ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

ỨNG DỤNG XỬ LÝ ẢNH SỐ VÀ VIDEO SỐ

Practice 02
YOLO – DARKNET

Giảng viên hướng dẫn Thầy Lý Quốc Ngọc
Thầy Nguyễn Mạnh Hùng
Thầy Phạm Minh Hoàng

Sinh viên thực hiện Võ Nguyễn Hoàng Kim
21127090

PHỤ LỤC

A. BẢNG TỰ ĐÁNH GIÁ

	Ghi chú	Mức độ hoàn thành
Thực nghiệm bằng mã nguồn mẫu được cung cấp	Mô hình: YOLOv4	100%
	Tập dữ liệu: ChessPieces	100%
	Thực nghiệm và báo cáo	100%

B. TỔNG QUAN

I. Mô hình YOLOv4

1. YOLOv4 là gì

- YOLO (You Only Look Once) là một thuật toán nhận diện đối tượng thời gian thực được phát triển bởi Joseph Redmon và Ali Farhadi vào năm 2015. Đó là một trình phát hiện đối tượng giai đoạn đơn sử dụng mạng nơ-ron tích chập (CNN) để dự đoán các hộp giới hạn và xác suất lớp của các đối tượng trong hình ảnh đầu vào. YOLO được triển khai lần đầu bằng cách sử dụng framework Darknet. Thuật toán YOLO chia hình ảnh đầu vào thành một lưới các ô, và cho mỗi ô, nó dự đoán xác suất có mặt của một đối tượng và tọa độ của hộp giới hạn của đối tượng. Nó cũng dự đoán lớp của đối tượng. Khác với các trình phát hiện đối tượng hai giai đoạn như R-CNN và các biến thể của nó, YOLO xử lý toàn bộ hình ảnh trong một lần đi qua, làm cho nó nhanh hơn và hiệu quả hơn [1].
- YOLOv4 là phiên bản thứ tư trong dòng sản phẩm mô hình You Only Look Once. YOLOv4 đặt ưu tiên vào việc nhận diện thời gian thực và thực hiện việc huấn luyện trên một GPU duy nhất. Ý định của các tác giả là để các kỹ sư thị giác và nhà phát triển có thể dễ dàng sử dụng framework YOLOv4 của họ trong các lĩnh vực tùy chỉnh [2].

2. Cấu trúc YOLOv4

- Về kiến trúc,tổng quan, YOLOv4 bao gồm 3 phần chính [3], cụ thể là:
    - Backbone: YOLOv4 thường sử dụng một deep neural network làm mạng gốc, thường dựa trên một mô hình được huấn luyện trước như CSPDarknet-53. Mạng backbone này trích xuất các đặc trưng cấp cao từ hình ảnh đầu vào.
    - Neck: Bước tiếp theo trong phát hiện đối tượng là trộn và kết hợp các tính năng được hình thành trong xương sống ConvNet để chuẩn bị cho bước phát hiện. YOLOv4 xem xét một số tùy chọn cho neck như FPN, PAN,… Cuối cùng, YOLOv4 chọn PANet để tổng hợp tính năng của mạng. Ngoài ra, YOLOv4 thêm khối SPP sau CSPDarknet53 để tăng trường tiếp nhận và tách các tính năng quan trọng nhất khỏi backbone [2].
    - Head: YOLOv4 triển khai head giống như YOLOv3 để phát hiện với các bước phát hiện dựa trên anchor và ba cấp độ chi tiết phát hiện [2].
- Ngoài ra, YOLOv4 còn sử dụng BoF và BoS để tăng tốc quá trình training và cải thiện độ chính xác của mô hình. Cả 2 phương pháp này được sử dụng cho phần Backbone và Detector [3].

II. Tập dữ liệu

1. ChessPieces

- Download: https://drive.google.com/file/d/18-AUodrP2NDTvPWC2NsDbhrZt9XO21OR/view
- Mô tả tập dữ liệu:
    - Đây là tập dữ liệu ảnh bàn cờ và các quân cờ khác nhau. Tất cả ảnh đều được chụp một góc không đổi. Trong tập dữ liệu này,

- Theo file **README.dataset.txt,** có tổng cộng 2894 nhãn trên 292 hình, tập dữ liệu bao gồm 12 lớp (classes), được ghi trong tập tin _classes.txt: “white-king”, “white-queen”, “white-bishop”, “white-knight”, “white-rook”, “white-pawn”, “black-king”, “black-queen”, “black-bishop”, “black-knight”, “black-rook”, “black-pawn”.
- Các tập tin như **_annotion.txt** hay **train.txt** và **val.txt** chứa dữ liệu đánh nhãn tọa độ và số thứ tự lớp (class) tương ứng của các đối tượng có trong từng tập tin ảnh.

C. THỰC NGHIỆM

I. Thiết lập Google Colab và môi trường

- Chuyển đổi Runtime sang chế độ GPU (Tesla T4).
- Cấp quyền truy cập vào Google Drive để truy xuất, lưu trữ mã nguồn, dữ liệu.
- Tải mã nguồn YOLOv4-PyTorch.
- Tải tập dữ liệu ChessPieces lên Google Drive.
- Cài đặt môi trường, các hàm thư viện cần thiết cho YOLOv4-PyTorch:
  - Thực hiện cài đặt các thư viện trong **requirements.txt**, tuy nhiên, do một số phiên bản của thư viện yêu cầu đã cũ, do đó, ta thay đổi và cài đặt các bản mới nhất của thư viện tương ứng.
- Chuẩn bị dữ liệu: cần chuẩn bị dữ liệu phù hợp theo chuẩn Yolo Dataset và đưa vào thư mục tương ứng trong mã nguồn:
  - Để thực hiện điều này, ta chạy lệnh (như trong hướng dẫn [4]) để tạo và sao chép nội dung (là các ảnh và các file txt) của tập dữ liệu ChessPieces vào tương ứng các thư mục train, data và test trong thư mục mã nguồn pytorch-YOLOv4.
  - Tải bộ trọng số **yolov4.conv.137.pth** (được cung cấp) vào thư mục pytorch-YOLOv4 để quá trình huấn luyện được hiệu quả hơn

II. Huấn luyện mô hình

1. Tinh chỉnh thông số

- Do ảnh được sử dụng trong tập dữ liệu có kích thước là 416×416, nên các giá trị cho biến *width height* trong mã nguồn đều được chuyển đổi thành 416.
- Thay đổi giá trị của các biến *classes* trong mã nguồn thành số lớp tương ứng mà ta có là 12.
- Thay đổi các giá trị của hệ số như *learning_rate, burn_in, batch* và *subdivisions* trong các file configure (cfg) thành các thông số phù hợp.
  - Ở đây có thể chỉ lưu ý đến 2 thông số chính là *Cfg.batch = 4* và *Cfg.subdivisions = 1* theo đúng như hướng dẫn được kèm theo của mã nguồn (theo file **Use_yolov4_to_train_your_own_data.md**).
- Đồng thời, do giới hạn của môi trường Google Colab (bản miễn phí), nên số lượng *epoch* trong mã nguồn đã giảm đi để tránh tình trạng interrupt đột ngột do đạt đến giới hạn của GPU. Số lượng epoch được sử dụng ở đây là 100.

2. Câu lệnh

- Sử dụng lệnh sau để thực hiện huấn luyện mô hình

```
!python train.py -l <learning-rate> -g <gpu-id> -pretrained <pretrained-backbone-networkt> -classes <number-of-classes> -dir <training-image-dir>
```

- Theo đó, lệnh được sử dụng trong bài lab này sẽ tương ứng là

```
!python train.py -l 0.001 -g 0 -pretrained yolov4.conv.137.pth -classes 12 -dir train
```

3. Quá trình huấn luyện

- Sau khi thực hiện gọi câu lệnh để huấn luyện mô hình, chương trình sẽ thực thi quá trình huấn luyện qua từng epoch. Do giá trị epoch được cài đặt trong mã nguồn là 100, do đó, mô hình sẽ thực hiện huấn luyện qua 100 chu kỳ. Mỗi lần hoàn thành một epoch, mô hình sẽ duyệt qua toàn bộ dữ liệu huấn luyện một lần và cập nhật các trọng số dựa trên độ lỗi giữa đầu ra và giá trị thực tế.
- Theo thực nghiệm, mỗi epoch cần trung bình khoảng 40 đến 60 giây cho cả việc huấn luyện và ghi trọng số tương ứng vào file.
- Với mỗi epoch, xong khi hoàn thành, kết quả nó trả ra sẽ là một file bộ trọng số có đuôi là **.pth** với tên tương ứng với epoch được hoàn thành, và được lưu trong thư mục **checkpoints** (là thư mục con trong pytorch-YOLOv4).
    - Ví dụ, với epoch25, sau khi hoàn thành huấn luyện, file kết quả mà nó trả ra sẽ là bộ trọng số, được đặt tên là **Yolov4_epoch25.pth**
    - Sau khi quá trình huấn luyện của mỗi epoch kết thúc, kết quả tích lũy sẽ được trả ra như sau (dưới đây là kết quả tích lũy sau khi thực hiện xong epoch 25):

```
Epoch 25/100:  99%|▉| 200/202 [00:40<00:00, 10.34iAccumulating evaluation results...
DONE (t=0.38s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.580
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.903
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.688
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.597
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.588
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.507
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.724
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.726
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.748
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.722
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
2024-03-29 05:57:14,504 train.py[line:446] INFO: Created checkpoint directory
2024-03-29 05:57:15,656 train.py[line:454] INFO: Checkpoint 25 saved !
Epoch 25/100:  99%|▉| 200/202 [00:43<00:00,  4.63i
```

- Bảng dưới đây được tổng hợp từ các kết quả tích lũy sau khi hoàn thành các epoch 1, 25, 50, 75 và 100, trong đó:
    - AP (Average Precision)
        - AP được sử dụng để đo lường độ chính xác của mô hình trong việc phát hiện và phân loại đối tượng. Nó thường được tính bằng cách so sánh các hộp giới hạn (bounding boxes) được dự đoán với các hộp giới hạn thực tế của đối tượng.
        - Giá trị AP càng cao, mô hình càng chính xác trong việc phát hiện đối tượng. Một AP tốt sẽ bao gồm cả việc đảm bảo độ chính xác (precision) cao và tỷ lệ bao phủ (recall) cao.
        - AP: là giá trị trung bình của độ chính xác tính dựa trên IoU từ 0.50 đến 0.95 cho tất cả các khu vực của đối tượng, với tối đa 100 đối tượng được dự đoán.
        - AP_{50}, AP_{75}: lần lượt là là giá trị AP được tính dựa trên IoU tại ngưỡng 0.50 và 0.75 cho tất cả các khu vực của đối tượng.
    - AR (Average Recall):
        - AR đo lường tỷ lệ bao phủ của mô hình, tức là tỷ lệ của số lượng đối tượng được dự đoán đúng so với tổng số lượng đối tượng có thể phát hiện được.
        - Giá trị AR càng cao, mô hình càng có khả năng phát hiện được nhiều đối tượng. Tuy nhiên, AR không chỉ ra độ chính xác của việc phát hiện, mà chỉ tập trung vào việc bao phủ các đối tượng.

- $AR^{max = 100}$: là giá trị AR được tính dựa trên IoU từ 0.50 đến 0.95 cho tất cả các khu vực của đối tượng, với tối đa 100 đối tượng được dự đoán.
- AR_S, AR_M: lần lượt là giá trị AR được tính cho các đối tượng có kích thước nhỏ (Small) và trung bình (Medium) với IoU từ 0.50 đến 0.95 và tối đa 100 đối tượng được dự đoán.

o Trong kết quả này, chúng ta không nhận xét đến các AP_L và AR_L do không có đối tượng nào trong phạm vi này được dự đoán. Điều này có thể là do không có đối tượng lớn nào trong dữ liệu hoặc mô hình không dự đoán chính xác đối tượng lớn. Ta nhận thấy điều này do các kết quả trả ra từ AP_L và AR_L đều là -1.000 trong quá trình huấn luyện mô hình.

	AP	**AP_{50}**	**AP_{75}**	**$AR^{max = 100}$**	**AR_S**	**AR_M**
Epoch 1	0.000	0.001	0.000	0.031	0.095	0.027
Epoch 25	0.580	0.903	0.688	0.726	0.748	0.722
Epoch 50	0.611	0.893	0.798	0.757	0.786	0.754
Epoch 75	0.602	0.879	0.732	0.757	0.811	0.754
Epoch 100	0.651	0.876	0.856	0.815	0.768	0.817

4. Nhận xét

- Có thể thấy, giá trị AP của mô hình tăng dần sau mỗi lần huấn luyện từ 0.000 ở epoch 1 đến 0.651 ở epoch 100, điều này cho thấy mô hình đã học được cách pháp hiện và phân loại đối tượng một cách chính xác hơn qua thời gian.
- Nhìn tổng quát, đối với IoU tại ngưỡng 50, giá trị của AP tăng mạnh trong khoảng đầu (từ epoch 1 đến 25) và có xu hướng giảm dần ở các epoch sau. Tuy giá trị giảm đi không đáng kể, nhưng nó có thể là dấu hiệu cho thấy tình trạng mà mô hình đã hoặc có thể gặp phải, chẳng hạn như: quá khớp (Overfitting), các vấn đề ở siêu tham số của mô hình khiến cho quá trình huấn luyện có sự biến động (mô hình học không ổn định), hoặc cũng có thể đến từ sự đa dạng của dữ liệu.
    - Tuy nhiên, trong phạm vi bài này, ta không rõ việc giảm giá trị này đến từ đâu nên không thể tinh chỉnh lại mã nguồn, nhưng đây là vấn đề cần lưu ý nếu tiếp tục triển khai mô hình này.
- Tuy nhiên, đối với ngưỡng 75, giá trị AP lại thể hiện việc tăng đều qua từng lần huấn luyện (dù giá trị tăng không đáng kể)

➔ Từ các giá trị AP có được, ta có thể nhận xét rằng mô hình đã có cải thiện đáng kể trong việc phát hiện các đối tượng với độ chính xác khá cao. Điều này cho thấy mô hình đã học tập rất tốt.

- Đối với các giá trị AR, nhìn chung, chúng đều có sự tăng dần qua từng lần thực hiện huấn luyện, dù các chỉ số chỉ dừng ở mức tương đối khoảng từ 0.7 đến 0.8

➔ Điều này cho thấy rằng mô hình cải thiện trong việc bao phủ các đối tượng và đặc biệt là trong việc phát hiện các đối tượng trung bình và nhỏ.

III. Dự đoán – Kiểm tra

1. Tinh chỉnh dữ liệu

- Sau khi huấn luyện mô hình hoàn tất, ta thực hiện sao chép file **_classes.txt** (có thể tìm thấy trong thư mục **train**) vào thư mục **data** và đổi đuôi file thành dạng **.names**. Khi đó, trong thư mục **data**, ta có được một file mới là **_classes.names** chứa các lớp mà chương trình dùng.

2. Câu lệnh

- Để thực hiện việc dự đoán, ta gọi lệnh theo cấu trúc sau:

!python models.py <number-of-classes> <weight-file> <image-file> <image-height> <image-width> <name-of-classes-file>

- Theo đó, trong mã nguồn này, câu lệnh sẽ được thực thi như sau:

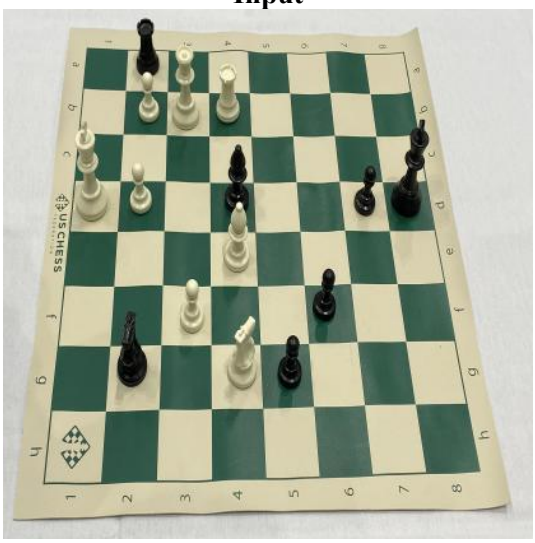
```
!python models.py 12 ./checkpoints/Yolov4_epoch100.pth ./test/
d7887071e972604ddf5940d8eb2702e7_jpg.rf.78288a14dc86417b6d3e798fa7fb5cf3.jpg 416 416
./data/_classes.names
```

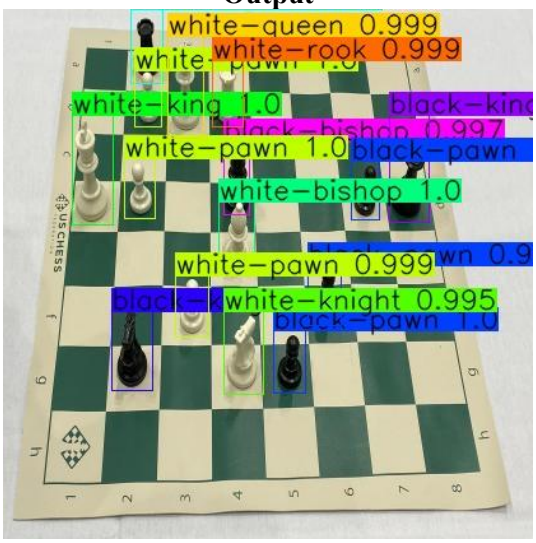
    - Trong phần này, ảnh được dùng để kiểm thử và dự đoán lấy từ thư mục **test**, tên ảnh được giữ nguyên theo đúng với bộ dữ liệu gốc.
    - File trọng số được sử dụng là kết quả trả ra của lần huấn luyện cuối cùng (tương ứng với kết quả của epoch 100), được lưu trữ trong thư mục **checkpoints**.
- Sau khi gọi lệnh để dự đoán, kết quả trả ra sẽ là một ảnh đã được đóng bounding box: xác định các lớp của từng đối tượng tương ứng trong ảnh và xác suất dự đoán tương ứng của đối tượng đó, tên ảnh sẽ là "**predictions.jpg**".
    - Tuy nhiên, do việc cài đặt mã nguồn, nên mỗi lần dữ đoán, chỉ dự đoán được một ảnh (theo tên truyền vào trong câu lệnh) và kết quả trả ra sẽ ghi đè lên ảnh dự đoán trước đó (nếu có). Điều này gây bất cập cho chúng ta khi muốn thu thập nhiều kết quả cùng lúc.

3. Kết quả

- Dưới đây là một số kết quả thu được từ việc huấn luyện mô hình và dự đoán:

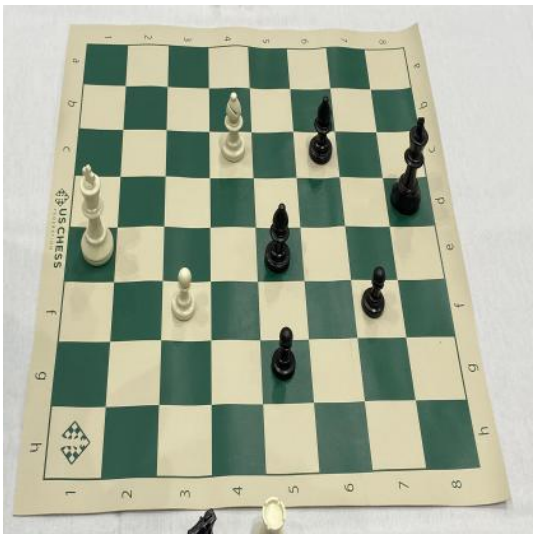
Input **Output**

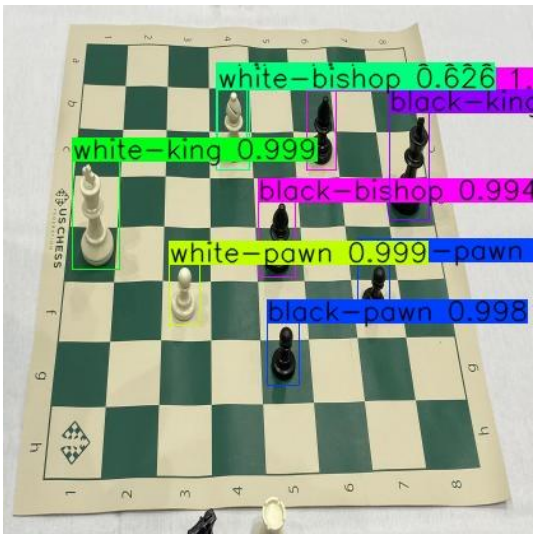




(1)

Input **Output**





(2)

- Với output (1), ta có được các thông số như sau cho kết quả dự đoán:

```
-----------------------------------
          Preprocess : 0.001689
     Model Inference : 0.451557
-----------------------------------
-----------------------------------
      max and argmax : 0.002160
                 nms : 0.001574
Post processing total : 0.003734
-----------------------------------
-----------------------------------
          Preprocess : 0.003207
     Model Inference : 0.044385
-----------------------------------
-----------------------------------
      max and argmax : 0.001393
                 nms : 0.000980
Post processing total : 0.002373
```

(a)

```
-----------------------------------
black-bishop: 0.997245
black-king: 0.994706
black-knight: 0.988767
black-pawn: 0.999861
black-pawn: 0.999711
black-pawn: 0.998816
black-rook: 0.992482
white-bishop: 0.999537
white-king: 0.999688
white-knight: 0.994656
white-pawn: 0.999724
white-pawn: 0.999547
white-pawn: 0.998762
white-queen: 0.998741
white-rook: 0.999364
save plot results to predictions.jpg
```

(b)

  - Trong đó, (a) biểu thị cho các số liệu, số liệu này cung cấp thông tin quan trọng về hiệu suất và tốc độ của quá trình dự đoán của mô hình phát hiện đối tượng, giúp ta hiểu rõ hơn về cách mà mô hình hoạt động và tối ưu hóa quy trình làm việc của nó. thông số tương ứng của kết quả sau khi mô hình dư đoán
  - Và (b) biểu thị cho xác suất được dự đoán cho việc phát hiện các quân cờ trong bức ảnh. Những số liệu này cung cấp thông tin về độ chắc chắn của mô hình trong việc dự đoán các quân cờ khác nhau trong bức ảnh kiểm định. Xác suất cao hơn thường đồng nghĩa với độ tự tin cao hơn của mô hình trong việc phát hiện đúng loại quân cờ tương ứng.

4. Nhận xét

- Có thể thấy, tui chỉ với chu kỳ huấn luyện là 100, nhưng kết quả mà mô hình học được cũng như dự đoán được là khá tốt. Nó thực hiện dự đoán gần như chính xác hoàn toàn đối với các đối tượng trong một bức ảnh.

➔ Điều này chứng tỏ hiệu suất mà mô hình đạt được đã rất tốt trong việc huấn luyện cũng như dự đoán.

D. TÀI LIỆU THAM KHẢO

[1] A. Mirkhan, "YOLO Algorithm: Real-Time Object Detection from A to Z," KILI TECHNOLOGY, [Online]. Available: https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z#what-is-yolo?.

[2] J. Solawetz, "What is YOLOv4? A Detailed Breakdown.," Roboflow, [Online]. Available: https://blog.roboflow.com/a-thorough-breakdown-of-yolov4..

[3] PIXTA VIETNAM, "YOLOv4 – Kỷ nguyên mới cho những mô hình họ YOLO," PIXTA VIETNAM, 09 March 2023. [Online]. Available: https://pixta.vn/yolov4-ky-nguyen-moi-cho-nhung-mo-hinh-ho-yolo.

[4] Bộ môn Thị giác máy tính và Điều khiển học thông minh, "HƯỚNG DẪN THỰC HÀNH #02: YOLO VÀ DARKNET".