từng phần bức ảnh), vì vậy những dự đoán sẽ được cung cấp thông tin bởi nội dung toàn cục của bức ảnh. Ngoài ra, dự đoán được đưa ra chỉ với một mạng đánh giá duy nhất, thay vì rất nhiều mạng đối với mô hình R-CNN. Do đó tốc độ của YOLO là cực nhanh, đáng kể hơn so với RCNN, Fast RCNN.

Sử dụng phương pháp giới hạn vùng đề xuất, YOLO truy xuất vào hình ảnh trong việc dự đoán boundary, nhờ đó hạn chế nhận diện sai đối tượng. YOLO xác định vật thể trên mỗi grid cell, điều này sẽ đa dạng hóa không gian trong việc đưa ra dự đoán.

YOLO áp đặt các ràng buộc về không gian trên những bounding box, mỗi grid cell chỉ có thể dự đoán rất ít bounding box và duy nhất một lớp. Các ràng buộc này hạn chế khả năng nhận biết số object nằm gần nhau, cũng như đối với các object có kích thước nhỏ.

YOLO sử dụng các đặc tính tương đối thô để dự đoán bounding box, do mô hình sử dụng nhiều lớp downsampling từ ảnh đầu vào. Bởi các hạn chế này của mô hình khi huấn

luyện để dự đoán bounding box từ dữ liệu, dẫn đến YOLO không thực sự tốt trong việc nhận diện các object với tỉ lệ hình khối mới hoặc bất thường so với tập data. YOLOv2 [21] đã khắc phục phần nào vấn đề này, nhưng vẫn thua kém nhiều so với Fast RCNN.

Ngoài ra, trong quá trình huấn luyện, hàm mất mát không có sự đánh giá riêng biệt giữa lỗi của bounding box kích thước nhỏ so với lỗi của bounding box kích thước lớn. Việc coi chúng như cùng loại và tổng hợp lại làm ảnh hưởng đến độ chính xác toàn cục của mạng. Lỗi nhỏ trên box lớn nhìn chung ít tác hại, nhưng chỉ số lỗi nhỏ với box rất nhỏ sẽ đặc biệt ảnh hưởng đến giá trị IOU.

Chương 4

Xây dựng mô hình Yolo trên Jetson TX2

Như đa số các hệ thống nhúng hay máy tính nhúng khác, các tác vụ cụ thể cần phải được chạy trên hệ điều hành riêng biệt. Đối với Jetson TX2 điều này cũng không ngoại lệ. Sự cần thiết về quản lý các tài nguyên trong xây dựng và khởi tạo các tác vụ cũng như việc hỗ trợ các công cụ xử lý tiến tác vụ được Jetpack đáp ứng tốt cho mạch Jetson.

4.1 Cài đặt hệ điều hành

NVIDIA Jetson TX2 sử dụng hệ điều hành Jetpack. Đây là là bộ công cụ phát triển phần mềm hỗ trợ các giải pháp xây dựng các ứng dụng trí tuệ nhân tạo (AI) [15]. Jetpack hỗ trợ các gói thư viện ứng dụng, framework hay bộ công cụ hỗ trợ cho việc phát triển các ứng dụng liên quan, cụ thể:

- **TensorRT [17]**: Công cụ cung cấp khả năng suy luận trong các mô hình học sâu, nhận dạng, phân loại hay nhận dạng vật thể qua mạng thần kinh nhân tạo. TensorRT giúp tăng đáng kể tốc độ suy luận và xử lý các tiến trình.
- **cuDNN**: Thư viện Cuda Deep Neural Network hỗ trợ, làm tăng hiệu suất cho framework học sâu, hỗ trợ sử dụng các hàm và biến đổi tensor.
- **CUDA**: Bộ công cụ cung cấp môi trường phát triển toàn diện trên nền tảng C và C++ với mục đích là tăng tốc độ xử lý trong các ứng dụng. Bộ công cụ gồm trình biên dịch cho GPUs, thư viện toán học và công cụ debugging tối ưu hóa cho các ứng dụng.
- **Multimedia API**: Ứng dụng API camera với thông số điều khiển ở mỗi khung hình, hỗ trợ EGL [28] stream, CSI, ISP, GStreamer, V4L2.
- **Computer Vision**: Bộ phần mềm VisionWorks hỗ trợ cho các ứng dụng thị giác máy tính và xử lý hình ảnh. OpenCV là thư viện mở phổ biến trong thị giác máy tính, xử lý ảnh, học máy cùng với tối ưu tốc độ trong xử lý theo thời gian thực.

- **Developer Tools**: NVIDIA Nsight hỗ trợ phân tích tối ưu quá trình xử lý. NVIDIA Nsight Graphic hỗ trợ sửa lỗi và định hình các ứng dụng đồ họa.

4.1.1 Chuẩn bị

Quá trình cài đặt hệ điều hành chuẩn bị khá nhiều các thiết bị, bao gồm:

- Máy chủ sử dụng hệ điều hành Ubuntu (v18.04 hoặc v16.04), trong đề tài sử dụng phiên bản 16.04.
- USB Mirco-B kết nối từ mạch Jetson đến máy chủ Linux.
- Thiết bị ngoại vi: Bàn phím và chuột kết nối với mạch Jetson.
- Cáp HDMI cho ra màn hình hiển thị giao diện đồ họa của mạch Jetson.
- Cáp Ethernet kết nối với card mạng của Jetson (sử dụng cùng router với máy host).

Ghi chú: Các thiết bị ngoại vi (bàn phím và chuột), cáp HDMI không bắt buộc trong quá trình cài đặt Jetpack, nhưng cần thiết trong giai đoạn sau cài đặt.

4.1.2 Cài đặt

Trong đề tài sử dụng máy host với Ubuntu v16.04 chạy trên máy ảo VMware [27] cần phải cài đặt mạng phù hợp để có thể sử dụng địa chỉ IP trực tiếp từ router. Chọn

`''Bridge: Connected directly to the physical network''`

trong mục cài đặt mạng.

Tạo tài khoản và tải xuống phiên bản Jetpack trên máy host từ trang chủ của NVIDIA [18] với phiên bản phù hợp, điều này rất quan trọng do mỗi phiên bản Jetpack được thiết kế phù hợp cho từng nền tảng phần cứng. Với phiên bản Jetpack 3.3 phù hợp cho Jetson TX2.

Sau khi tải xuống, mặc định file Jetpack chưa thể được thực thi. Cần cấp quyền thực thi cho file này, mở cửa sổ Terminal tại nơi lưu file và chạy lệnh:

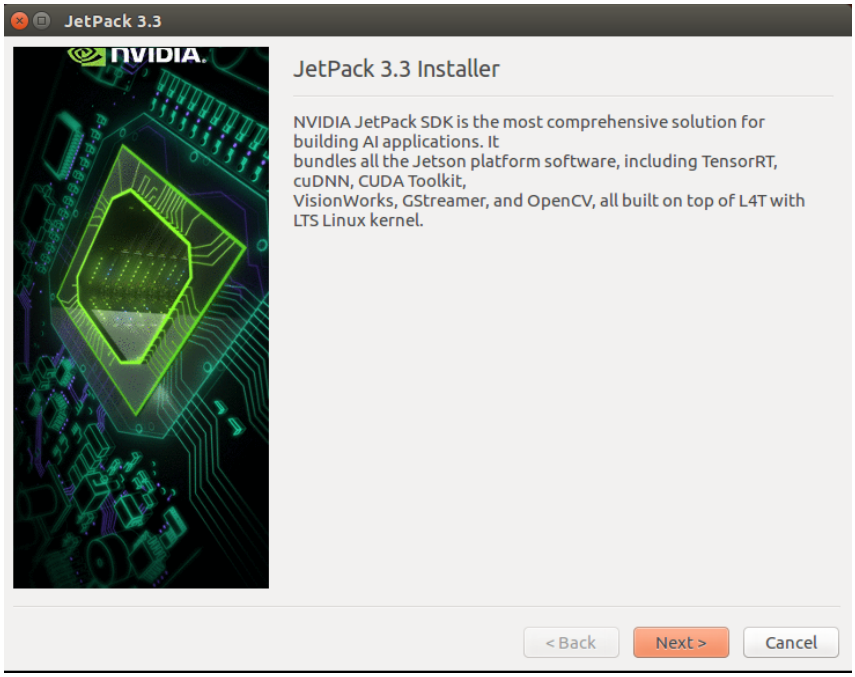
```
$ chmod 777 JetPack-${VERSION}.run
```

Sau khi cấp quyền truy cập cho file, chạy file cài bằng lệnh:

```
$ sudo ./JetPack-${VERSION}.run
```

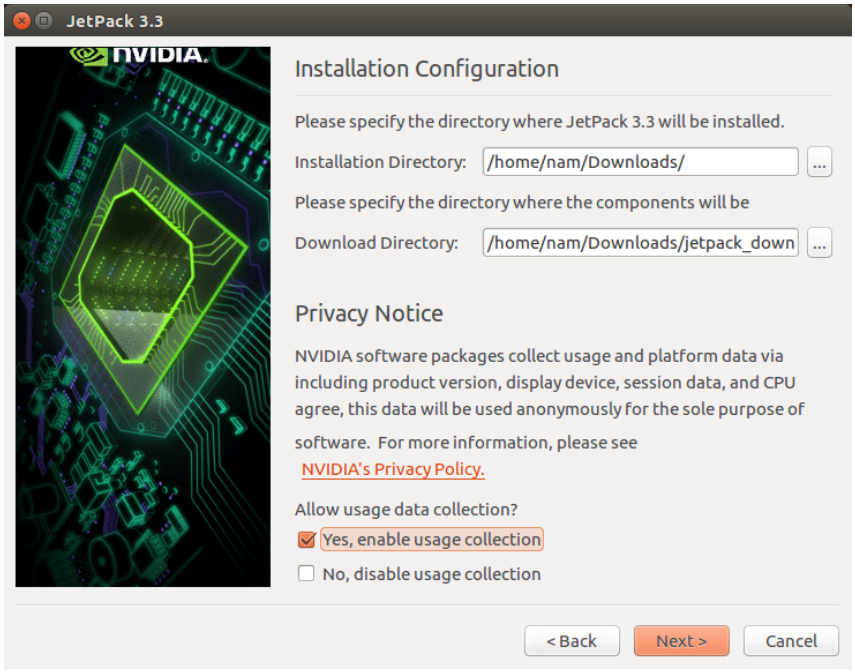
Hệ thống sẽ yêu cầu nhập mật khẩu cho lệnh dưới quyền quảng trị root.

Xuất hiện cửa sổ pop-up:



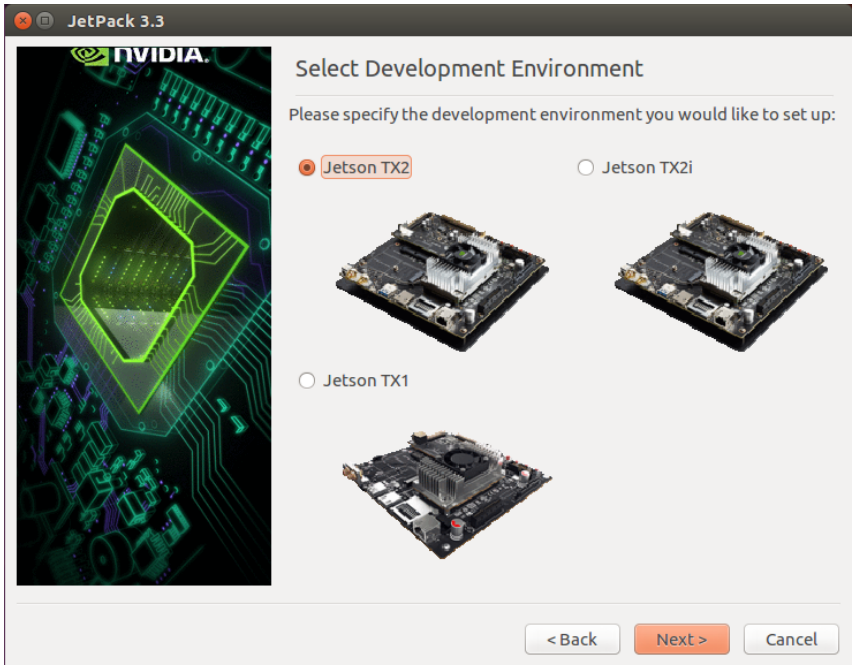
Hình 4.1: Cửa sổ cài đặt Jetpack

Nhấn `"Next"` để chuyển sang cửa sổ tùy chọn vị trí file cài trên máy host và mạch Jetson.



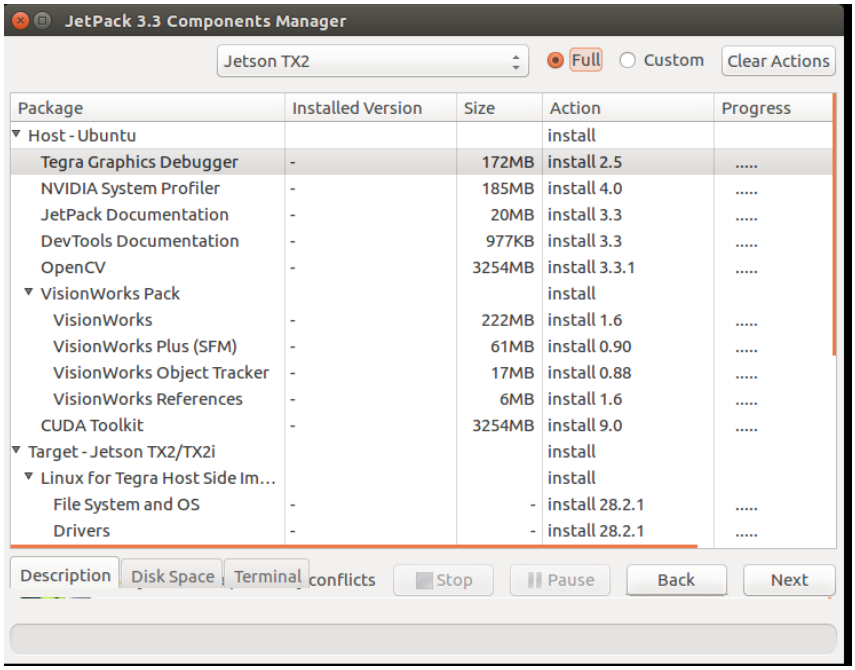
Hình 4.2: Cửa sổ tùy chọn ví trí file lưu

Nhấn "Next" để chuyển sang cửa sổ tùy chọn môi trường cài đặt, chọn Jetson TX2 trong trường hợp này.

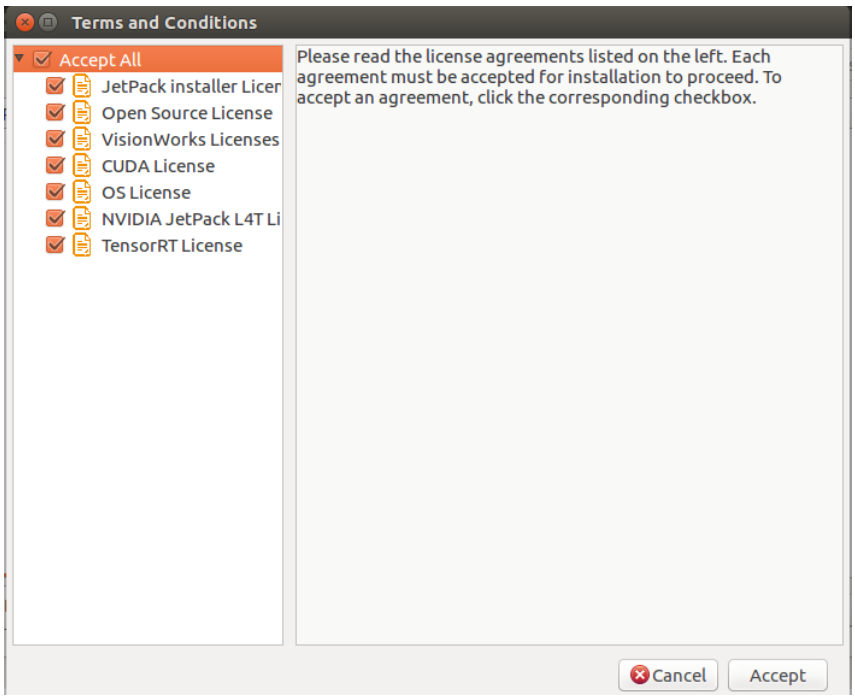


Hình 4.3: Cửa sổ tùy chọn môi trường cài đặt

Cửa sổ tiếp theo hiển thì tùy chọn cài đặt các gói công cụ sẽ cài đặt, có thể thay đổi tùy chọn cài đặt tùy vào từng mục đích.

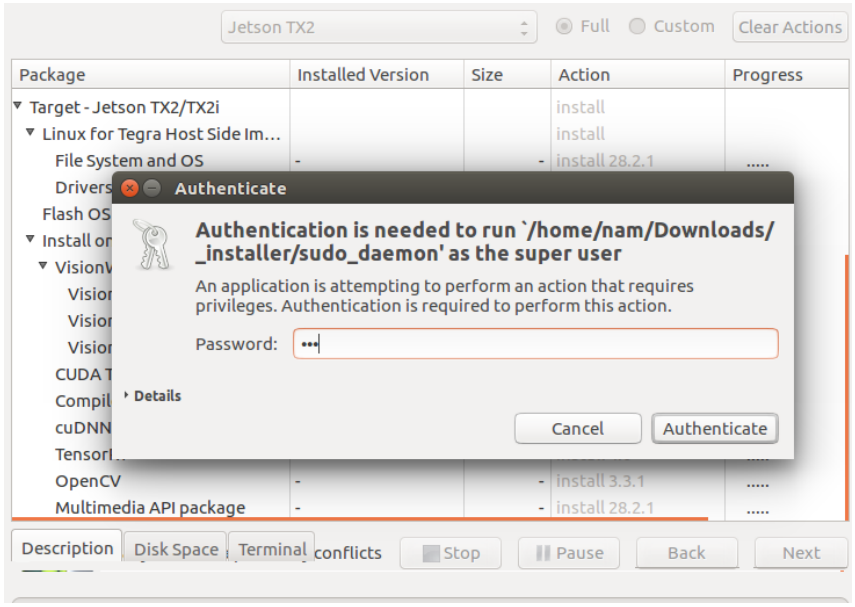


Hình 4.4: Cửa sổ tùy chọn công cụ



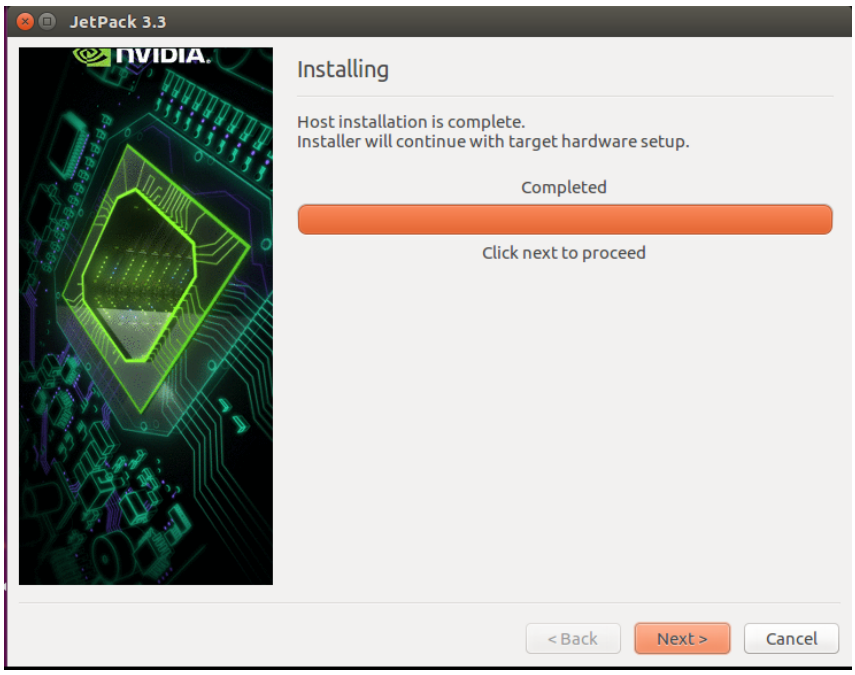
Hình 4.5: Xác nhận điều khoản và bản quyền

Yêu cầu mật khẩu xác thực để bắt đầu quá trình cài đặt.



Hình 4.6: Yêu cầu xác thực để cài đặt các gói phần mềm

Quá trình cài đặt ở bước này sẽ diễn ra lâu hay nhanh tùy thuộc vào số lượng các gói được cài đặt. Thời gian cài đặt trung bình mất 1.5 giờ.

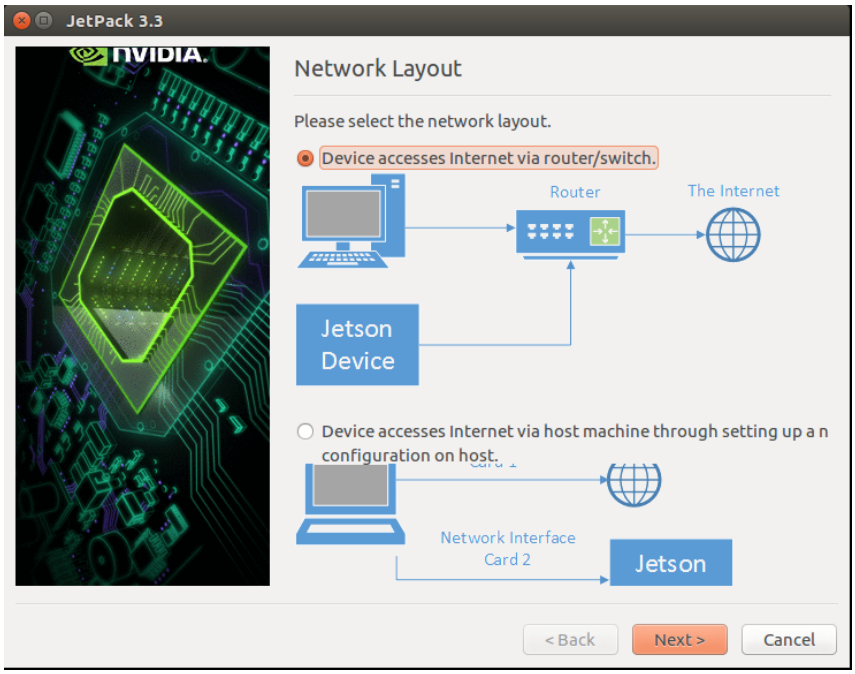


Hình 4.7: Hoàn tất quá trình cài đặt

Tiếp theo nhấn `"Next"` đến với cửa sổ tùy chọn layout mạng. Có 2 kiểu layout:

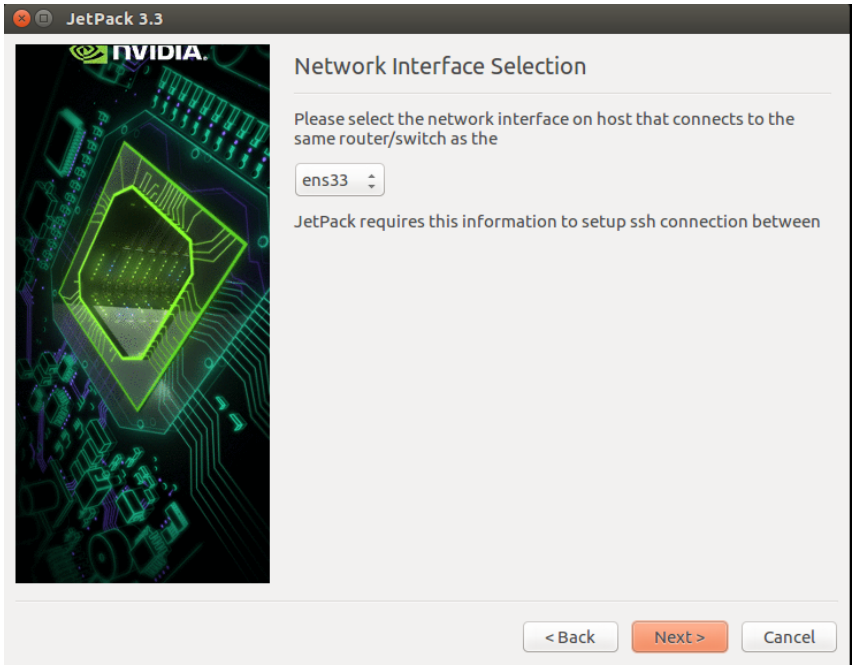
- Kết nối máy host và mạch Jetson với router, lúc này cả hai đều có một địa chỉ IP riêng.
- Mạch Jetson truy cập vào Internet thông qua máy host.

Trong phạm vi báo cáo này kiểu layout đầu tiên được sử dụng.



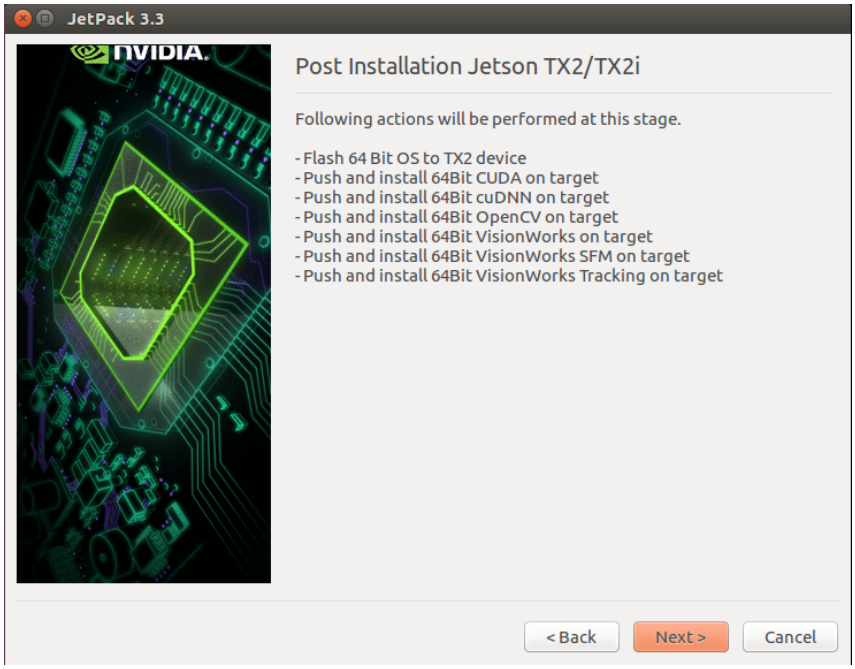
Hình 4.8: Tùy chon layout

Tiếp theo là tùy chọn cổng Interface trên máy host kết nối đến router tên interface ở đây là `ens33`. Tiếp theo, cửa sổ hiển thị các modules sẽ được cài đặt vào board Jetson.



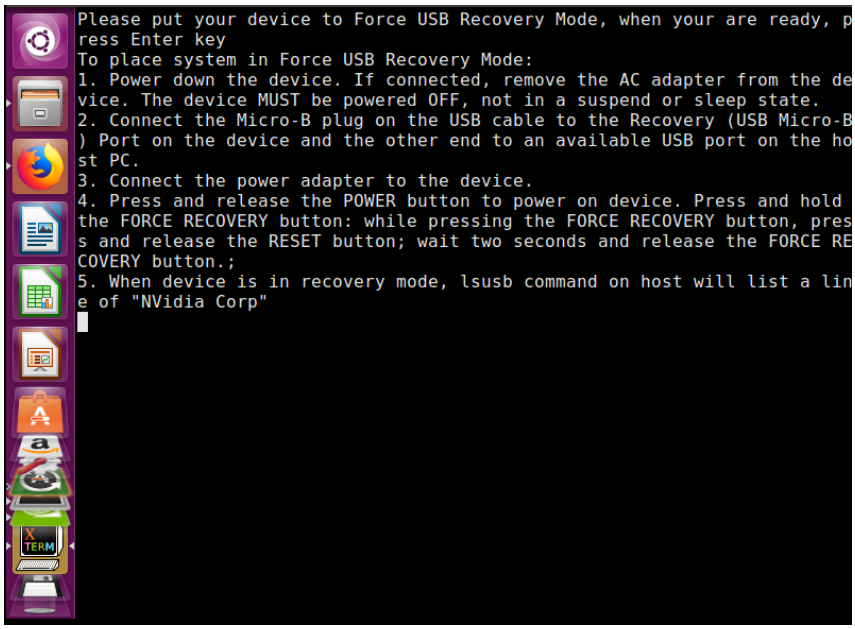
Hình 4.9: Tùy chọn tên Interface

Nhấn `"Next"`, cửa sổ hiện lên yêu cầu kết nối các thành phần với nhau theo layout đã



Hình 4.10: Xác nhận tiến trình cài đặt

chọn trước đó



Hình 4.11: Kết nối các thành phần

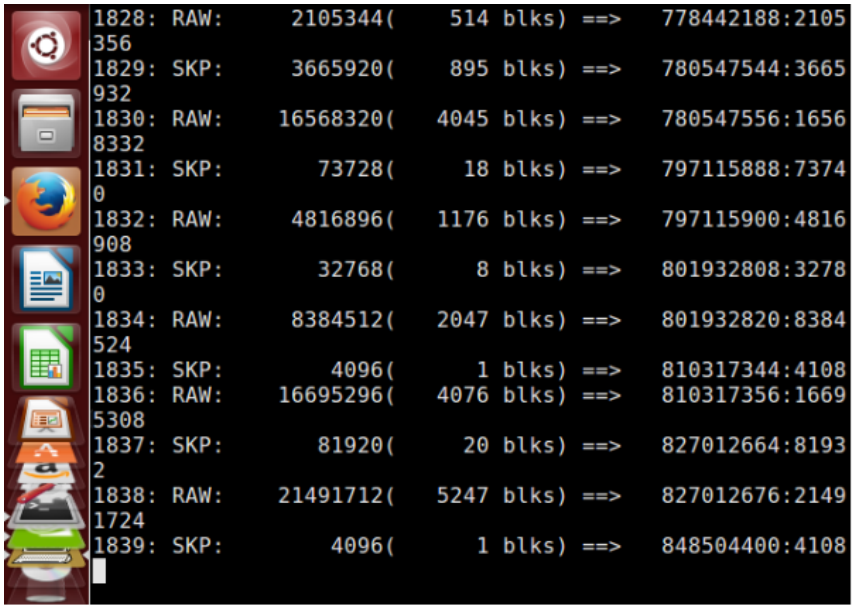
4.1.3 Kết nối phần cứng

Dựa vào cấu trúc loại layout đã chọn để tiến hành kết nối các thành phần với nhau. Trước tiên cần tắt nguồn của Jetson.

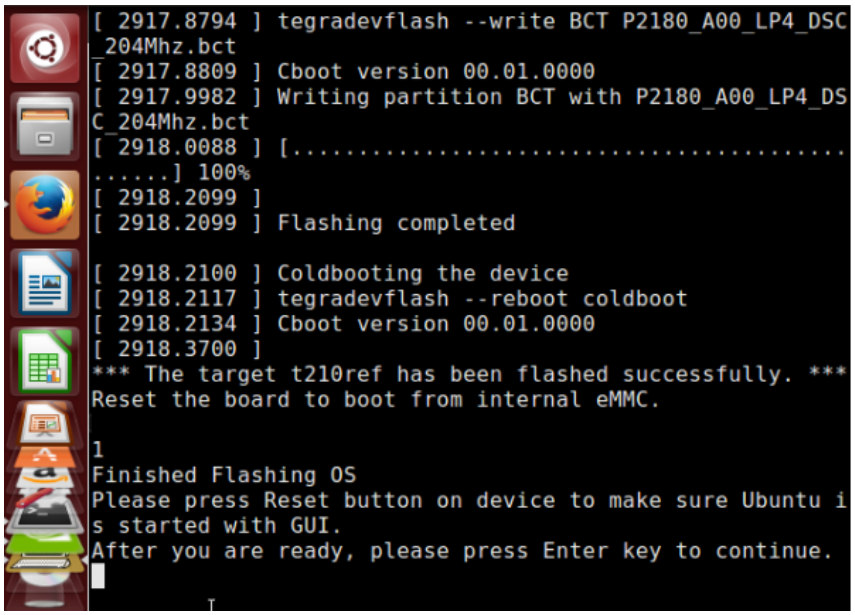
- Tắt nguồn mạch Jetson.
- Kết nối mạng cho mạch Jetson thông qua cáp Ethernet, đồng thời cũng kết nối mạng cho máy host, máy host có thể dùng wifi nhưng để đảm bảo mạng được ổn định, tránh lỗi xảy ra khi cài đặt thì Ethernet là lựa chọn được ưu tiên.
- Nối mạch Jetson và máy host với nhau bằng cáp Micro-B/USB.
- Cấp nguồn mạch Jetson.
- Nhấn nút nguồn để khởi động Jetson như bình thường. Sau đó, nhấn và giữ nút `"RECOVERY FORCE"` đồng thời nhấn `"RESET"`, thả nút `"RESET"` giữ `"RECOVERY FORCE"` trong 2 giây sau đó thả ra.

Kiểm tra interface trên máy host trong của sổ terminal bằng lệnh: `lsusb` kết quả sẽ trả về thông tin tên interface `NVIDIA Corp`.

Nhấn "Enter" để tiếp tục quá trình cài đặt.



Hình 4.12: Tiến trình cài đặt



Hình 4.13: Kết thúc cài đặt

4.2 Cài đặt kernel

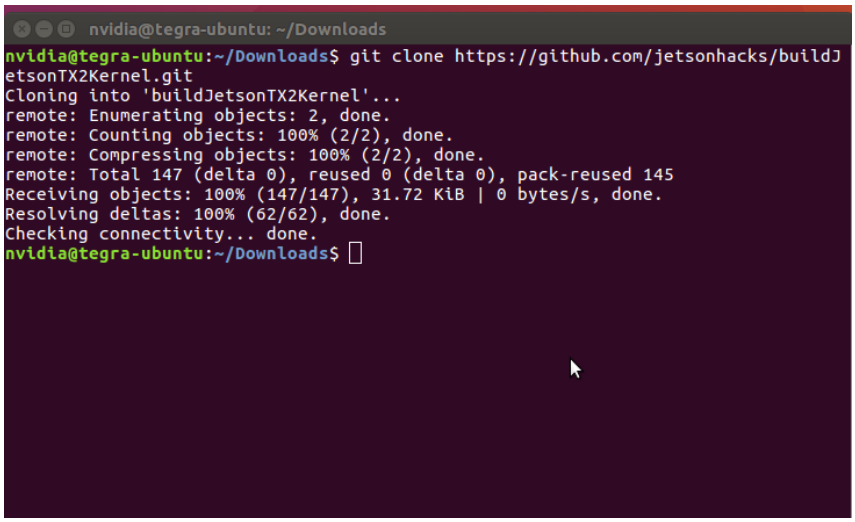
Trong nhiều trường hợp sử dụng, hệ điều hành của chúng ta cần có Kernel để dễ dàng trong việc quản lý bộ nhớ, điều phối các file hay can thiệp vào quá trình điều khiển driver. Kernel có vai trò quan trọng trong biên dịch, thêm hay điều chỉnh các module từ thiết bị.

Để kiểm tra phiên bản kernel hiện tại trên Jetson với lệnh:

`$ uname -r` - với ý nghĩa xuất thông tin về kernel của hệ thống.

Tiến hành cập nhật Kernel, trong cửa sổ Terminal dùng lệnh:

`$ git clone https://github.com/jetsonhacks/buildJetsonTX2Kernel.git` [8]

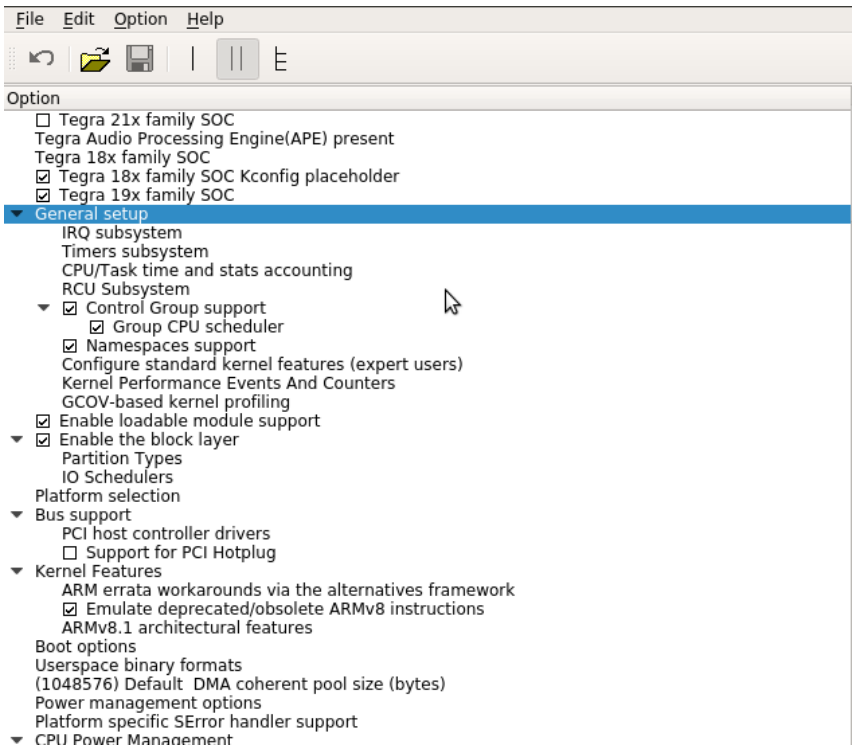


Hình 4.14: Tải xuống thư mục cài đặt

Lệnh: `$ ./getKernelSources.sh`

Tải nguồn kernel từ website NVIDIA developer, sau đó giải nén và lưu tại `/usr/src/kernel`.

Lưu ý: Cần đảm bảo sự đồng bộ trong các phiên bản của L4T và kernel. Việc cập nhật hay cài đặt mới kernel có thể gây ra sự xung đột trong các tiền trình quản lý tác vụ hoạt động cụ thể.



Hình 4.15: Giao diện quản lý driver

Cửa sổ quản lý driver của kernel xuất hiện. Tại cửa sổ này, người dùng có thể thêm mới, thay đổi hoặc điều chỉnh các gói driver. Người dùng sẽ trở nên chủ động hơn trong việc quản lý tiến trình hoạt động của hệ thống hoặc các thiết bị ngoại vi.

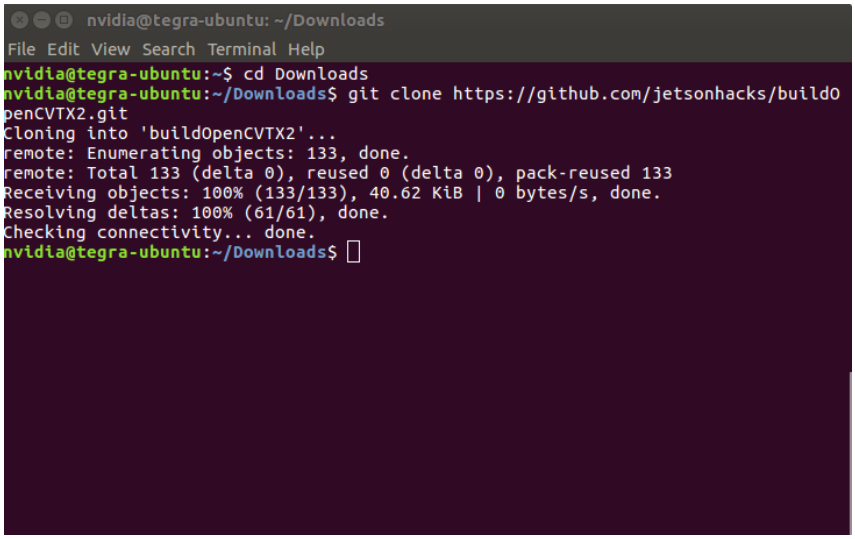
4.3 Cài đặt thư viện OpenCV

OpenCV [2] là thư viện mã nguồn mở, được phát triển bằng ngôn ngữ C/C++, là công cụ phát triển trên Python, Matlab, ... OpenCV được thiết kế nhằm mục đích tăng khả năng tính toán trong các ứng dụng thời gian thực liên quan đến thị giác máy tính. Trong phạm vi đề tài, OpenCV là thư viện chính trong nhận dạng và xác định vật thể.

Một điểm quan trọng, đối với Jetson TX2 như đã đề cập, còn có thêm nhân CUDA. Việc tính toán xử lý sẽ trở nên nhanh hơn khi xây dựng OpenCV với nhân CUDA.

Băt đầu tại cửa sổ terminal với lệnh:

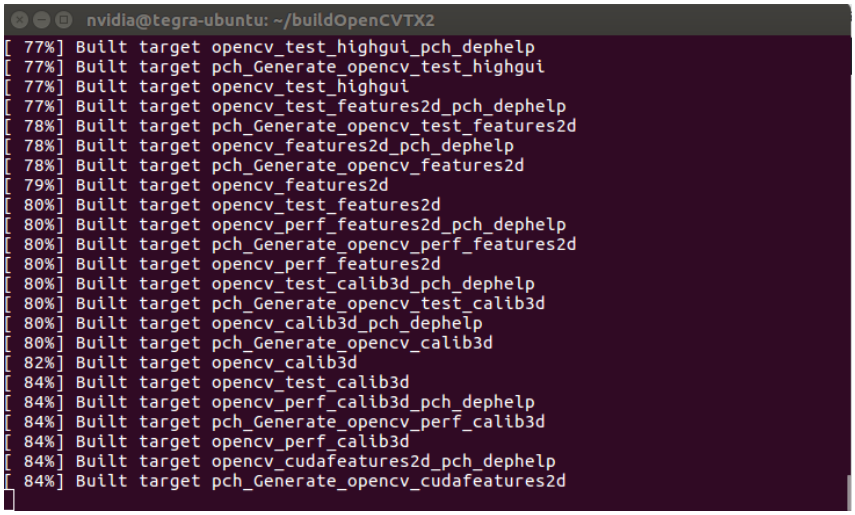
```
$ git clone https://github.com/jetsonhacks/buildOpenCVTX2.git
```



Hình 4.16: Tải xuống gói cài đặt

Chạy tệp `buildOpenCV.sh` để tải xuống và build thư viện OpenCV với lệnh:

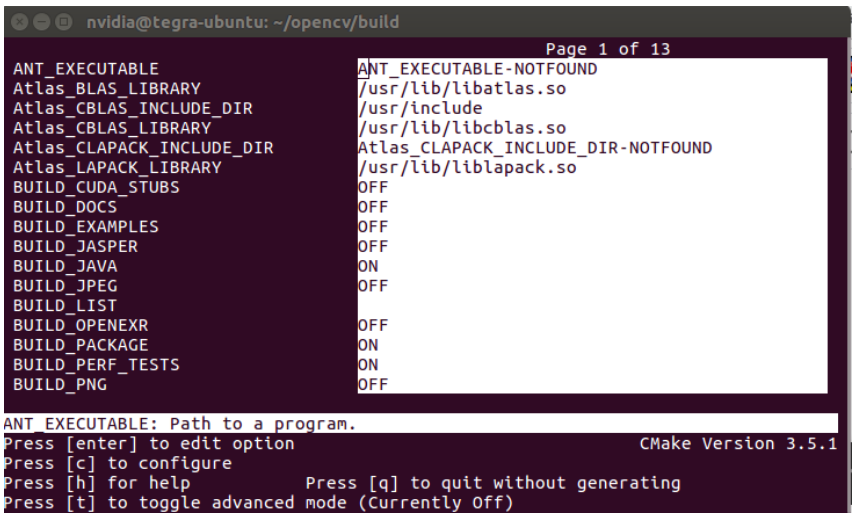
`$ ./buildOpenCV.sh` - quá trình này mất thời gian khá lâu (khoảng 1.5 giờ).



Hình 4.17: Tiến trình cài đặt

Sau khi cài đặt xong, để kiểm tra các module được bổ sung trong lúc cài OpenCV:

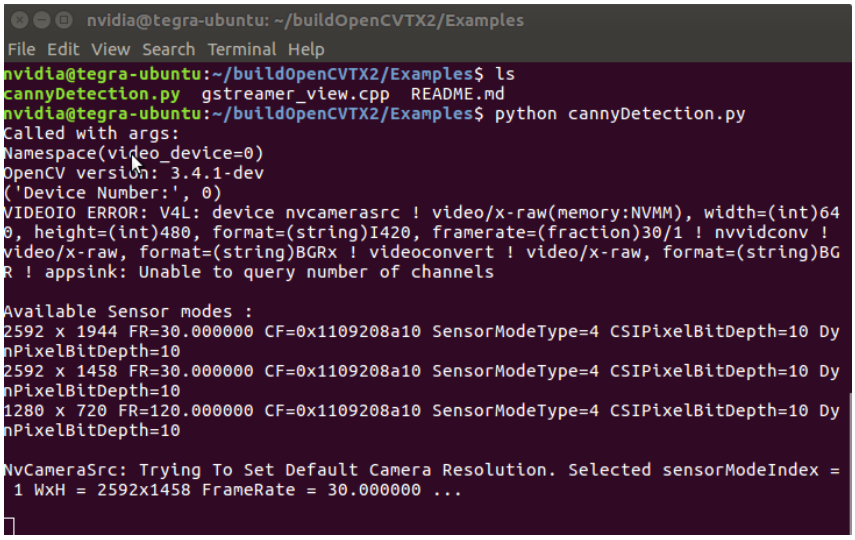
```
$ cd /opencv/build
$ apt search ccmake
$ sudo apt-get install cmake-curses-gui
$ ccmake ..
```



Hình 4.18: Giao diện quản lý packages

Có thể tùy chọn các gói cài đặt trong giao diện này tùy vào nhu cầu sử dụng trong các ứng dụng cụ thể.

Chạy thử ví dụ `cannyDetection.py` trong folder `buildOpenCVTX2/Examples`



Hình 4.19: Khởi chạy chương trình

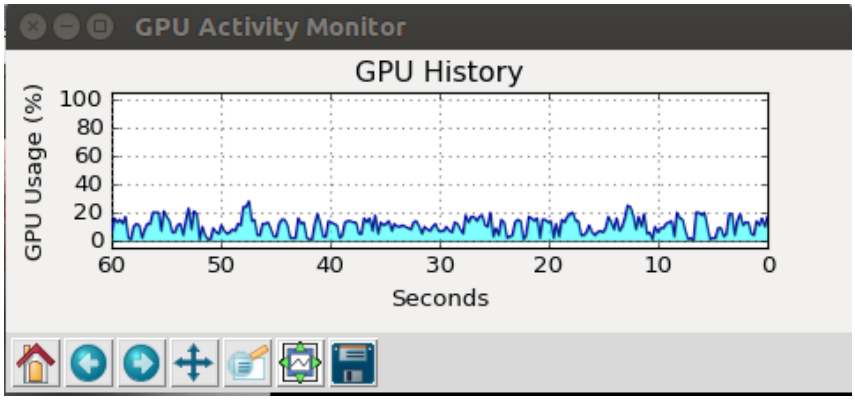
Chương trình chạy thử xác định đường viền của vật thể, chương trình này sử dụng Camera trên board Jetson TX2. Với các thông số phiên bản OpenCV 3.4.1, kích thước 2592 x 1458, số frame trên giây đạt tối đa 30 fps. Các thông số về độ phân giải có thể thay đổi trong chương trình chính.

Hình 4.20: Kết quả hiển thị

Hình 4.21: Một trong các chế độ hiển thị

Các thông tin về giá trị màu của vật thể nhận dạng đường viền RGB, hai thành phần `x` và `v` lần lượt là giá trị trỏ chuột ở trục hoành và trục tung.

Trong quá trình chạy demo nhận dạng, GPU phải hoạt động nhiều hơn mức bình thường do yêu cầu xử lý các đối tượng trong mức khung hình 30 fps.

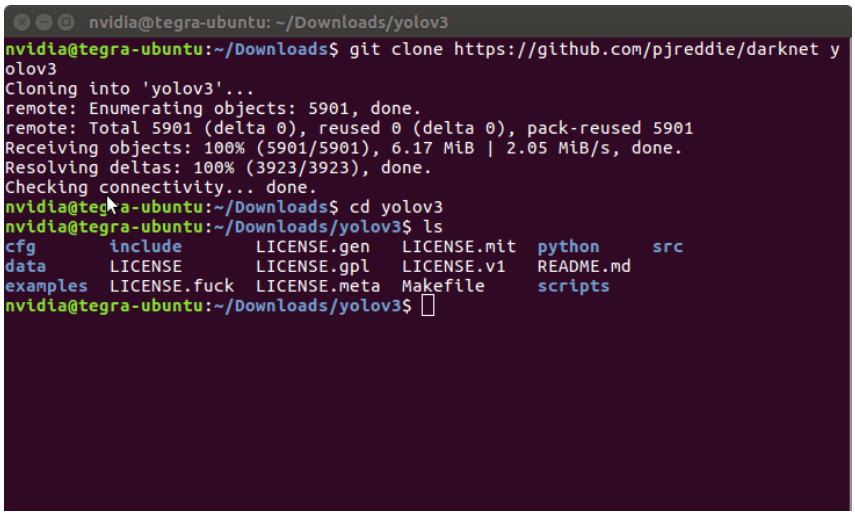


Hình 4.22: Trạng thái GPU

4.4 Xây dựng mô hình YOLO

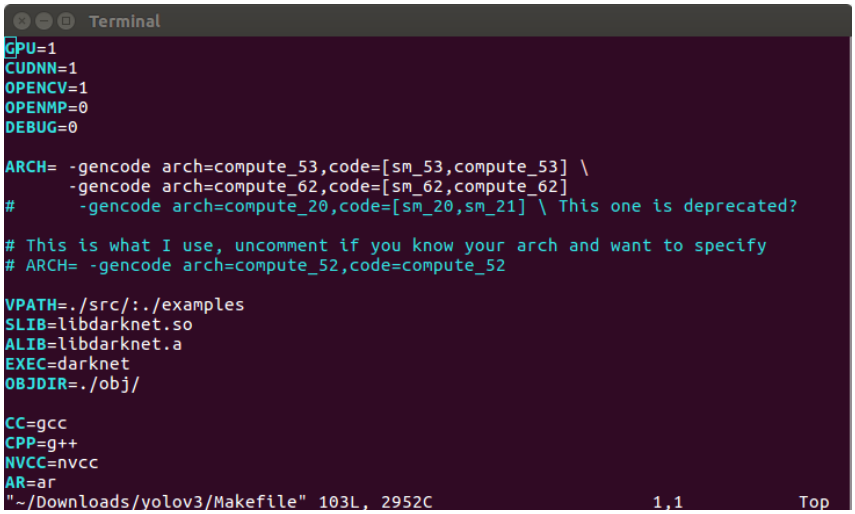
Tải xuống folder cài đặt framework darknet [20] - một mã nguồn mở được viết cho các ứng dụng học sâu (Deep Learning).

```
$ git clone https://github.com/pjreddie/darknet yolov3
```



Hình 4.23: Tải xuống source darknet

Tiếp theo sau, ta đi đến các thông số trong tệp `Makefile`. Makefile là một file dạng script chứa các thông tin liên quan đến cấu trúc project hay các lệnh tạo file [10]. Makefile giúp đơn giản hóa việc biên dịch chương trình trong trường hợp có nhiều module, dễ dàng quản lý hoặc thay đổi. Lệnh `make` sẽ đọc nội dung trong `Makefile`, hiểu kiến trúc của project và thực thi các lệnh.



Hình 4.24: Điều chỉnh tệp Makefile

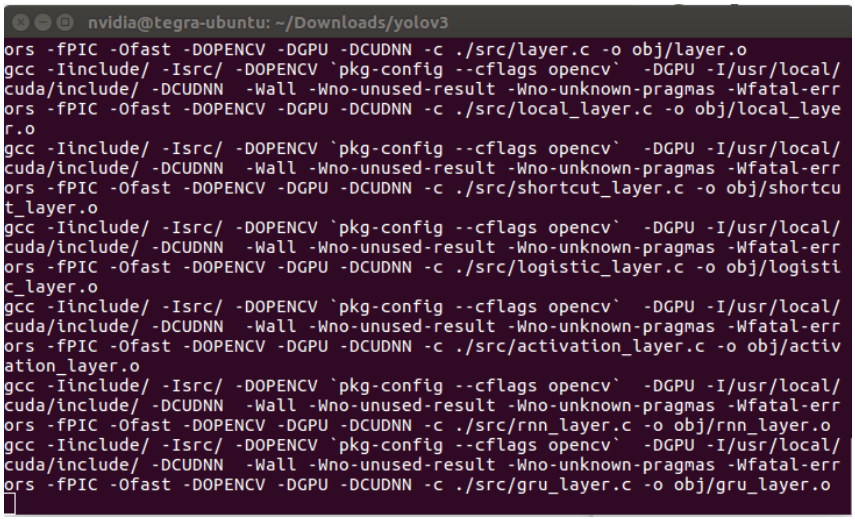
Có thể thấy các thông tin liên quan đến GPU, CUDNN hay OPENCV đều được đặt là 1 để đảm bảo rằng khi chương trình được xây dựng, sẽ hoạt động trên 3 thành phần này. Cụ thể:

- Với GPU - đơn vị xử lý đồ họa hỗ trợcông việc tính toán xác định nhận dạng liên quan đến hình ảnh.
- Với CUDNN - thư viện hỗ trợ trong tính toán các tensor ảnh, liên quan đến neural network.
- Với OPENCV - thư viện xử lý hình ảnh chính trong mô hình YOLO.

Giá trị `arch=compute_62 code=[sm_62,compute_62]` cho Jetson TX2 và `arch=compute_53 code=[sm_53,compute_53]` cho Jetson TX1.

Điều chỉnh chế độ max clock và build bằng lệnh `make`.

```
$ sudo nvpmodel -m 0
$ make
```



Hình 4.25: Xây dựng mô hình darknet

Tùy chỉnh các thông số ở file `yolov3.cfg` tại `yolov3/cfg`. Các thông số liên quan [9]:

`batch` chỉ thị `batch size` tròn suốt quá trình huấn luyện. Bộ dữ liệu huấn luyện lên đến vài trăm hình ảnh, tuy nhiên các hình ảnh này không được huấn luyện cùng nhau. Tiến trình huấn luyện liên quan đến sự lặp lại, cập nhật kích thước của mạng thần kinh (neural netwok) dựa trên tỉ lệ lỗi phát sinh trong tiến trình huấn luyện. Không cần thiết phải dùng tất cả các ảnh trong bộ dữu liệu để cập nhật kích thước của mạng nơ-ron, thay vào đó số lượng ảnh trong bộ dữ liệu dùng để cập nhật lại mạng nơ-ron có thể thay đổi cho phù hợp, giá trị `batch size` đặc trưng cho số lượng này.

Giá trị `subdivisions` đặc trưng cho số ảnh xử lý cùng lúc trong phạm vi giá trị `batch`, điều chỉnh khi sử dụng với giá trị `batch` trong vấn đề về bộ nhớ với GPU.

Thông số về kích thước từng ảnh `width` và `height` là khuôn mẫu kích thước chung. Ảnh từ bộ dữ liệu sẽ được đưa về các kịch thước này trước khi vào bước huấn luyện. Kích thước càng lớn thì giá trị nhận được càng tốt, tuy nhiên thời gian sẽ dài hơn trong tiến trình huấn luyện. Giá trị ngưỡng màu `channels = 3` chỉ thị kênh màu RGB của ảnh đầu vào.

File cấu hình còn có các thông số quyết định cách cập nhật kích thước của mạng nơ-ron. Giá trị `momentum` kiểm xoát kích thước mạng nơ-ron khi quá lớn, `decay` kiểm soát giá trị `momentum`.

Giá trị `learning_rate` đặc trưng cho tốc độ học từ dữ liệu trong `batch`, giá trị này nằm trong khoảng 0.01 đến 0.001.

Sau khi đã cài đặt và cấu hình phù hợp, tải xuống mô hình huấn luyện và chạy video nhận diện.

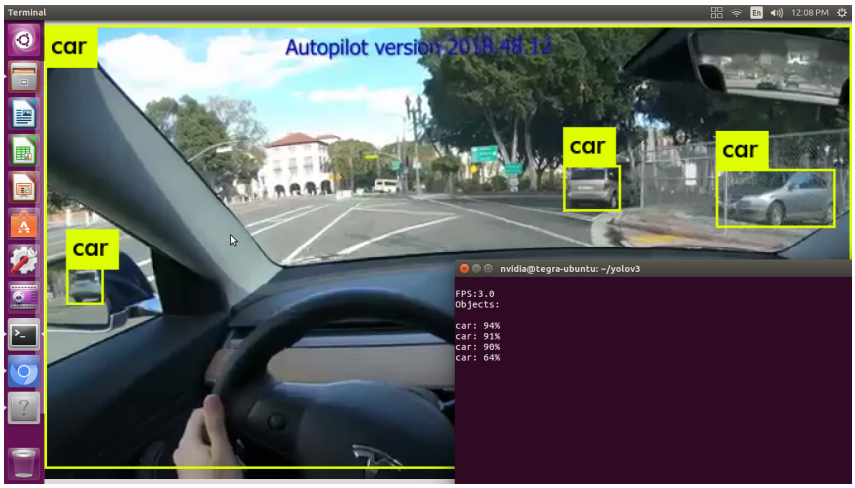
```
[net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[convolutional]
batch_normalize=1
```

Hình 4.26: Cấu hình thông số

```
$ wget https://pjreddie.com/media/files/yolov3.weights
$ ./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
traffic.mp4
```

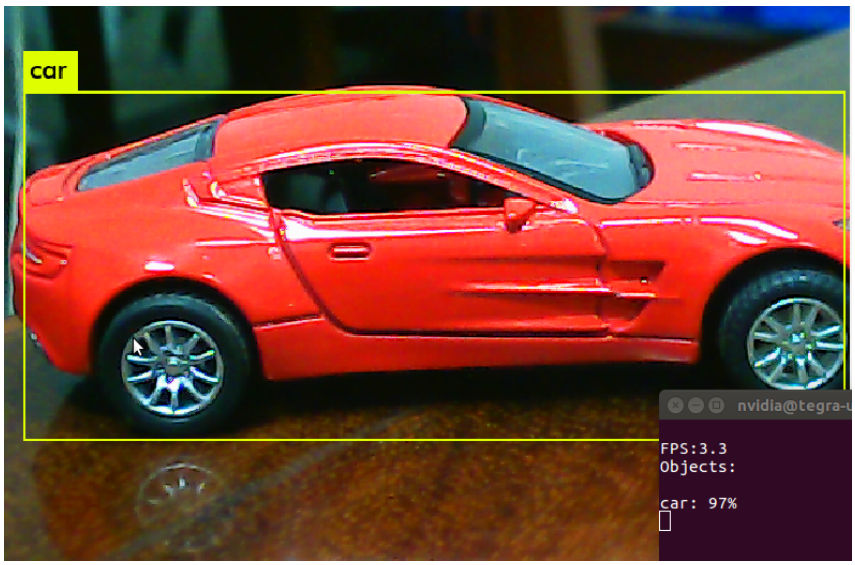


Hình 4.27: Nhận dạng đối tượng trong video [29]

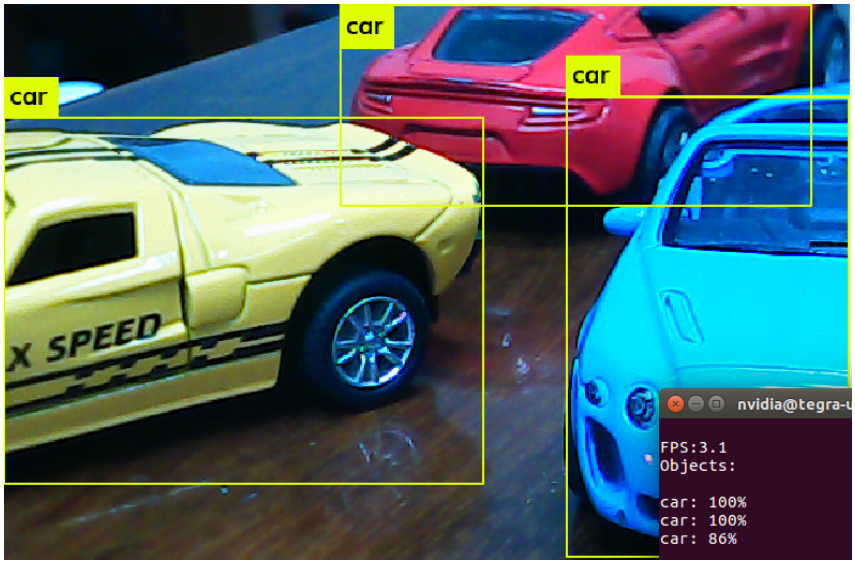
Ta sẽ có thông tin về giá trị khung hình (FPS) trong lúc nhận diện, tên và xác xuất các đối tượng được tìm thấy trong khung hình.

Khi sử dụng stream trực tiếp trong nhận dạng đối tượng được tiên hành bởi lệnh:

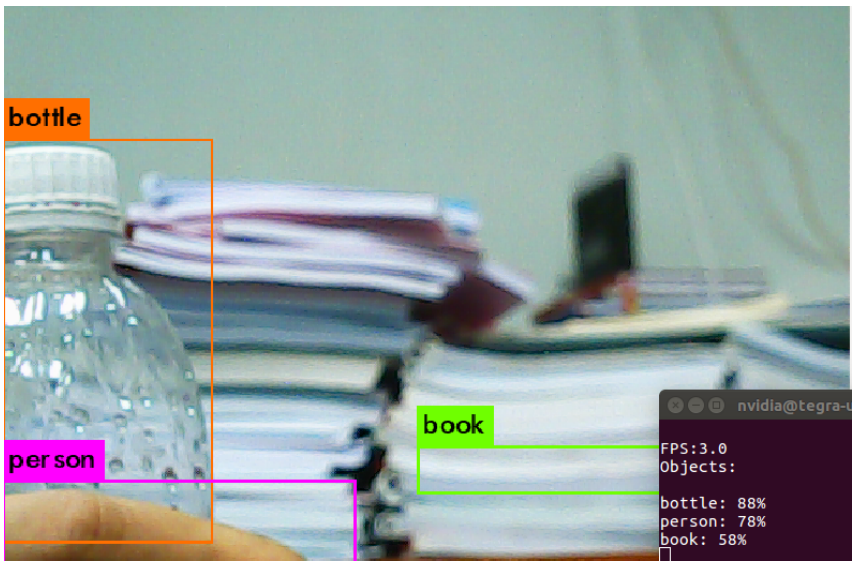
```
$ ./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
-c 1
```



Hình 4.28: Kết quả nhận dạng (1)



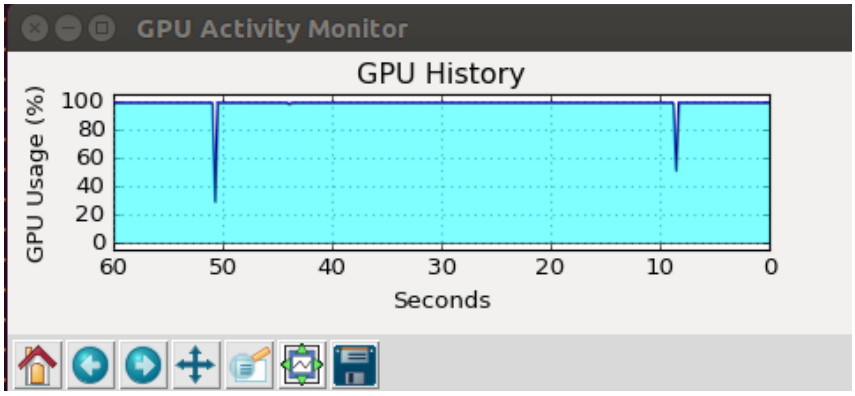
Hình 4.29: Kết quả nhận dạng (2)



Hình 4.30: Kết quả nhận dạng (3)

4.5 Đánh giá kết quả

Do hoạt động ở chế độ max clock nên trạng thái hoạt động của GPU dường như là tối đa.



Hình 4.31: Trạng thái hoạt động của GPU

Về cơ bản khi triển khai mô hình nhận dạng vật thể YOLO trên NVIDIA Jetson TX2, với yêu cầu về khả năng nhận dạng, việc gán nhãn cho vật thể đạt được yêu cầu. Do tính chất của mô hình nhận dạng, các đối tượng nhận dạng với bộ dữ liệu lớn nhưng không tập trung vào các chi tiết trên vật thể hay các chi tiết phụ khác. Điều này một phần phụ thuộc vào tiêu cự của camera, chất lượng khung hình bắt được. Giá trị khung hình (FPS) dao động từ 3.0 đến 3.4. Thời gian phản hồi trong trường hợp vật thể thay đổi trạng thái nhanh là rất dài nên khó có thể sử ứng dụng trong các hệ thống yêu cầu tốc độ cao. Xác suất đối tượng ở mức chấp nhận được từ 60% đến 98%.

Bảng 4.1: Đánh giá kết quả mô hình YOLO trên Jetson TX2

Yếu tố	Đặc điểm chế	Đánh giá
Thời gian nhận dạng	Phản hồi tốt	Chấp nhận được
Gán nhãn đối tượng	Phân biệt rõ đối tượng	Chấp nhận được
Tỉ lê khung hình	Giá trị FPS còn thấp	Hạn chế
Xác xuất	Thấp khi vật thể ở xa	Hạn chế
Chất lượng khung hình	Tiêu cự ngắn	Hạn chế

4.6 Định hướng cải thiện

Trong các ứng dụng cụ thể, bộ dữ liệu huấn luyện có tập trung vào nhóm các đối tượng nhằm thu nhỏ phạm vi nhận dạng. Từ đó tốc độ khung hình có thể tăng lên cùng với cải thiện chất lượng khung hình thông qua nâng cấp camera. Tốc độ di chuyển của đối tượng sẽ ít bị ảnh hưởng đến việc nhận dạng của mô hình rất nhiều.

Tăng chất lượng bộ dữ liệu huấn luyện cũng góp phần nâng cao khả năng nhận dạng của mô hình, giảm thời gian phản hồi từ lúc camera bắt khung hình đến khi nhãn được gán trên đối tượng.

KẾT LUẬN

NVIDIA Jetson TX2 là một máy tính nhúng được phát triển đặc biệt cho các ứng dụng liên quan đến bài toán nhận dạng, thị giác máy tính hay là trí tuệ nhân tạo. Jetson TX2 là một nền tảng lí tưởng cho các ứng dụng nhận dạng cơ bản và phổ biến, đảm nhận một chức năng trong các hệ thống ứng dụng thực tế. Với hệ điều hành tùy biến dựa trên Linux (Jetpack), mạch phát triển đã phát huy được những lợi thế có được. Nhân CUDA là thành phần quan trong quyết định đến sự khác biệt này so với các mạch phát triển khác.

Mô hình nhận dạng YOLO được sử dụng trong nhận dạng vật thể có tốc độ xử lý nổi bật trong nhóm các mô hình nhận dạng. Được đánh giá cao và lựa chọn ưu tiên trên mạch phát triển Jetson. Nhìn chung mô hình đã cho được kết quả rất đáng hi vọng trong quá trình nhận dạng của mình. Như đã đề cập cần phải cải thiện rất nhiều để có thể đưa vào ứng dụng thực tế.

Trong quá trình triển khai demo, em gặp rất nhiều khó khăn, cũng như thất bại trong việc cài đặt nền tảng hay thiết lập khởi động cho mô hình. Em đã thành công trong việc xây dựng mô hình nhận dạng sử dụng các mô hình huấn luyện trước. Có một điều rất hạn chế là chưa có thể tận dụng tốt được camera có sẵn trên board. Tuy nhiên do hạn hẹp về mặt thời gian nên em phải sử dụng camera webcam, em hi vọng có cơ hội giải quyết hạn chế này trong tương lai.

Tài liệu tham khảo

[1] Tanya Amert et al. "GPU scheduling on the NVIDIA TX2: Hidden details revealed". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2017, pp. 104–115.

[2] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[3] *FPT Cuộc đua số Cuộc đua số*. 2018. URL: https://cuocduaso.fpt.com.vn/en (visited on 03/21/2019).

[4] *FPT Tech Insight FPT đầu tư mạnh vào công nghệ Automotive*. 2018. URL: https://techinsight.com.vn/fpt-dau-tu-manh-vao-cong-nghe-automotive/ (visited on 03/21/2019).

[5] Dustin Franklin. "NVIDIA Developer Blog NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge". In: (2017). URL: https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/ (visited on 03/21/2019).

[6] Jim Harlow. *JetsonHacks Jetson RACECAR*. 2017. URL: https://www.jetsonhacks.com/category/robotics/jetson-racecar/ (visited on 03/21/2019).

[7] Jonathan Hui. *Medium Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. 2018. URL: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 (visited on 03/25/2019).

[8] *JetsonHacks Build Kernel and Modules – NVIDIA Jetson TX2*. URL: https://www.jetsonhacks.com/2017/03/25/build-kernel-and-modules-nvidia-jetson-tx2/ (visited on 04/04/2019).

[9] *Learn OpenCV*. URL: https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/ (visited on 04/07/2019).

[10] *Linux Programming: Makefile P1*. URL: http://eslinuxprogramming.blogspot.com/2015/04/gnu-make.html (visited on 04/05/2019).

[11] *Mathworks Deep Learning*. URL: https://www.mathworks.com/solutions/deep-learning/object-recognition.html (visited on 05/12/2019).

[12] *MIT Racecar Racecar*. 2017. URL: https://mit-racecar.github.io/ (visited on 03/21/2019).

[13] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. Vol. 821. John Wiley & Sons, 2012.

[14] Milind Naphade et al. "The nvidia ai city challenge". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE. 2017, pp. 1–6.

[15] *NVIDIA Autonomous Machine Jetpack*. URL: `https://developer.nvidia.com/embedded/jetpack` (visited on 03/30/2019).

[16] *NVIDIA Developer CUDA Zone*. URL: `https://developer.nvidia.com/cuda-zone` (visited on 03/21/2019).

[17] *NVIDIA Developer Tensor*. URL: `https://developer.nvidia.com/tensorrt` (visited on 03/30/2019).

[18] *NVIDIA Jetson Download Center*. URL: `https://developer.nvidia.com/embedded/downloads` (visited on 03/31/2019).

[19] *NVIDIA NVIDIA JETSON SOLUTIONS FOR DRONES UAVS*. URL: `https://www.nvidia.com/en-us/autonomous-machines/uavs-drones-technology/` (visited on 03/21/2019).

[20] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. URL: `https://pjreddie.com/darknet/` (visited on 04/04/2019).

[21] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[22] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[23] Kanokwan Rungsuptaweekoon, Vasaka Visoottiviseth, and Ryousei Takano. "Evaluating the power efficiency of deep learning inference on embedded GPU systems". In: *2017 2nd International Conference on Information Technology (INCIT)*. IEEE. 2017, pp. 1–5.

[24] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming, portable documents*. 2010.

[25] Zheng Tang et al. "CityFlow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification". In: *CVPR 2019: IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[26] *Ugenard LAB Error Sum of Squares (SSE)*. 2018. URL: https://hlab.stanford.edu/brian/error_sum_of_squares.html (visited on 03/25/2019).

[27] *vmware Workstation Pro*. URL: https://www.vmware.com/asean/products/workstation-pro.html (visited on 03/31/2019).

[28] *Wikipedia EGL (API)*. URL: https://en.wikipedia.org/wiki/EGL_(API) (visited on 03/31/2019).

[29] *Youtube: Tesla Autopilot in Heavy LA Traffic*. URL: https://www.youtube.com/watch?v=m3-QzTFxoUg (visited on 04/05/2019).