Hand Detection For Grab-and-Go Groceries

1st Văn Sỹ
Khoa Công Nghệ Thông Tin.
Chuyên nghành Khoa Học Dữ Liệu
Bô môn Computer Vision
vansy151101@gmail.com
MSSV:19495751

2nd Lê Mỹ Thanh Lành
Khoa Công Nghệ Thông Tin.
Chuyên nghành Khoa Học Dữ Liệu
Bô môn Computer Vision
lmtlanhcva01@gmail.com
MSSV:19525231

***Tóm tắt nội dung*—Hệ thống nhận diện bàn tay (Hand detection system) là một thành phần quan trọng trong việc thực hiện tự động hoá hoàn toàn quy trình thanh toán cho các của hàng tạp hoá dạng Grab-and-Go. Dự án này thực hiện bằng cách sử dụng YOLO V2 và kiến trúc mới hơn của YOLO network, bộ dữ liệu được sử dụng là Egohand, training trên GPU Nvidia Tesla T4.**

I. Giới thiệu

Computer vision ngày càng được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau, trải nghiệm mua hàng và thanh toán ở Amazon Go là một ví dụ. Thay vì đợi xếp hàng để thanh toán, khách hàng chỉ cần lấy sản phẩm khỏi kệ, hệ thống tự nhận biết được sản phẩm đã lấy và tính tổng tiền, khách hàng thanh toán bằng thẻ hoặc cách hình thức thanh toán online khác. Quá trình tự động này chủ yếu dựa vào hệ thống thị giác máy tính có khả năng nhận biết các mặt hàng được lấy bởi một khách hàng nhất định.

Việc phát hiện bàn tay là một thành phần quan trọng trong một hệ thống như vậy. Bên cạnh độ chính xác, hệ thống cũng phải thực hiện việc nhận diện trong thời gian ngắn.Thuật toán phát hiện cũng phải có light-weight để mang đến việc tính toán nhẹ nhàng cho các hệ thống nhúng.

II. Một số phương pháp tiếp cận bài toán

Nhiều phương pháp Computer vision truyền thống đã được đề xuất để phát hiện bàn tay trong một hình ảnh. Dựa vào tần số quang phổ giống với bàn tay [1]. Một cách khác là sử dụng phân đoạn thông tin thông qua cấp pixel [2]. Tuy nhiên, các phương pháp này đều dựa vào việc phát hiện thông qua tông màu của da, vì vậy hệ thống có thể bị nhầm lẫn với khuôn mặt hoặc cách vùng da khác cũng có trong ảnh.

Gần đây, việc áp dụng Deep CNN đã cải thiện hiệu suất đáng kể của việc phân loại và phát hiện đối tượng. Việc phát hiện đối tượng gặp nhiều thách thức hơn liên quan đến việc đề xuất các Bounding-box cho các đối tượng tương ứng.

A. Region Proposal Based Detectors

Regions with CNNs (R-CNN) là một mạng phát hiện đối tượng dựa vào hệ thống đề xuất các khu vực có khả năng chứa đối tượng bên trong. Mặc dù nhanh và chính xác hơn đáng kể so với các phương pháp truyền thống, nhưng vẫn gặp các vấn đề về việc đề xuất các vùng khả năng. Để tìm các vùng đề xuất cho một ảnh duy nhất, sử dụng Selective search [3] mất 5s trên CPU 8 nhân. Fast R-CNN [4] sử dụng lại bản đồ đặc trưng từ đầu ra tích chập theo các vùng đề xuất cho lớp ROI pooling, so với R-CNN thì cách tiếp cận này nhanh hơn 25 lần. Tuy nhiên, Fast R-CNN vẫn phụ thuộc vào các vùng đề xuất, đây là vấn đề dẫn đến sự tắc nghẽn trong quá trình tính toán. Faster R-CNN ra đời đã giải quyết được vấn đề này, thời gian thực hiện nhanh hơn 10 lần so với Fast R-CNN.

Tất cả các phương pháp kể trên đều cho độ chính xác cao, nhưng chi phí tính toán và thời gian thực hiện không phù hợp cho một hệ thống nhúng.

Detection Framework	mAP	FPS
R-CNN O-Net BB	66.0	0.02
Fast R-CNN	70.0	0.5
Faster R-CNN VGG16	73.2	7
YOLO	63.4	45
SSD300	74.3	46
YOLOv2 480x480	77.8	59

Hình 1. Bảng so sánh hiệu suất và tốc độ giữa các mô hình được huấn luyện trên bộ dữ liệu PASCAL VOC 2007+2012 [5]

B. Single Shot Detectors

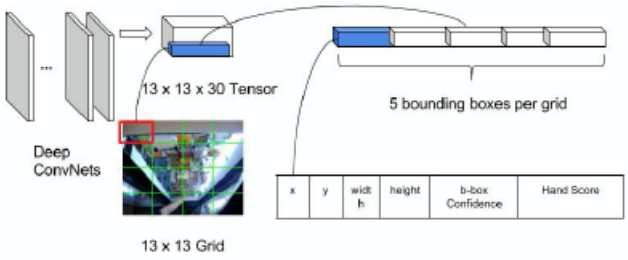
Là một cách tiếp cận khác của việc phát hiện đối tượng, nó gộp các thành phần riêng biệt của từng bước phát hiện đối tượng vào một mạng nơ-ron duy nhất, các thuật toán này chỉ cần thao tác với ảnh một lần và có thể đem lại tốc độ xử lý cao hơn so với họ mô hình R-CNN.

You Only Look One (YOLO) [6] là mạng có thể dự đoán các bounding-box và phân lớp các đối tượng cùng một lúc dựa trên activitions map đầu ra từ Deep ConvNets. YOLO9000 [5] là phiên bản nâng cấp của YOLO và có thể phát hiện ra 9000 lớp khác nhau, nó nhanh hơn và có thể đạt 73,4 mAP trên tập dữ liệu PASCAL [7], trong khi YOLO đạt 63.4 mAP. Single Shot MultiBox Detector (SSD) [8] đã xem xét các feature map khác nhau từ các lớp tích chập để dự đoán các bounding-box

với các tỉ lệ kích thước khác nhau. SSD phù hợp để sử dụng trong các trường hợp mà các bounding-box có tỉ lệ lớn.

III. Phương thức thực hiện

Đối với các phương pháp tiếp cận như R-CNN thì việc đề xuất 2000 vùng khả năng và bộ phân loại nhị phân để xem vùng nào chưa bàn tay thì điều này dẫn đến hệ thống có tốc độ cực kì chậm (mỗi ảnh mất 5s để đề xuất 2000 vùng khả năng chưa kể thời gian xử lý bộ phân loại nhị phân). Vì thế model được sử dụng trong bài toán này chính là YOLOv2. Chúng tôi sẽ mô tả nguyên tắc của model trong các phần sau.



Hình 2. Giải thích về YOLO workflow. S=13, B=5, C=1 cho nhận diện bàn tay.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Hình 3. Kiến trúc mạng Darknet-19.

IV. Sơ lược về YOLO

A. YOLOv2

1) Khái niệm: Yolov2 là bản cải tiến của Yolov1 nhóm tác giả thực hiện nhận ra YOLOv1 gặp vấn đề về localization khá nhiều (Bounding Box không tốt), hơn nữa, Recall của YOLOv1 cũng khá là thấp (Phát hiện được ít vật thể). Vì vậy, trong YOLOv2 tập trung vào cải thiện 2 vấn đề này mà vẫn giữ được độ chính xác tốt.

2) Kiến trúc mạng: Yolov2 hay còn gọi là Darknet 19 vì có 19 lớp Convolution (Conv).

a) Hàm loss:

$$\sum_{i=1}^{S^2}\sum_{j=1}^{B}\left(loss_{i,j}^{xywh} + loss_{i,j}^{p} + loss_{i,j}^{c}\right)$$

Trong đó:

- i = 1,...,S^2 là chỉ số của ô lưới
- j = 1,...,B là chỉ số của khe hộp neo

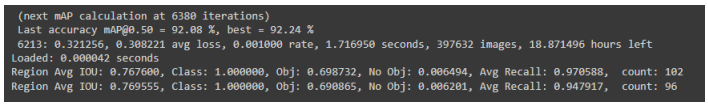
b) Hộp neo: Đầu ra của Yolov2 là 1 ma trận có shape:

$$Shape(A) = S * S * B * (4 + 1 + C)$$

- Với mỗi bbox ta predict ra: $t_x, t_y, t_w, t_h, t_{object}$ phân phối giai cấp $(p_1, p_2, ...p_c)$
- (t_x, t_y, t_w, t_h) không phải giá trị thực của bbox, mà chỉ là giá trị offset của bbox so với 1 anchor box cho trước. Anchor box này có kích thước (p_w, p_h) được định nghĩa sẵn

3) Điều chỉnh kiến trúc mạng Yolov2: Trước khi sử dụng Yolov2 thì phải cài Darknet [10] và tải xuống yolov2.weight và yolov2-tiny-voc.cfg. Sửa đổi hình ảnh đầu vào thành 608x608 cho phù hợp với bộ weight. Với việc nhận diện một lớp là hand và num = 5 thì cần phải thay đổi fiter ở lớp Convolution cuối cùng theo công thức filter = 5 * (num + class)

4) Kết quả: Sau khi tiến hành đào tạo mô hình khoản 6000 vòng thì thu được độ chính xác cuối cùng $mAP@0.50 = 92.08\%, best = 92.24\%$



Hình 4. Độ chính xác

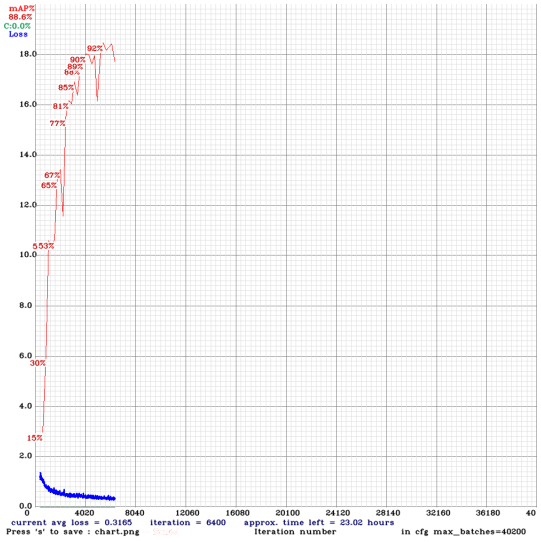
Từ đó ta có biểu đồ tổn thất và độ chính xác trong quá trình đào tạo, do việc train bằng máy tính cá nhân và sử dụng GPU của colab nên em chỉ chạy 6000 lần nhưng cũng được kết quả khá cao. Biểu đồ được thể hiện ở hình 5.

B. YOLOv3

1) Khái niệm: YOLOv3 là một bản nâng cấp đáng giá của V2, nó không những cải thiện hiệu suất mà còn giải thích những gì còn chưa rõ của YOLOv2.

2) Kiến trúc mạng:

a) Backbone: Sử dụng một backbone mới tên là Darknet-53(Hình 6).



Hình 5. Loss and mAP Yolov2

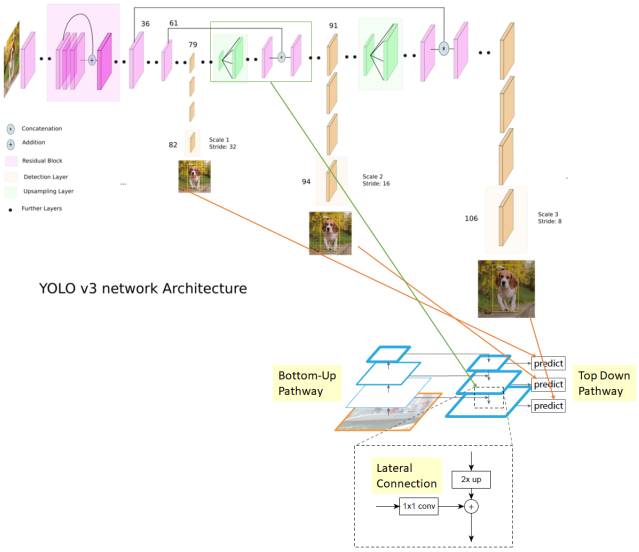
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Hình 6. Darknet-53

b) Neck: Từ các phiên bản trước, phát hiện vật thể nhỏ luôn là một điểm yếu. YOLOv3 đã giải quyết bằng cách áp dụng Feature Pyramid Network, thực hiện phát hiện đối tượng ở 3 scale khác nhau của feature map (hình 7).

3) Tổng kết: Những cải tiến bao gồm:

- Một backbone mới Darknet-53.
- Thêm Feature Pyramid Network, thực hiện dự đoán tại 3 scale.
- Hàm loss mới.



Hình 7. Kiến trúc của YOLOv3 với Feature Pyramid Network

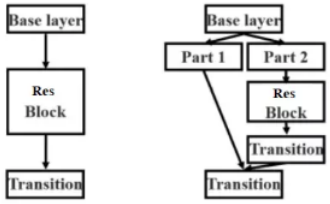
C. YOLOv4

1) Khái niệm: YOLOv4 là sự lắp ghép có chọn lọc của các nghiên cứu trong Object Detection lại với nhau.

2) Kiến trúc mạng: Về kiến trúc, YOLOv4 sử dụng

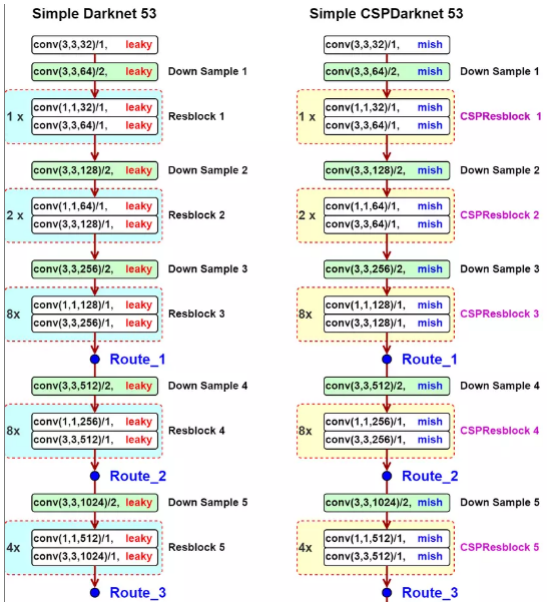
- Backbone: CSPDarkNet53
- Neck: SPP, PAN

a) Backbone: Ý tưởng chính của CSPBlock được thể hiện như Hình 8, áp dụng vào Residual Block. Thay vì chỉ có một đường đi từ đầu tới cuối, CSPBlock chia thành 2 đường đi. Nhờ việc chia làm 2 đường như vậy, ta sẽ loại bỏ được việc tính toán lại gradient (đạo hàm), nhờ đó có thể tăng tốc độ trong training. Hơn nữa, việc tách làm 2 nhánh, với mỗi nhánh là một phần được lấy từ feature map trước đó, nên số lượng tham số cũng giảm đi đáng kể, từ đó tăng tốc trong cả quá trình inference chứ không chỉ trong training.



Hình 8. Sự khác nhau giữ một Residual Block bình thường và CSPResBlock

YOLOv4 áp dụng ý tưởng của CSPBlock, thay thế Residual Block thông thường của YOLOv3 thành CSPResBlock, đồng thời đổi activation function từ LeakyReLU thành Mish, tạo nên CSPDarkNet53



Hình 9. DarkNet53 với CSPDarkNet53

b) Hàm mất mát:

$$LOSS = 1 - IoU + \frac{p^2(b, b^{g,t})}{c^2} + \alpha v -$$

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i log(C_i) + (1 - \hat{C}_i) log(1 - C_i) \right] -$$

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i log(C_i) + (1 - \hat{C}_i) log(1 - C_i) \right] -$$

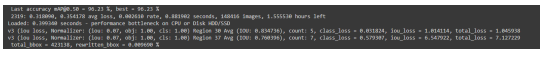
$$\sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in class} \left[\hat{p}_i(c) log(P_i(c)) + (1 - \hat{p}_i(c)) log(1 - P_i(c)) \right]$$

3) Điều chỉnh kiến trúc mạng Yolov4: Như ở v2 em tiếp tục sử dụng bộ weight được đào tạo trước với bộ dữ liệu Imagenet(608x608). Nên vẫn phải chỉnh đầu vào thành: 608x608.

Yolov4 có 2 lớp có cách tính filter khác so với bản Yolov2 $Filter = 3 * (num + class)$ (num trong bài là 5)

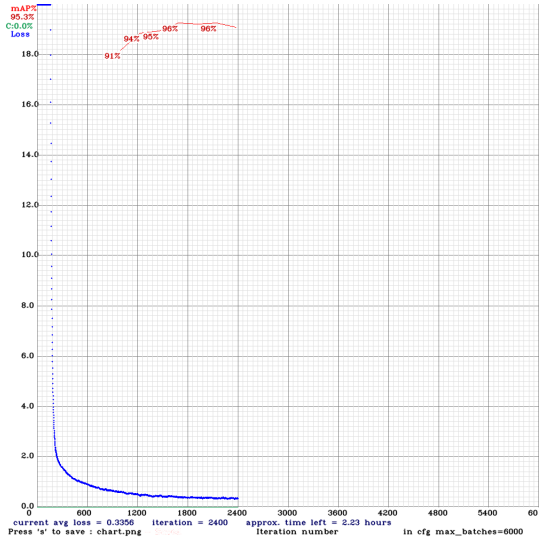
Lưu ý : Yolov4 có 2 lớp Yolo nên phải sửa filter và class 2 lớp Convolution

4) Kết quả: Với Yolov4 thì được đào tạo khoảng 2400 vòng và trả về độ chính xác cuối cùng là $mAP@0.50 = 95.32\%, best = 96.23\%$



Hình 10. Độ chính xác Yolov4

Bên cạnh đó là biểu đồ tổn thất và độ chính xác được đào tạo trong 2400 vòng được thể hiện ở hình 11



Hình 11. Loss and mAP Yolov4

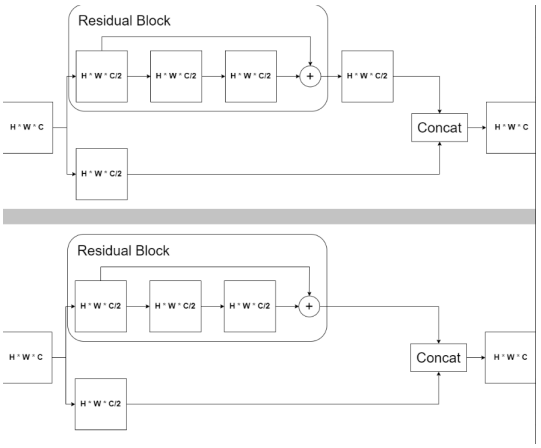
D. Yolov5

1) Yolov5 là gì: YOLOv5 không có quá nhiều thay đổi so với YOLOv4. Một số đóng góp từ tác giả của YOLOv5 (Glenn Jocher) đã trao đổi với tác giả của YOLOv4 và áp dụng luôn vào trong YOLOv4. YOLOv5 tập trung vào tốc độ và độ dễ sử dụng.

2) Kiến trúc mạng Yolov5: Yolov5 sử dụng

- Backbone: C3 module
- Neck: SPPF, PAN

a) Backbone: YOLOv5 cải tiến CSPResBlock của YOLOv4 thành một module mới, ít hơn một lớp Convolution gọi là C3 module. Chi tiết được thể hiện ở hình bên dưới.



Hình 12. Sự khác biệt giữa CSPResBlock trong YOLOv4 (trên) và C3 Module (dưới)

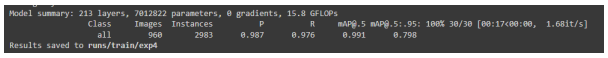
b) Hàm mất mát: YOLOv5 có sử dụng 3 đầu ra từ PAN Neck, để phát hiện objects tại 3 scale khác nhau. Tuy nhiên, Glenn Jocher nhận thấy rằng sự ảnh hưởng của các object tại

mỗi scale đến Objectness Loss là khác nhau, do đó, công thức của Objectness Loss được thay đổi như sau:

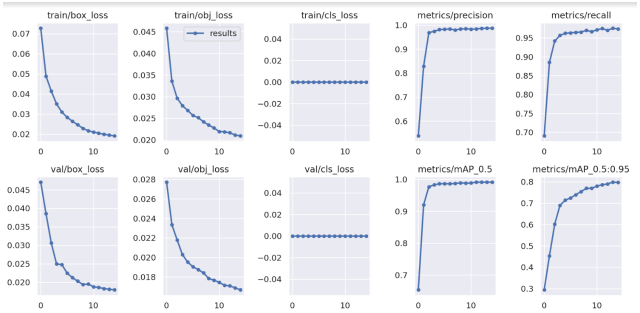
$$L_{obj} = 4.0L_{obj}^{small} + 1.0L_{obj}^{medium} + 4.0L_{obj}^{large}$$

3) Điều chỉnh kiến trúc mạng Yolov5: Yolov5 không cần chỉnh sửa gì

4) Kết quả: Yolov5 được em đào tạo 2 lần, lần đầu với 10 epoch và lần 2 với 15 epoch. Với learning rate lần lượt là 0.01 và 0.001.
Độ chính xác cuối cùng $mAP@.5 = 0.991\%, mAP@.5 : .95 = 0.798\%$



Hình 13. Độ chính xác Yolov5



Hình 14. Kết quả đào tạo Yolov5

V. Bộ dữ liệu

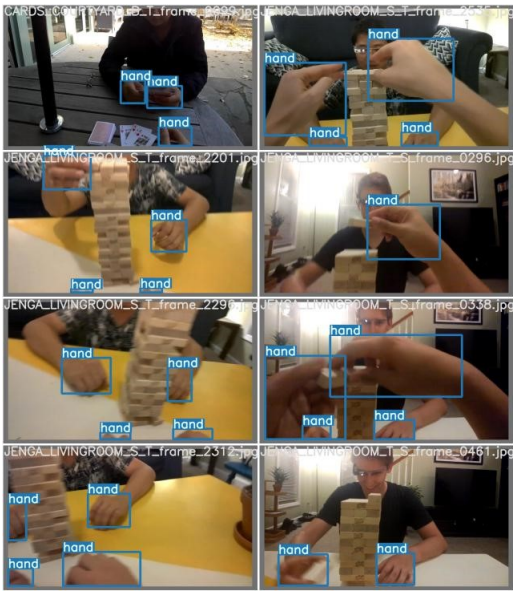
A. Egohand

Egohand [9] bao gồm 48 video từ Google Glass, mỗi video có 100 khung hình với độ phân giải1280×720 và mỗi khung hình này đều được định vị sẵn các ground truth bounding-box của bàn tay (hình 15). Đối với mỗi video, chọn ra 70 khung hình đầu để huấn luyện, 30 khung hình cuối để thử nghiệm. Tổng bộ dữ liệu đào tạo bao gồm 3360 ảnh, bộ dữ liệu thử nghiệm bao gồm 1440 ảnh.

B. Tiền xử lý dữ liệu

Bước tiền xử lý ảnh diễn ra trước khi cho dữ liệu vào mạng, đầu tiên sẽ thay đổi kích thước đầu vào về đúng với kích thước mà phiên bản YOLO yêu cầu. Sau đó ảnh được biến đổi một cách ngẫu nhiên để tăng thêm dữ liệu đào tạo(Scaling, translation, flipping, recolor) và toạ độ ground truth bounding-box được cập nhật theo tương ứng(Hình 16).

Các ground truth bounging-box được chúng tôi trích xuất bằng Python và chuyển sang dạng format yêu cầu của YOLO là Class, center X, center Y, width, heigh.



Hình 15. Một số mẫu của bộ dữ liệu Egohand và ground truth bounding-box của chúng

Hình 16. Một số ví dụ cho việc chuyển đổi để tăng dữ liệu huấn luyện.

VI. Kết quả

A. Huấn luyện

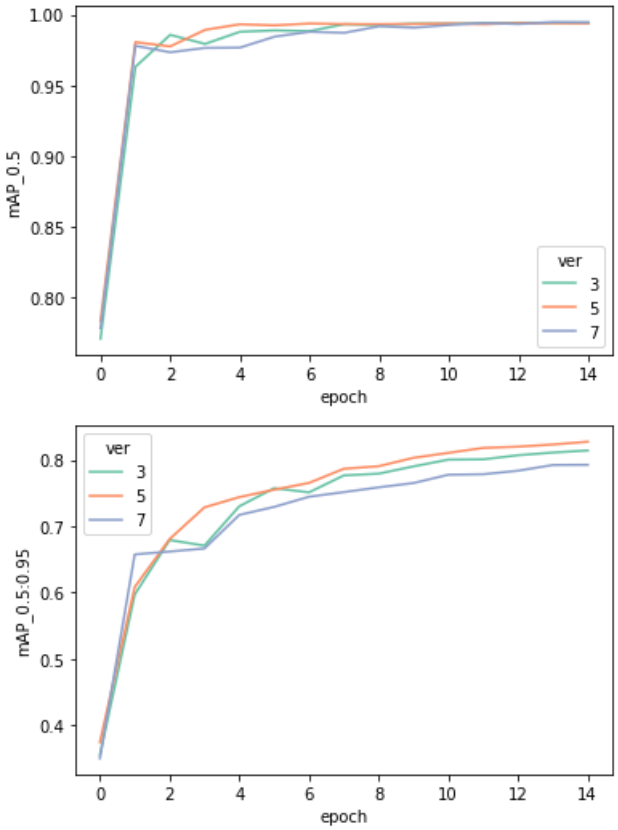
Việc huấn luyện dựa trên mã nguồn mở của các phiên bản YOLO với free GPU Google Colab, sử dụng pre-trained weights trên tập PASCAL COCO, batch size 16, epoch 15. Kết quả tổng hợp ở hình 17

B. Nhận diện thời gian thực trên GPU Tesla T4

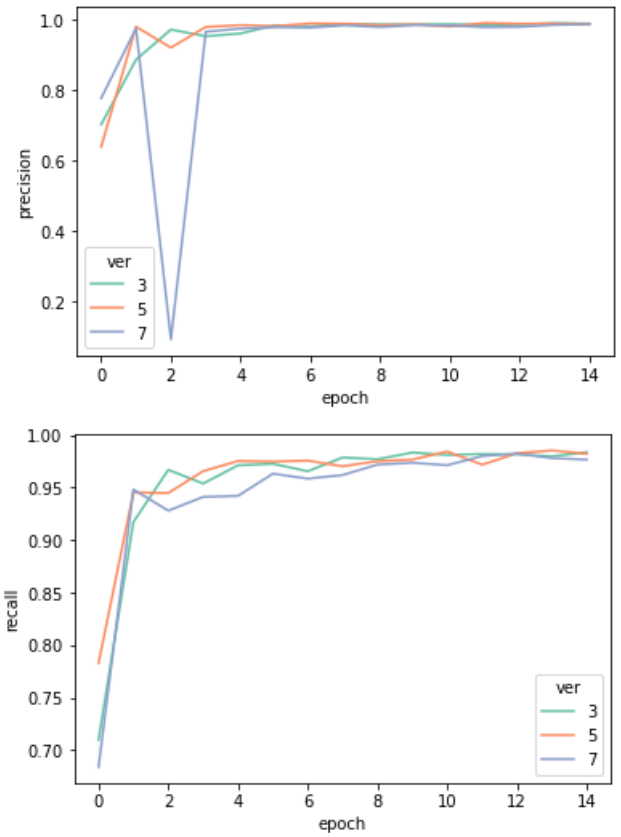
Tận dụng GPU miễn phí của Google Colab, triển khai nhanh nhận diện tay thời gian thực đạt kết quả 30fps (Hình 20)

	YOLOv3	YOLOv5	YOLOv7
Batch size	16	16	16
Epoch	15	15	15
Final Box_loss	0.016091	0.015456	0.01845
Final Object_loss	0.018112	0.01727	0.007296

Hình 17. Kết quả quá trình training



Hình 18. mAP với IOU 0.5 và IOU 0.5-0.95



Hình 19. Precision và Recall

Hình 20. Nhận diện thời gian thực với GPU của Google Colab

VII. Kết luận và dự định tiếp theo

Kết quả huấn luyện và đánh giá trên tập Egohand tốt, nhưng khi triển khai thì còn nhiều trường hợp chưa nhận diện đúng, cần cải thiện chất lượng dữ liệu, tìm cách tối ưu hoá quá trình huấn luyện. Với dữ liệu cần cải thiện về góc nhìn (vì góc camera nhận diện sẽ là góc thứ nhất), để tránh tình trạng khách hàng lấy sản phẩm nhưng không nhận diện được tay thì cần cải tiến thêm nhận diện cánh tay để hỗ trợ, mở rộng thêm nhiều class để quá trình xử lý thông tin cho kết quả được chi tiết hơn(ví dụ như các lớp về sản phẩm, hành vi khách hàng,...).

References

Tài liệu

[1] M. Klsch and M. Turk. Robust hand detection., 2004.

[2] Betancourt, Lpez, Regazzoni, and Rauterberg. A sequential classifier for hand detection in the framework of egocentric vision., 2014.

[3] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A.W. M. Smeulders. Selective search for object recognition. International Journal of Computer Vision, 104(2):154–171, 2013.
[4] R. Girshick. Fast r-cnn. In International Conference on Computer Vision (ICCV), 2015.
[5] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242, 2016.
[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. ArXiv e-prints, June 2015.
[7] M. Everingham, L. Van Gool, C. K. I.Williams, J.Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2):303– 338, June 2010.
[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. ArXiv e-prints, Dec. 2015.
[9] S. Bambach, S. Lee, D. J. Crandall, and C. Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In The IEEE International Conference on Computer Vision (ICCV), December 2015.
[10] https://github.com/AlexeyAB/darknet.git