

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



Báo cáo đồ án chuyên ngành

**ĐỀ TÀI: XÂY DỰNG VÀ ĐÁNH GIÁ CÁC MÔ HÌNH ỨNG DỤNG
PHÁT HIỆN LỬA CHẠY TRÊN THIẾT BỊ JETSON NANO**

Giảng viên hướng dẫn: **TS. Lê Kim Hùng**

Lớp: **NT114.L11.MMCL**

Sinh viên thực hiện: **Vũ Hà Anh**

TP HCM, Ngày 12 tháng 01 năm 2020

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

MỤC LỤC

| | |
|---|-----------|
| PHỤ LỤC HÌNH ẢNH..... | 4 |
| PHỤ LỤC BẢNG..... | 4 |
| CHƯƠNG 1: TÓM TẮT ĐỀ TÀI..... | 5 |
| CHƯƠNG 2 TỔNG QUAN | 6 |
| 2.1. Đặt vấn đề..... | 6 |
| 2.2. Mô hình Yolo | 7 |
| 2.1.1. Giới thiệu về mạng Yolo..... | 7 |
| 2.1.2. Lịch sử phát triển Yolo | 7 |
| 2.3. Các mô hình nhận diện vật thể tương tự | 7 |
| 2.4. Các đề tài liên quan | 8 |
| CHƯƠNG 3 CƠ SỞ LÝ THUYẾT | 9 |
| 3.1 Mô hình Yolo | 9 |
| 3.1.1. Các thuật ngữ cơ bản..... | 9 |
| 3.1.2. Kiến trúc mạng..... | 10 |
| 3.1.3. Nguyên lý hoạt động..... | 11 |
| 3.1.4. Huấn luyện mô hình..... | 12 |
| 3.2 Tensorflow Lite | 13 |
| 3.2.1 Quy trình phát triển..... | 13 |
| CHƯƠNG 4: TRIỂN KHAI THỰC TẾ..... | 14 |
| 4.1 Chuẩn bị dữ liệu huấn luyện..... | 14 |
| 4.1.1. Gán nhãn dữ liệu | 15 |
| 4.2 Xây dựng mô hình..... | 16 |
| 4.2.1. Cấu hình chuẩn bị..... | 16 |
| 4.2.2. Huấn luyện mô hình | 18 |
| 4.2.3 Kiểm thử mô hình..... | 20 |
| 4.2.4 Tối ưu hóa mô hình bằng phương pháp Tensorflow Lite | 21 |
| 4.2.5 Triển khai trên thiết bị Jetson Nano | 21 |
| CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC..... | 22 |
| 5.1 Đánh giá hiệu năng giữa các mô hình | 22 |
| 5.2 Phương hướng phát triển đề tài | 23 |
| CHƯƠNG 6: TỔNG KẾT | 24 |
| TÀI LIỆU THAM KHẢO | 25 |

PHỤ LỤC HÌNH ẢNH

| | |
|--|----|
| Hình 1: Biểu đồ so sánh các mô hình | 8 |
| Hình 2: Kiến trúc mạng Yolo | 10 |
| Hình 3: Kiến trúc mạng Yolo | 11 |
| Hình 4: Mô tả Grid Size | 11 |
| Hình 5: Mô phỏng đầu ra của Yolo | 12 |
| Hình 6: Dữ liệu đưa vào | 14 |
| Hình 7: Định dạng .XML của nhãn hình ảnh | 15 |
| Hình 8: Định dạng .txt của nhãn hình ảnh | 15 |
| Hình 9: Giao diện phần mềm gán nhãn hình ảnh | 16 |
| Hình 10: Nội dung file yolo.names | 17 |
| Hình 11: Nội dung file yolo.data | 17 |
| Hình 12: Nội dung của file cấu hình | 18 |
| Hình 13: Định dạng file sau khi huấn luyện | 19 |
| Hình 14: Biểu đồ kết quả huấn luyện | 19 |
| Hình 15: Phát hiện lửa bằng Camera | 20 |
| Hình 16: Sử dụng Test Set | 20 |
| Hình 17: Định dạng .tflite | 21 |
| Hình 18: Đẩy ứng dụng lên Web Server | 22 |
| Hình 19: Phát hiện lửa trên ThingsBoard | 22 |

PHỤ LỤC BẢNG

| | |
|---|----|
| Bảng 1: Thông số được xây dựng trên Laptop | 23 |
| Bảng 2: Thông số được xây dựng trên Jetson Nano | 23 |

CHƯƠNG 1: TÓM TẮT ĐỀ TÀI

Tại đề tài này em tập trung vào các vấn đề chính như sau: chuẩn bị và gán nhãn các dữ liệu đưa vào, xây dựng mô hình phát hiện đám cháy, phát hiện vật thể trong thời gian thực qua Camera và đánh giá hiệu năng giữa các mô hình nhận diện vật thể. Các mô hình được sử dụng để phát hiện lửa trong đề tài này bao gồm: các phiên bản của Yolo (Yolov3, Yolov4, Yolov5), Darknet.Conv.29 chạy trên các nền tảng khác nhau, mô hình Tensorflow Lite được chuyển đổi từ Darknet.conv29. Tuy nhiên, các mô hình này chỉ thích hợp chạy trên các thiết bị có cấu hình cao với CPU và GPU mạnh, để đưa vào chạy trên các thiết bị cấu hình thấp ta cần phải qua các bước cấu hình và thiết lập khác nhau, các phương pháp sẽ được trình bày sau trong bản báo cáo này.

Thiết bị Jetson Nano phiên bản 2gb được em sử dụng để xây dựng các mô hình trong đồ án. Với bộ nhớ RAM chỉ có 2gb nên thiết bị luôn xảy ra tình trạng thiếu bộ nhớ trong quá trình phát hiện lửa. Để đảm bảo quá trình này được diễn ra thông suốt, em đã sử dụng một số cấu trúc rút gọn của các mô hình tuy nhiên vẫn đảm bảo độ chính xác ở trong mức ổn định khi đào tạo đưa dữ liệu vào. Đồng thời, việc chuẩn bị và gán nhãn dữ liệu cũng phải được đảm bảo cẩn thận, nhằm tránh các trường hợp mô hình nhận diện sai các vật thể. Ví dụ như khi thực thi mô hình phát hiện lửa, ánh đèn của bóng đèn điện thường bị nhận diện nhầm, cho nên việc lựa chọn hình ảnh cần lựa chọn kỹ càng, tránh đi những hình ảnh khiến mô hình bị hiểu nhầm trong khi tìm kiếm vật thể. Ngoài ra, giải pháp Tensorflow Lite cũng được em triển khai để tối ưu hóa mô hình nhằm đảm bảo độ ổn định của mô hình khi chạy trên thiết bị Jetson Nano.

Sau khi thực hiện xây dựng những mô hình được nói trên, kết luận được đưa ra là giải pháp sử dụng Tensorflow Lite là phương án thích hợp nhất trong việc phát triển ứng dụng phát hiện lửa trên thiết bị Jetson Nano nói riêng, cũng như các thiết bị có cấu hình thấp hơn so với Laptop và PC nói riêng. Mặc dù hệ điều hành Jetpack mà Jetson Nano đang sử dụng chưa hỗ trợ chạy trên GPU cho Tensorflow Lite, nhưng đây vẫn là giải pháp tối ưu nhất để chạy ứng dụng phát hiện lửa và dễ dàng tích hợp đưa ứng lên Web Server.

CHƯƠNG 2 TỔNG QUAN

2.1. Đặt vấn đề

Hiện nay để phát hiện lửa có rất nhiều phương pháp, ví dụ như: quan sát bằng mắt thường của con người, hệ thống vệ tinh, cảm biến không khí MQ-135, hệ thống xử lý hình ảnh,... Quan sát bằng mắt thường là một trong những phương pháp truyền thống, tuy nhiên không thiết thực khi có hỏa hoạn xảy ra. Hệ thống vệ tinh cần thời gian quét dài và không thể cung cấp hình ảnh đám cháy theo thời gian thực. Cảm biến MQ-135 mang lại sự nhầm lẫn giữa khói bụi từ môi trường bình thường và khói của đám cháy, hơn nữa việc phân biệt số lượng cảm biến lớn ngoài tự nhiên không phù hợp với tính chất tiết kiệm chi phí. Từ các phương pháp này ta có thể thấy, kỹ thuật xử lý hình ảnh đem lại ưu điểm vượt trội, chỉ cần một máy quay chất lượng cao, ta có thể giám sát một cánh rừng vài hecta, đồng thời phương pháp này còn cung cấp chi tiết về ba yếu tố của đám cháy là màu sắc, chuyển động và kết cấu.

Trong đề tài này, em đánh giá hiệu năng phát hiện lửa của ba phiên bản Yolov3, Yolov4, Yolov5, bản cải tiến của Yolov4-tiny có tên là Darknet.Conv.29 và mô hình Tensorflow Lite được chuyển đổi từ Darknet.Conv.29. Từ kết quả so sánh sau quá trình đào tạo mô hình, nhận thấy rằng phiên bản càng về sau, mô hình Yolo có độ chính xác càng cao, tuy Yolov5 mang đến hiệu quả cao nhất nhưng một số thư viện trong mô hình này không tương thích với hệ điều hành Jetpack của thiết bị Jetson Nano. Vì thế, mô hình Tensorflow Lite là một lựa chọn thích hợp, tuy không hỗ trợ GPU cho Tensorflow Lite, nhưng kết quả trong lúc thực hiện phát hiện lửa Tensorflow Lite vẫn đem lại kết quả ổn định nhất, có tốc độ khung hình mỗi giây giao động từ 8 đến 10. Từ kết quả này, em tin rằng, nếu như có thể cấu hình mô hình này theo một định dạng khác như TensorRT đang được Nvidia (công ty sản xuất Jetson Nano) hỗ trợ thì có thể đạt kết quả tốt hơn nhiều.

Yolo là mô hình phát hiện vật thể nổi tiếng với khả năng xử lý hình ảnh trong thời gian thực, nhưng đối với vật thể nhỏ, nó luôn có tỷ lệ phát hiện chính xác thấp hơn. Tuy nhiên ở đề tài này, em đã có khắc nhược điểm này bằng cách đưa hình ảnh ngọn lửa có kích thước nhỏ vào tệp huấn luyện, điều chỉnh các thông số về kích thước, độ lọc ảnh,... trong mô hình

Đối với thiết bị Jetson Nano bản 2gb em sử dụng trong đề tài, với bộ nhớ ít cho nên việc chạy các mô hình trong một thời gian dài là bất khả thi. Việc ra đời các mô hình rút gọn, được biết với cái tên là tiny mang lại hiệu quả cao, rút ngắn thời gian đào tạo và có khả năng chạy trên các thiết bị cấu hình thấp. Các thực nghiệm trên thiết bị Jetson Nano là các mô hình Yolov3-tiny, Yolov4-tiny, Darknet.Conv.29 và bản chuyển đổi từ Darknet.Conv.29 sang phiên bản Tensorflow Lite. Đồng thời, để có khả năng phát triển lên quy mô lớn hơn, em đã tích hợp Flask để có thể đưa chương trình lên Web Server và quản lý qua ThingsBoard (một chương trình hỗ trợ quản lý các thiết bị IoT).

2.2. Mô hình Yolo

2.1.1. Giới thiệu về mạng Yolo

Yolo là tên viết tắt từ cụm từ “You only look once”, tức là chúng ta chỉ cần nhìn 1 lần, điều đó chứng minh hiệu quả của Yolo trong bài toán phát hiện vật thể. Khác với bài toán Classification chỉ có thể dự đoán nhãn của vật thể. Yolo giải quyết bài toán Object Detection, không chỉ có thể phát hiện nhiều vật thể với nhiều nhãn khác nhau mà Yolo còn xác định vị trí của từng vật thể đó. Do đó, Yolo có thể phát hiện nhiều vật thể có nhãn khác nhau trong một khung ảnh, điều này rất phù hợp cho vấn đề xác định vật thể trong thời gian thực được đặt ra trong đề tài này.

2.1.2. Lịch sử phát triển Yolo

Phiên bản đầu tiên của Yolo là Yolov1 được ra mắt vào tháng 5 năm 2016 của tác giả Joseph Redmon với paper “You only look once: Unified, Real-Time Object Detection”. Nó là bước ngoặt cho việc phát triển của các bài toán phát hiện vật thể trong thời gian thực. Vào tháng 12 năm 2017 thì tác giả lại đưa ra một phiên bản khác mang tên Yolo9000.

Tháng 4 năm 2018, một phiên bản mới được phát triển với tên gọi Yolov3, đây là mô hình được sử dụng nhiều nhất cho đến thời điểm hiện tại, được hỗ trợ trên các nền tảng khác nhau: Pytorch, Darknet, Keras, Tensorflow Lite. Khác hẳn với hai phiên bản trước, ở phiên bản này, tốc độ huấn luyện mô hình được cải thiện đáng kể, dễ dàng cài đặt và cấu hình. Đặc biệt với nền tảng Darknet, người dùng có thể dễ dàng tạo ra một model riêng cho bản thân sau vài tiếng đồng hồ.

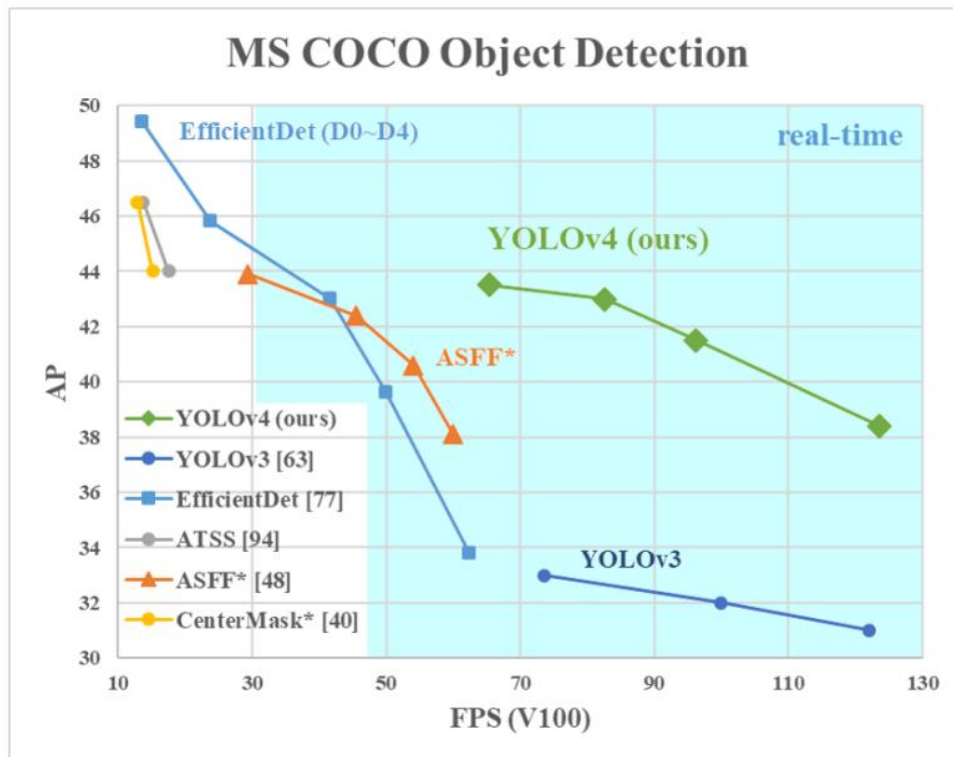
Vào đầu năm 2020, Alexey Bochkovskiy đã giới thiệu Yolov4, nó vượt trội hơn hẳn Yolov3 về độ chính xác trung bình, số khung hình trên một giây, tiết kiệm tài nguyên của thiết bị, thời gian đào tạo mô hình một cách đáng kể. Tính tới thời điểm hiện tại, tuy ra mắt chưa đầy một năm, nhưng số lượng các chủ đề về yolov4 luôn là một đề tài nóng trên các trang của ngành công nghệ thông tin.

Sau đó không lâu thì Glenn Jocher đã phát hành YOLOv5, có rất nhiều những tranh cãi xảy ra với cái tên Yolov5 này. Cho đến hiện nay, vẫn chưa có một Paper nào nói về vấn đề này. Phiên bản này hiện nay chỉ chạy được trên nền tảng Pytorch. Yolov5 là một mô hình khá gọn nhẹ, nhưng hiệu năng vẫn cao hơn các phiên bản trước.

2.3. Các mô hình nhận diện vật thể tương tự

Trong các thử nghiệm, YOLOv4 đã đạt được giá trị AP là 43,5% (65,7% của AP50) so với tập dữ liệu COCO của Microsoft và đạt được tốc độ thời gian thực gần 65 FPS trên Tesla V100, vượt trội so với các máy dò nhanh và chính xác cao trong các chi tiết về cả “tốc độ và độ chính xác”.

YOLOv4 nhanh gấp đôi EfficientDet bên cạnh hiệu quả tương ứng, so với YOLOv3, AP và FPS đã tăng lần lượt 10% và 12%. Bài báo mô tả đầy đủ về độ chính xác và tốc độ đặc biệt của YOLOv4 là những đóng góp tuyệt vời cho lĩnh vực khoa học.



Hình 1: Biểu đồ so sánh các mô hình

2.4. Các đề tài liên quan

Hiện nay có hai mô hình về ứng dụng phát hiện lửa được tham khảo nhiều nhất, đó là:

- Đề tài “**Fire Detection CNN**” của tác giả **Toby Breakon** sử dụng mạng CNN hay còn được biết đến với tên gọi “mạng nơ-ron tính chập”, đây là một mô hình sử dụng bài toán Classification. Đề tài này dựa vào bản Tensorflow 1.5, với cách cài đặt và triển khai dễ dàng, người dùng không tốn thời gian để đào tạo mô hình. Mô hình này chỉ có thể nhận diện trong khung ảnh là có lửa hay không có lửa và không thể xác định vị trí ngọn lửa trong khung ảnh.
- Đề tài “**Fire Net**” của tác giả **Olafenwa Moses** sử dụng mô hình Yolov3 trên nền tảng Keras, các tập tin sau khi được huấn luyện sẽ lưu dưới định dạng .h5. Mô hình được chạy bằng các câu lệnh trong thư viện Python là ImageAI cũng do tác giả tạo ra, điều này giúp người dùng chỉ cần chạy mô hình với 6 dòng code. Tuy tác giả nói rằng đây là một mô hình của Yolov3 nhưng tác giả đã thay đổi các thông số về anchor, dẫn tới việc chỉ

sử dụng thư viện ImageAi để chạy mô hình này và không thể chuyển đổi mô hình sang chạy trên các nền tảng khác như Pytorch, Darknet. Một nhược điểm nữa chính là tất cả câu lệnh đều dựa trên thư viện ImageAi, thư viện này không thể phát hiện vật thể trong thời gian thực.

CHƯƠNG 3 CƠ SỞ LÝ THUYẾT

3.1 Mô hình Yolo

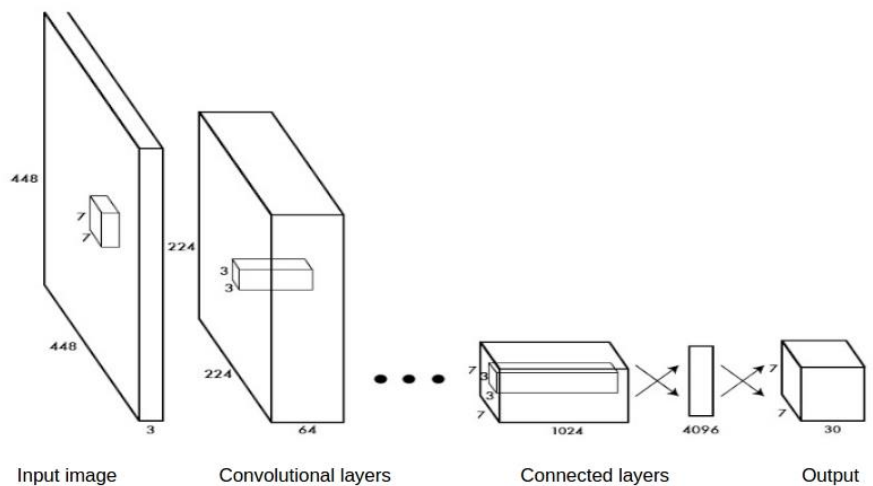
3.1.1. Các thuật ngữ cơ bản

- **Bounding Box** là khung hình bao quanh vật thể.
- **Anchor Box** là cơ sở để xác định vị trí vật thể, giúp bounding box có khả năng dịch tâm, tùy chỉnh kích thước và tạo khung hình bao quanh vật thể với kích thước chính xác nhất.
- **Feature map** là một khối output mà ta chia nó thành một mạng lưới với nhiều ô vuông và áp dụng tìm kiếm vật thể trên mỗi ô vuông.
- **Non-max suppression:** Phương pháp giúp giảm thiểu trường hợp nhiều Bounding Box xếp chồng lên nhau, tất cả đều được quy nạp về một Bounding Box có xác suất lớn nhất.
- **Training set** là một tập dữ liệu có kích thước lớn, được dùng để training trong quá trình huấn luyện máy học. Có thể hiểu đây chính là tập dữ liệu máy dùng để học và rút trích được những đặc điểm quan trọng và ghi nhớ lại. Tập training set sẽ gồm 2 phần:
 - Input: sẽ là những dữ liệu đầu vào. Ví dụ với bài toán nhận dạng hình ảnh: input sẽ là những bức hình.
 - Output: sẽ là những kết quả tương ứng với tập input. Ở đây chúng ta huấn luyện mô hình phát hiện lửa. Output sẽ là ngọn lửa.
- **Testing Set** là tập dữ liệu dùng để kiểm thử sau khi máy đã học xong. Một mô hình máy học **sau khi được huấn luyện**, sẽ cần phải được kiểm chứng xem nó có đạt hiệu quả không. Sau mỗi quá trình huấn luyện gian khổ, các mô hình này sẽ được kiểm chứng độ chính xác, để kiểm nghiệm được độ chính xác của mô hình này, người ta dùng tập Testing set. Khác với Training set, Testing set chỉ gồm các giá trị input mà không có các giá trị output. Máy tính sẽ nhận những giá trị input này, và xử lý các giá trị, sau đó đưa ra output tương ứng cho giá trị input.

- **Validation test** cũng giống như tập training set, nó cũng bao gồm các cặp giá trị input và output tương ứng. Điểm khác biệt ở đây là nó được sử dụng để kiểm thử độ chính xác của mô hình máy học **trong quá trình huấn luyện**. Sự khác biệt giữa Validation test và Testing set là: Testing được dùng để kiểm thử **sau** quá trình huấn luyện, còn Validation set được sử dụng để kiểm thử **trong** quá trình huấn luyện.
- **Class:** Mỗi vật thể cần phát hiện được gọi là một class.

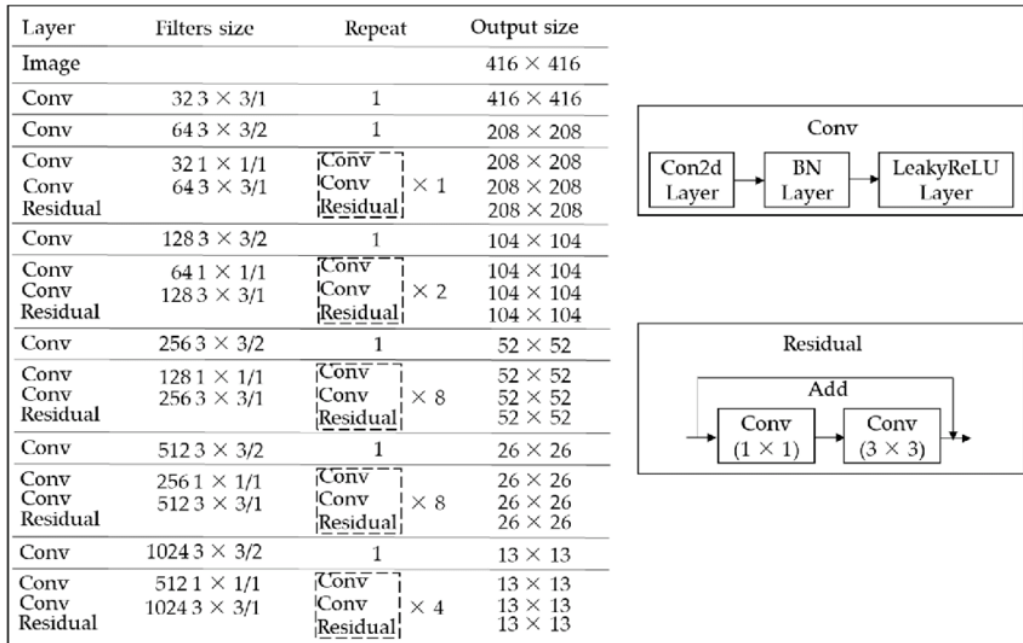
3.1.2. Kiến trúc mạng

YOLO là một deep net kết hợp giữa convolutional layers và connected layers. Hình ảnh đầu vào sẽ được chỉnh lại kích thước phù hợp để đưa vào Convolutional layers. Tại Connected layers là một Feature map có kích thước **7 x 7 x 1024** sẽ được sử dụng làm Input cho các lớp có tác dụng dự đoán nhãn và tọa độ Bounding Box của vật thể.



Hình 2: Kiến trúc mạng Yolo

Hình ảnh dưới đây là cách sắp xếp các lớp trong file .cfg của Yolo, mỗi phiên bản Yolo sẽ có cách sắp xếp khác nhau, để nâng cao độ chính xác khi nhận diện vật thể người ta thường thêm các lớp vào. Còn khi cần tăng tốc độ khung hình, tiết kiệm tài nguyên máy tính thì thường cắt bớt các layer (các bản yolo-tiny có ít layer hơn).



Hình 3: Kiến trúc mạng Yolo

3.1.3. Nguyên lý hoạt động

Khi một hình ảnh được đưa vào để phát hiện vật thể, Yolo phân chia hình ảnh thành một mạng lưới có kích thước **7 x 7** ô (hay còn gọi là grid size= 7 x 7).

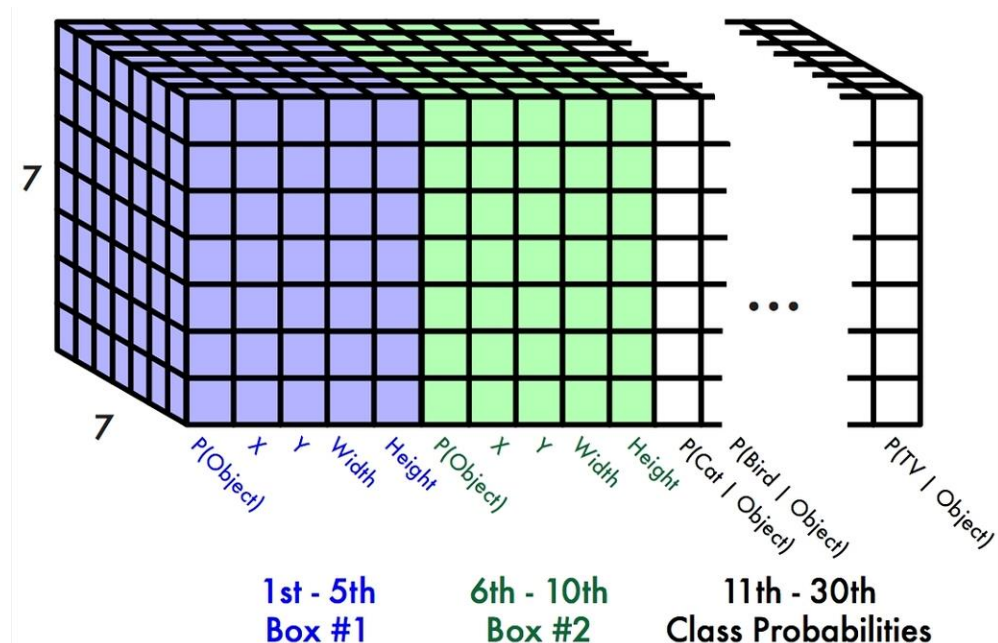


Hình 4: Mô tả Grid Size

YOLO sẽ dự đoán xem trong mỗi ô xem liệu có vật thể mà điểm trung tâm rơi vào ô đó. Và dự đoán điểm trung tâm, kích thước của object đó và xác xuất object đó là object nào trong các objects cần xác định.

Mỗi ô này có trách nhiệm dự đoán 2 hộp (boxes_number=2) bao quanh. Mỗi 1 hộp mô tả hình chữ nhật bao quanh một object.

Giả sử ta đang huấn luyện YOLO nhận dạng 20_objects khác nhau. Sau khi qua các layers, image input sẽ được biến đổi thành 1 tensor kích thước 7x7x30.



Hình 5: Mô phỏng đầu ra của Yolo

Có nghĩa là mỗi ô sẽ có 30 tham số, tham số thứ nhất là xác suất ô có chứa một vật thể, tham số 2,3,4,5 lần lượt là x center (vị trí trục hoành của tâm vật thể), y_center (vị trí trục tung của tâm vật thể), chiều rộng vật thể, chiều dài vật thể. Tương tự tham số 6,7,8,9,10 là của box thứ hai. Tham số thứ 11 là xác suất object trong ô là object thứ nhất (trong 20 objects cần nhận dạng). Tương tự tham số 12 là xác suất object trong ô là object thứ hai ... cho đến tham số 30 là xác suất object trong ô là object thứ 20.

Ta có thể hiểu đơn giản như sau:

1. Image input được resize thành 1 image 448x448x3 (image_dimension = 448x448 với số channels = 3, YOLO sử dụng hệ màu HSV)
2. Qua các layers, biến đổi image 448x448x3 thành 1 grid có kích thước 7x7 với số tham số cho mỗi ô trong grid là 30 ($30 = 5 \times \text{boxes_number} + \text{number_of_objects}$)
3. Neural net có nhiệm vụ huấn luyện các **trọng số của các layers** để có được mô hình tốt cuối cùng.
4. Để nhận dạng một image mới, các bước 1,2 sẽ được thực hiện. Sau đó dựa vào các **tham số trong grid 7x7x30** ta sẽ xác định được các box chứa object với xác suất cao (các box đè lên nhau sẽ được loại bằng phương pháp NMS, chỉ giữ lại box có xác suất cao nhất).

3.1.4. Huấn luyện mô hình

Neural net sẽ tính toán từng ảnh (có thể lặp lại 1 ảnh) để tối ưu hàm mất mát.

Việc tối ưu này sẽ giúp mạng neural tìm ra 1 bộ trọng số tốt nhất để biểu diễn dữ liệu của bạn và giúp nhận dạng các ảnh mới.

Sau khi mô hình được huấn luyện xong, các file lưu trữ kết quả sẽ có các định dạng .weights (darknet), .pt (pytorch), .tflite (Tensorflow Lite), .pb (Tensorflow), .h5 (Keras),...

Trong quá trình xác định lửa trong khung ảnh, mô hình được sử dụng sẽ lấy các thông số được huấn luyện từ trước, sau đó đem so sánh với kết quả rồi hiển thị thông số % có phải là lửa hay không.

3.2 Tensorflow Lite

TensorFlow Lite là một bộ công cụ giúp các nhà phát triển chạy mô hình TensorFlow trên các thiết bị di động, nhúng và IoT. Nó cho phép suy luận máy học trên thiết bị với độ trễ thấp và kích thước nhị phân nhỏ.

TensorFlow Lite bao gồm hai thành phần chính:

- Trình thông dịch TensorFlow Lite , chạy các mô hình được tối ưu hóa đặc biệt trên nhiều loại phần cứng khác nhau, bao gồm điện thoại di động, thiết bị Linux nhúng và bộ vi điều khiển.
- Bộ chuyển đổi TensorFlow Lite , chuyển đổi các mô hình TensorFlow thành một dạng hiệu quả để trình thông dịch sử dụng và có thể giới thiệu các tối ưu hóa để cải thiện kích thước và hiệu suất nhị phân.

Tensorflow Lite là một giải pháp tối ưu cho các thiết bị có cấu hình thấp, với ưu điểm tiết kiệm bộ nhớ, đưa ra dự đoán nhanh, tương thích với nhiều thiết bị khác nhau. Tuy nhiên Tensorflow Lite là một định dạng chỉ có thể sinh ra khi ta đã có tập tin lưu trữ dữ liệu đã được huấn luyện từ Tensorflow với định dạng .pb. Không có cách nào trực tiếp tạo ra Tensorflow Lite.

3.2.1 Quy trình phát triển

Quy trình sử dụng TensorFlow Lite bao gồm các bước sau:

1. Chọn một mô hình

Đầu tiên, phải chọn một mô hình muốn sử dụng để chuyển đổi qua định dạng .tflite. Mặc định để chuyển đổi, ta bắt buộc phải có thêm một mô hình Tensorflow Object Detection để thực hiện chuyển đổi.

2. Chuyển đổi mô hình

Nếu muốn chuyển đổi từ các nền tảng khác về Tensorflow Lite, ta bắt buộc phải thực hiện một bước trung gian đó là chuyển đổi về định dạng .pb của Tensorflow.

3. Triển khai trên thiết bị

Định dạng .tflite tương thích với nhiều thiết bị khác nhau, cho nên việc triển khai trên các thiết bị di động, nhúng,... khá dễ dàng với sự trợ giúp của nhiều tài liệu.

4. Tối ưu hóa mô hình

Phương pháp tối ưu hóa của mô hình TensorFlow Lite đã có sẵn trên trang chủ của Tensorflow. Tuy nhiên, sau khi tối ưu hóa, chất lượng mô hình sẽ giảm đi. Chính vì thế, nếu người dùng đã có một mô hình gọn nhẹ (dữ liệu đưa vào dưới 5000 ảnh) thì không nên sử dụng bộ công cụ này, tránh những ảnh hưởng xấu có thể xảy ra trong khi thực hiện phát hiện vật thể.

CHƯƠNG 4: TRIỂN KHAI THỰC TẾ

Để triển khai một mô hình ứng dụng phát hiện lửa trên thiết bị Jetson Nano ta cần thực hiện theo quy trình sau đây:

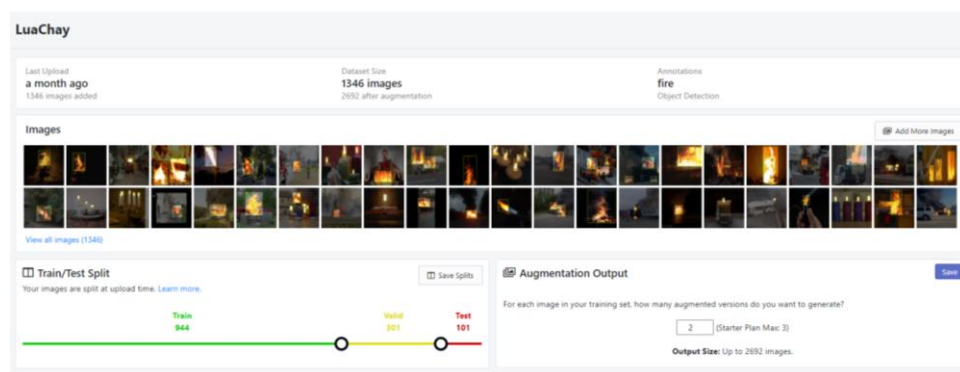
4.1 Chuẩn bị dữ liệu huấn luyện

Quy trình chuẩn bị dữ liệu để huấn luyện bước đầu tiên, cũng chính là bước quan trọng nhất để thực hiện quá trình xây dựng ứng dụng lửa. Dữ liệu được đưa vào phải thỏa các điều kiện:

- Hình ảnh phải khái quát, rõ ràng, tránh những hình ảnh bị vật thể khác che chắn.
- Bắt buộc phải gán nhãn và vẽ bounding box, để mô hình huấn luyện có thể xác định vật thể cần tìm.

Mô hình tối ưu nhất khi trong quá trình thực hiện huấn luyện, số lượng hình ảnh được tuân thủ theo tỉ lệ: 70% Training Set, 20% Validation Set, 10% Testing Set (không cần testing set model vẫn hoạt động bình thường)

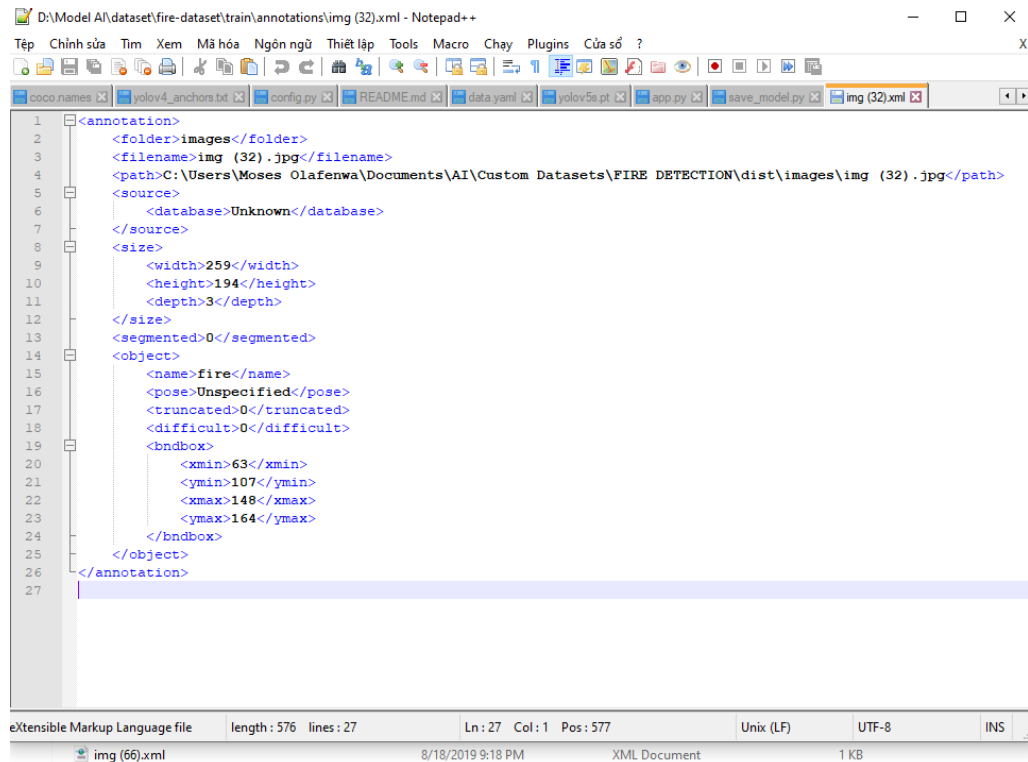
Trong đề tài này, em sử dụng 1346 ảnh trong việc huấn luyện.



Hình 6: Dữ liệu đưa vào

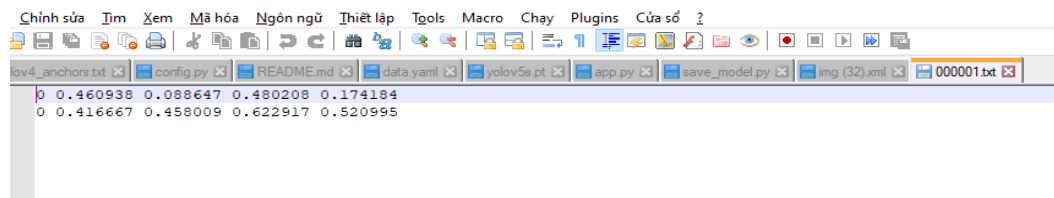
4.1.1. Gán nhãn dữ liệu

Mô hình chỉ hiểu thông tin được đưa vào khi hình ảnh đã gán nhãn. Nhãn hình ảnh cung cấp thông số theo thứ tự: thứ tự (hoặc tên) của class, tọa độ trục tung và hoành của tâm vật thể, chiều cao và chiều dài. Một tệp nhãn hình ảnh có định dạng:



```
1 <annotation>
2   <folder>images</folder>
3   <filename>img (32).jpg</filename>
4   <path>C:\Users\Moses Olafenwa\Documents\AI\Custom Datasets\FIRE DETECTION\dist\images\img (32).jpg</path>
5   <source>
6     <database>Unknown</database>
7   </source>
8   <size>
9     <width>259</width>
10    <height>194</height>
11    <depth>3</depth>
12  </size>
13  <segmented>0</segmented>
14  <object>
15    <name>fire</name>
16    <pose>Unspecified</pose>
17    <truncated>0</truncated>
18    <difficult>0</difficult>
19    <bndbox>
20      <xmin>63</xmin>
21      <ymin>107</ymin>
22      <xmax>148</xmax>
23      <ymax>164</ymax>
24    </bndbox>
25  </object>
26 </annotation>
27
```

Hình 7: Định dạng .XML của nhãn hình ảnh



```
0 0.460938 0.088647 0.480208 0.174184
0 0.416667 0.458009 0.622917 0.520995
```

Hình 8: Định dạng .txt của nhãn hình ảnh

Trong đề tài này em sử dụng định dạng .txt để triển khai mô hình và phần mềm LabelIMG để gán nhãn.



Hình 9: Giao diện phần mềm gán nhãn hình ảnh

4.2 Xây dựng mô hình

Mô hình được đào tạo dựa trên dữ liệu đưa vào từ bước trên, việc đào tạo mô hình sử dụng một lượng lớn tài nguyên của máy tính, tuy nhiên ta có thể huấn luyện trên Google Colab để đảm bảo hiệu quả và chất lượng của mô hình.

4.2.1. Cấu hình chuẩn bị

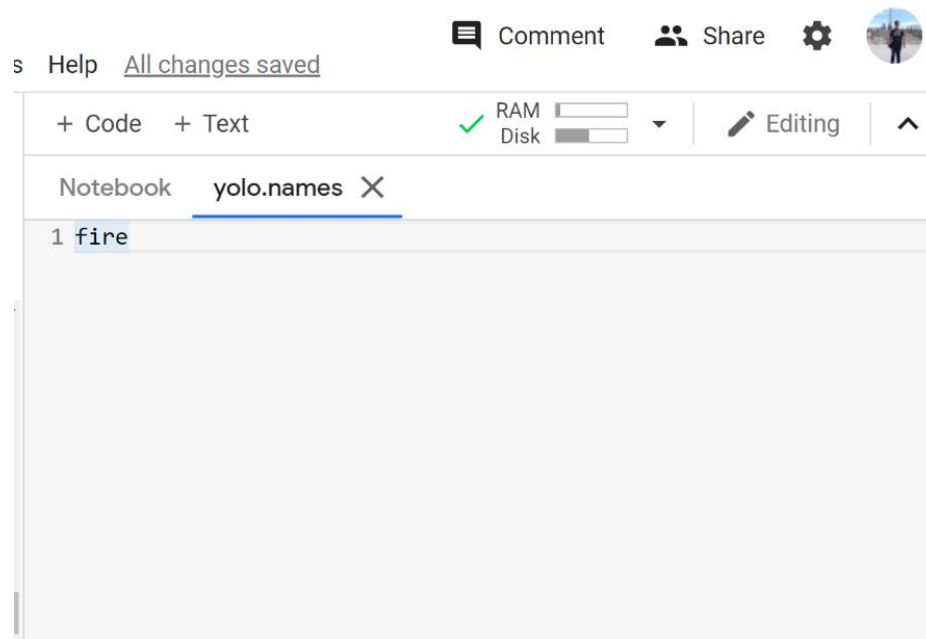
Bước 1: Tải nền tảng Darknet về, nền tảng Darknet cung cấp cho ta bộ mã nguồn có thể thực hiện huấn luyện các mô hình YOLOv3, YOLOv4, Resnet, SSD,...

➤ *! git clone <https://github.com/AlexeyAB/darknet.git>*

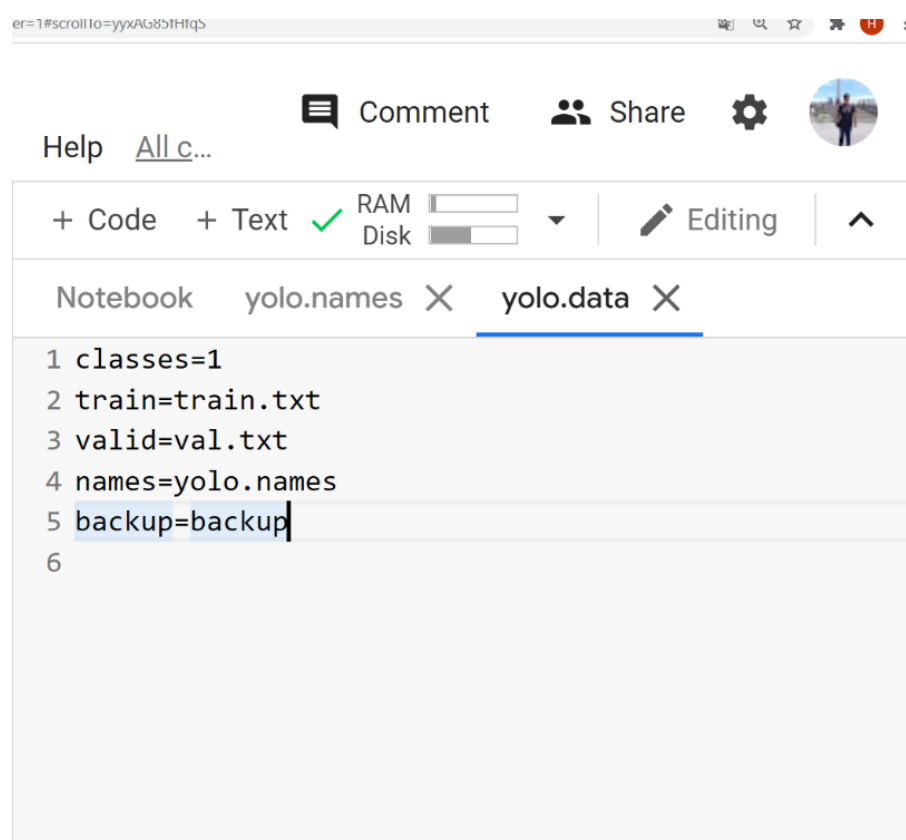
Bước 2: Tải mô hình huấn luyện hay còn được gọi là “Pretrain Model”, ở đây ta sử dụng model Darknet.Conv29, đây là một cải tiến của YOLOv4-tiny, một phiên bản khá gọn nhẹ, thích hợp sử dụng cho các thiết bị nhúng.

➤ *!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29*

Bước 3: Thiết lập hai files yolo.data và yolo.names. File yolo.names thiết lập tên đối tượng cần phát hiện, file yolo.data lưu trữ đường dẫn của dữ liệu, vị trí của tập .weights sau khi huấn luyện thành công.



Hình 10: Nội dung file yolo.names



Hình 11: Nội dung file yolo.data

4.2.2. Huấn luyện mô hình

Trong quá trình huấn luyện, ta cần điều chỉnh các tham số ở file .cfg. Đây là file config của phiên bản Darknet.Conv.29.

```
1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=64
7 subdivisions=16
8 width=416
9 height=416
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.00261
19 burn_in=1000
20 max_batches = 10000
21 policy=steps
22 steps=8000,9000
23 scales=.1,.1
24
25 [convolutional]
26 batch_normalize=1
27 filters=32
28 size=3
29 stride=2
30 pad=1
31 activation=leaky
32
33 [convolutional]
34 batch_normalize=1
35 filters=64
36 size=3
37 stride=2
38 pad=1
39 activation=leaky
```

Hình 12: Nội dung của file cấu hình

Các thông số cấu hình có ý nghĩa như sau:

- Tại dòng 6,7 sử dụng để xác định số lượng ảnh trong mỗi vòng train. Batch là số lượng ảnh trên mỗi iteration sẽ thực hiện huấn luyện, subdivision là tỉ lệ lấy ảnh trong mỗi Iteration. Có nghĩa là mỗi Iteration sẽ huấn luyện 64 ảnh, chia làm 4 lần lấy, mỗi lần 16 ảnh.
- Tại dòng 8,9 để định dạng kích thước ảnh.
- Tại dòng 17, filter = (số class + 5)*3.
- Tại dòng 18, learning_rate là tham số sử dụng trong việc huấn luyện các mạng Neural. Giá trị của nó là một số dương, thường nằm trong khoảng giữa 0 và 1. Tốc độ học kiểm soát tốc độ mô hình thay đổi các trọng số để phù hợp với bài toán. Tốc độ học lớn giúp mạng Neural được huấn luyện nhanh hơn nhưng cũng có thể làm giảm độ chính xác.
- Tại dòng 20, max_batches là số Iteration thực hiện tối đa, khác với Epoch trong quá trình đào tạo mô hình của Tensorflow, 1 Epoch hoàn

thành khi huấn luyện xong một vòng của toàn bộ các ảnh dữ liệu. 10000 Iteration tương đương xấp xỉ 500 epoch.

- Tại dòng 22, steps lần lượt bằng 80%, 90% của max_batches.

Sau khi thực hiện xong các bước trên, ta tiến hành huấn luyện mô hình bằng câu lệnh:

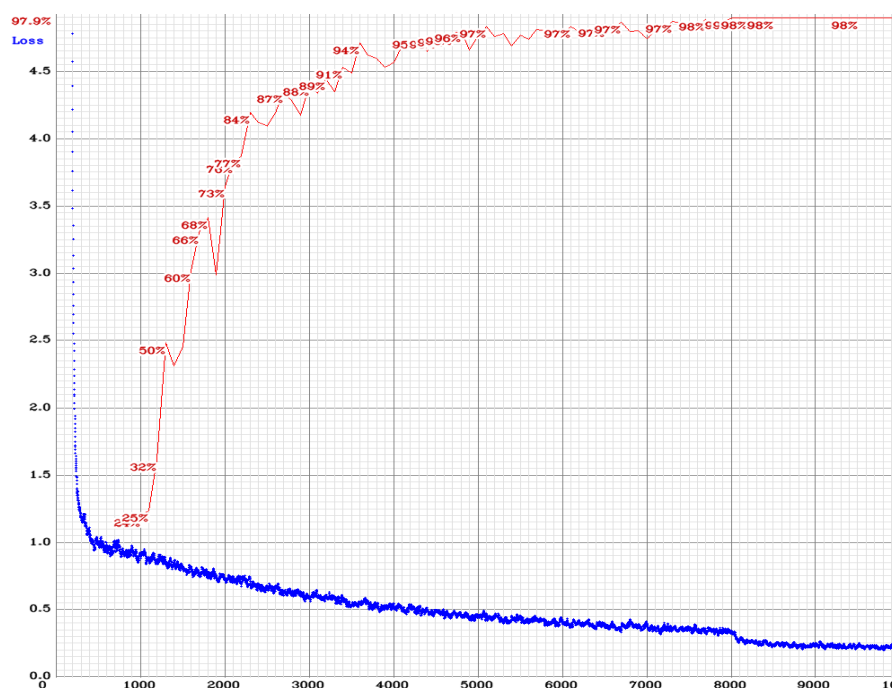
➤ `!./darknet detector train yolo.data data/tiny.cfg yolov4-tiny.weights`

Sau khi huấn luyện, ta đạt được một file có định dạng .weights nằm trong thư mục backup:

| | | | |
|---------------------------|---------------------|--------------|-----------|
| yolov4-tiny_best.weights | 12/20/2020 4:26 PM | WEIGHTS File | 22,971 KB |
| yolov4-tiny_final.weights | 12/11/2020 10:18 PM | WEIGHTS File | 22,971 KB |

Hình 13: Định dạng file sau khi huấn luyện

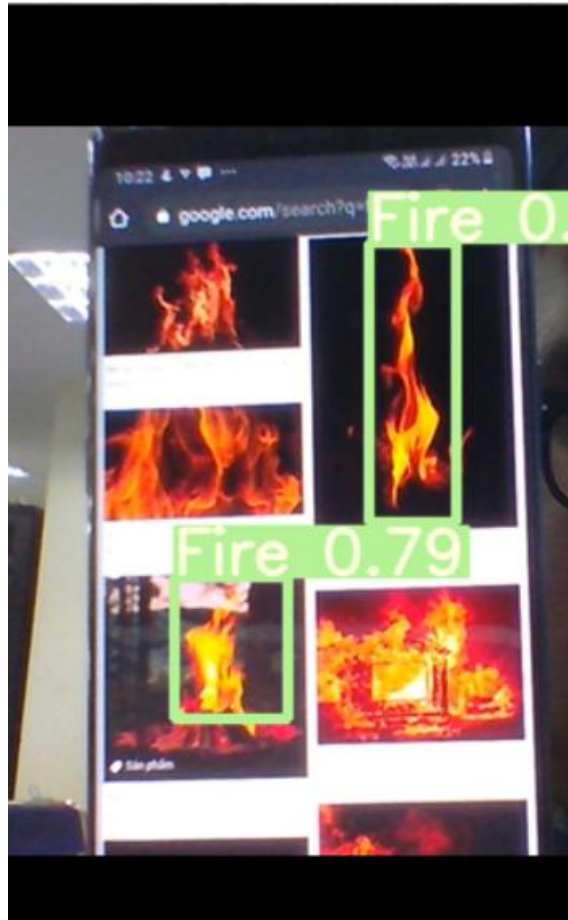
Với độ chính xác cao, đây là mô hình hoàn thiện để sử dụng xây dựng ứng dụng phát hiện lửa trên thiết bị Jetson Nano với độ chính xác cao nhất đạt 97.9%, tỉ lệ mất mát là 2.282%



Hình 14: Biểu đồ kết quả huấn luyện

4.2.3 Kiểm thử mô hình

Kết quả của mô hình vừa được huấn luyện sẽ được kiểm thử trước khi tối ưu hóa dưới dạng .tflite và nạp vào thiết bị Jetson nano. Với các kết quả dưới đây.



Hình 15: Phát hiện lửa bằng Camera



Hình 16: Sử dụng Test Set

Từ kết quả đạt được, mô hình này chạy hoàn toàn chính xác, với kết quả đáng tin cậy.

4.2.4 Tối ưu hóa mô hình bằng phương pháp Tensorflow Lite

Bước 1: Tải công cụ chuyển đổi tại đường dẫn

- Git clone <https://github.com/hunglc007/tensorflow-yolov4-tflite>

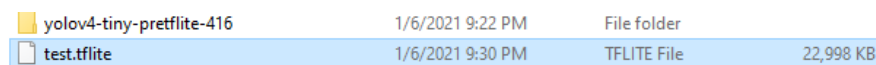

Bước 2: Lưu trữ mô hình dưới dạng .pb của Tensorflow với câu lệnh:

- `python save_model.py --weights ./data/yolov4-tiny.weights --output ./checkpoints/yolov4-tiny-416 --input_size 416 --model yolov4 --tiny`

Bước 3: Chuyển đổi từ mô hình Tensorflow về TFlite

- `python save_model.py --weights ./data/yolov4.weights --output ./checkpoints/yolov4-416 --input_size 416 --model yolov4 --framework tflite`

Kết quả thu được là một file có định dạng .tflite

| | | | |
|---|------------------|-------------|-----------|
|  yolov4-tiny-pretflite-416 | 1/6/2021 9:22 PM | File folder | |
|  test.tflite | 1/6/2021 9:30 PM | TFLITE File | 22,998 KB |

Hình 17: Định dạng .tflite

4.2.5 Triển khai trên thiết bị Jetson Nano

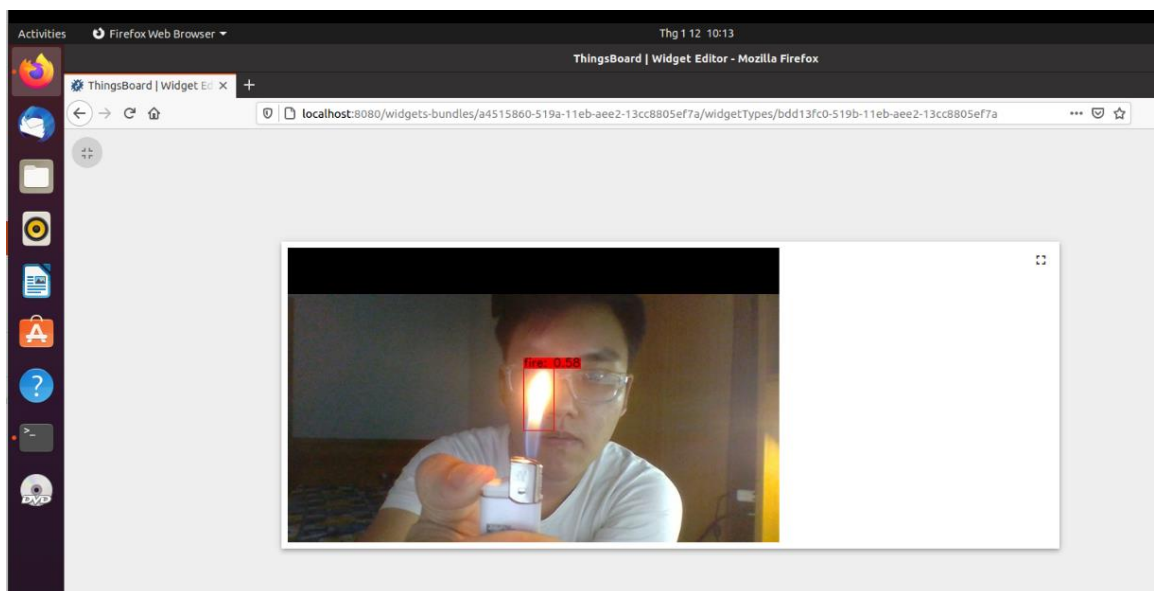
Trên thiết bị Jetson Nano, em đã triển khai mô hình TFlite kết hợp với Flask để có thể đưa được ứng dụng lên trên Web Server sau đó quản lý qua ThingsBoard, tuy có gây độ trễ và ảnh hưởng đến FPS, nhưng mô hình vẫn đáp ứng được các yêu cầu đề ra. Việc triển khai mô hình sẽ theo các bước sau:

Chạy ứng dụng phát hiện lửa trên thiết bị, ứng dụng sẽ được đẩy lên Web Server với địa chỉ: **<http://192.168.1.106:5000/>**

```
haanh@haanh-desktop: ~/tensorflow-yolov4-tflite
File Edit Tabs Help
h 27 MB memory) -> physical GPU (device: 0, name: NVIDIA Tegra X1, pci bus id: 0
000:00:00.0, compute capability: 5.3)
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployme
nt.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://192.168.1.106:5000/ (Press CTRL+C to quit)
* Restarting with stat
2021-01-12 10:07:55.630085: I tensorflow/stream_executor/platform/default/dso_lo
ader.cc:49] Successfully opened dynamic library libcudart.so.10.2
2021-01-12 10:08:01.842441: I tensorflow/stream_executor/platform/default/dso_lo
ader.cc:49] Successfully opened dynamic library libcuda.so.1
2021-01-12 10:08:01.855671: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-01-12 10:08:01.855841: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
742] Found device 0 with properties:
pciBusID: 0000:00:00.0 name: NVIDIA Tegra X1 computeCapability: 5.3
coreClock: 0.9216GHz coreCount: 1 deviceMemorySize: 1.92GiB deviceMemoryBandwidt
h: 194.55MiB/s
2021-01-12 10:08:01.855948: I tensorflow/stream_executor/platform/default/dso_lo
ader.cc:49] Successfully opened dynamic library libcudart.so.10.2
2021-01-12 10:08:01.860222: I tensorflow/stream_executor/platform/default/dso_lo
```

Hình 18: Đẩy ứng dụng lên Web Server

Sau khi ứng dụng phát hiện lửa đã được đưa lên Web Server, ta có thể từ bất cứ thiết bị nào trong mạng LAN truy cập. Ở đây, em đưa hình ảnh từ Camera, sau đó Stream lên chương trình ThingsBoard.



Hình 19: Phát hiện lửa trên ThingsBoard

CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC

5.1 Đánh giá hiệu năng giữa các mô hình

Để đưa ra sự lựa chọn chính xác giữa các mô hình, em đã thực hiện xây dựng nhiều mô hình, từ hiệu năng được biểu diễn, lựa chọn chính xác và cải thiện chất lượng sau đó tiến hành sử dụng trên thiết bị Jetson Nano.

Các mô hình ban đầu được xây dựng trên Laptop là YoloV3, YoloV4, YoloV5, với kết quả sau đây:

| | CPU | GPU | RAM | FPS | Độ chính xác | Độ mất mát |
|---------|--------|-----|-----------|------|--------------|------------|
| YoloV3 | 22.10% | 9% | 1961.5 MB | 11.5 | 80% | 3.16% |
| YoloV4 | 21.70% | 10% | 1961.5 MB | 12.5 | 88% | 2.90% |
| Yolo v5 | 5% | 15% | 1857 MB | 30 | 95% | 2% |

Bảng 1: Thông số được xây dựng trên Laptop

Mô hình được xây dựng trên Laptop có cấu hình:

- Memory: 16384MB RAM
- CPU: AMD Ryzen 5 4600H with Radeon Graphics (12 CPUs), ~3.0GHz
- GPU : NVIDIA GeForce GTX 1650 Ti

Từ đó đưa ra kết luận: Việc thực thi các mô hình này trên thiết bị Jetson Nano là không có khả năng. Do đó, ta phải áp dụng các phiên bản tiny trên thiết bị, sau khi thực hiện em đã đạt được kết quả như mong muốn với các thông số như sau:

| | CPU | GPU | RAM | FPS | Độ chính xác | Độ mất mát |
|--------------|-----|-----|--------|-----|--------------|------------|
| Yolov3-tiny | 96% | 84% | 1.41Gb | 6 | 77% | 1.50% |
| Yolov4-tiny | 88% | 64% | 1,65Gb | 6.5 | 84% | 2.20% |
| Darknet Conv | 85% | 79% | 1,52Gb | 7 | 98% | 2.50% |
| Tflite | 90% | 0% | 1,8Gb | 7.5 | 98% | 2.50% |

Bảng 2: Thông số được xây dựng trên Jetson Nano

5.2 Phương hướng phát triển đề tài

Để đề tài có thể triển khai và áp dụng trong thực tế, em có một số định hướng phát triển như sau:

- Xây dựng mô hình trên nền tảng TensorRT để có sự hỗ trợ của GPU trong quá trình thực hiện phát hiện lửa.
- Thiết lập hệ thống cảnh báo khi phát hiện đám cháy và gửi dữ liệu về ThingsBoard. Đồng thời nghiên cứu thêm về các cảm biến để tạo thành một hệ thống cảnh báo cháy sớm và hiệu quả nhất, giải quyết vấn đề hỏa hoạn trong xã hội ngày nay.

CHƯƠNG 6: TỔNG KẾT

Trong quá trình thực hiện đồ án chuyên ngành, em đã gặp một số khó khăn nhất định. Vì chưa có kinh nghiệm trong việc xử lý ngôn ngữ lập trình Python và kỹ thuật xử lý hình ảnh nên trong quá trình thực hiện đồ án diễn ra khá dài. Tuy nhiên, dưới sự giúp đỡ và hỗ trợ của giảng viên hướng dẫn, em đã có thể hoàn thành đề tài này đúng kì hạn và đạt được các yêu cầu đề ra. Đề tài này sẽ là bước đệm để em phát triển tốt hơn trong khóa luận tốt nghiệp.

Lời cuối cùng, em xin chân thành cảm ơn thầy Lê Kim Hùng trong học kỳ vừa qua đã hỗ trợ và hướng dẫn em để có thể hoàn thành đồ án chuyên ngành. Nhờ vào đề tài mà thầy giao em đã học hỏi thêm về nhiều kiến thức mới, đặc biệt là trong kỹ thuật xử lý hình ảnh, máy học, xây dựng ứng dụng trên thiết bị nhúng.

TÀI LIỆU THAM KHẢO

Tiếng anh:

1. <https://www.analyticssteps.com/blogs/introduction-yolov4>
2. <https://github.com/AlexeyAB/darknet>
3. <https://github.com/hunglc007/tensorflow-yolov4-tflite>
4. Paper Yolov3 Increment < <https://arxiv.org/pdf/1804.02767.pdf> >
5. Paper Yolov4 Optimal Speed < <https://arxiv.org/pdf/2004.10934.pdf> >

Tiếng việt

1. <https://viblo.asia/p/tim-hieu-ve-yolo-trong-bai-toan-real-time-object-detection-yMnKMdvr57P>
2. <https://forum.machinelearningcoban.com/t/object-detection-yolo/503>
3. <https://phamdinhkhanh.github.io/2020/03/09/DarknetAlgorithm.html>