

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ



BÁO CÁO
BÀI TẬP LỚN
HỌC KỲ 241 NĂM HỌC 2024-2025

**CLASSIFICATION OF DENTED CANS AND LIDS BY
DEGREE USING COMPUTER VISION**

Ngành: Kỹ thuật Điều khiển và Tự động hóa

Giảng viên hướng dẫn: PhD. PHẠM VIỆT CƯỜNG

Sinh viên thực hiện

—o0o—

Nguyễn Đình Tuấn Minh - L02 - 2212055

Phùng Khải Minh - L01 - 2212078

Nguyễn Song Minh Tâm - L01 - 2112240

Dương Quốc Thuận - L01 - 1951210

Vũ Tiến Nam - L01 - 2114137

TP. Hồ Chí Minh, tháng 12 năm 2024

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin được bày tỏ lòng biết ơn sâu sắc tới quý thầy cô Trường đại học Bách Khoa – Đại học Quốc gia Thành phố Hồ Chí Minh nói chung, quý thầy cô khoa Điện – Điện tử, quý thầy cô chuyên ngành Kỹ thuật Điều khiển và Tự động hóa nói riêng, đã truyền đạt cho chúng em những kiến thức bổ ích để phục vụ cho quá trình hoàn thành project về Classification Of Dented Cans And Lids By Degree Using Computer Vision.

Chúng em cũng xin gửi lời cảm ơn đặc biệt đến thầy Phạm Việt Cường. Chúng em rất ân tượng với thầy bởi sự tận tâm, nhiệt huyết và những kiến thức chuyên sâu mà thầy cung cấp trong quá trình giảng dạy. Sự hỗ trợ của thầy về mặt định hướng đã giúp nhóm rất nhiều, để chúng em có thể hoàn thành được project này.

Em xin chân thành cảm ơn!

TÓM TẮT

Nghiên cứu này trình bày một hệ thống phân loại tự động lon và nắp lon bị móp theo mức độ biến dạng dựa trên công nghệ thị giác máy tính. Hệ thống được thiết kế để nhận diện, phân loại các lỗi chất lượng của sản phẩm trong quy trình sản xuất và đóng gói, góp phần nâng cao hiệu quả kiểm tra tự động.

Quy trình thực hiện bao gồm:

- Phát hiện và cắt ảnh đối tượng: Sử dụng mô hình YOLO để nhận diện và cắt ảnh lon hoặc nắp lon từ hình ảnh đầu vào, đảm bảo độ chính xác trong việc phát hiện vị trí.
- Phân loại mức độ biến dạng: Sử dụng các mô hình học sâu (deep learning) như Xception để phân loại các đối tượng vào bốn nhóm chính: lỗi nghiêm trọng, lỗi lớn, lỗi nhỏ, hoặc không có lỗi.
- Huấn luyện và đánh giá mô hình: Áp dụng các phương pháp tiền xử lý dữ liệu, cân bằng lớp (class balancing) và tối ưu hóa mô hình để cải thiện độ chính xác và tính ổn định.

Kết quả cho thấy hệ thống đạt hiệu suất cao trong việc phát hiện và phân loại, phù hợp với yêu cầu sản xuất công nghiệp. Hệ thống này không chỉ giảm sự phụ thuộc vào kiểm tra thủ công, tăng hiệu quả quy trình mà còn đảm bảo chất lượng sản phẩm đồng nhất và giảm thiểu rủi ro từ sản phẩm lỗi.

Nghiên cứu mở ra tiềm năng ứng dụng lớn trong ngành công nghiệp đóng gói, sản xuất, nơi yêu cầu kiểm soát chất lượng tự động và chính xác.

PHÂN CÔNG CÔNG VIỆC

Tên	Lớp	MSSV	Nội dung công việc	Mức độ hoàn thành
Nguyễn Đình Tuấn Minh	L02	2212055	Xception, YOLOv8, Realtime, Giải pháp, HW15, 16	100%
Phùng Khải Minh	L01	2212078	Cơ sở lý thuyết, Ưu nhược điểm, Tổng hợp, HW12,13	100%
Nguyễn Song Minh Tâm	L01	2112240	GUI, HW14, 19	100%
Dương Quốc Thuận	L01	1951210	HW4, 10, 11	100%
Vũ Tiến Nam	L01	2114137	DATASET, HW5, 7, HW17, 18	100%

Mục lục

1	Giới thiệu	1
1.1	Lý do hình thành đề tài	1
1.2	Mục tiêu	1
1.3	Phạm vi nghiên cứu	2
1.4	Phương pháp thực hiện	2
2	Cơ sở lý thuyết	4
2.1	Object Detection	4
2.2	Image Classification	5
2.3	Lựa chọn mô hình	5
2.4	Khái niệm về YOLO (you only look one)	6
2.4.1	Giới thiệu	6
2.4.2	Kiến trúc mạng YOLO	6
2.4.3	Khái niệm về IoU và hàm tính về IoU	8
2.4.4	Cách YOLO hoạt động	9
2.5	YOLOv8	10
2.5.1	Giới thiệu	10
2.5.2	Kiến trúc mạng YOLOv8	10
2.5.3	So sánh với một số phiên bản khác	12
2.5.4	Các điểm cải tiến của YOLOv8 so với các phiên bản khác	13
2.6	Model Xception	14
2.6.1	Giới thiệu	14
2.6.2	Model Inception	14
2.6.3	Kiến trúc Xception	17

2.6.4	Hiệu suất và Điểm chuẩn	22
2.6.5	Ứng dụng của mô hình Xception	24
3	Nội dung thực hiện	28
3.1	GUI	28
3.2	DATASET	29
3.2.1	Lựa chọn tập dữ liệu DATASET	29
3.2.2	Xử lý DATASET	29
3.3	Training Model	33
3.3.1	Code Environment	33
3.3.2	Training YOLOv8	34
3.3.3	Training Xception	38
3.3.4	Real time và thực tế	53
3.4	Đánh giá hiệu suất	57
3.4.1	Model YOLOv8	57
3.4.2	Model Xception với Validation	58
3.4.3	Model Xception với Test	58
4	Ưu điểm và Nhược điểm	61
4.1	Ưu điểm	61
4.2	Nhược điểm	62
5	Giải pháp và Kết luận	64
5.1	Giải pháp	64
5.1.1	Nguyên nhân	64
5.1.2	Hướng giải pháp	65
5.2	Kết luận	65
6	Source	67
6.1	DATASET	67
6.2	Code	67
6.3	Weight	68

Danh sách bảng

3.1	Bảng thống kê số lượng Data train YOLOv8	31
3.2	Bảng thống kê Data train Xception	33

Danh sách hình vẽ

2.1	Ví dụ về Object Detection	4
2.2	Ví dụ về Image Classification	5
2.3	Bảng so sánh độ chính xác	6
2.4	Hàm kích hoạt ReLu	7
2.5	Kiến trúc YOLO	8
2.6	Công thức của IoU	8
2.7	Quy trình tìm Bounding Box	10
2.8	Kiến trúc YOLOv8, hình ảnh trực quan được thực hiện bởi người dùng GitHub RangeKing	11
2.9	So sánh về kích thước và tốc độ	12
2.10	Các phiên bản pretain models của YOLOv8	12
2.11	Inception Model	15
2.12	A canonical Inception module (Inception V3)	15
2.13	A simplified Inception module	16
2.14	Xception Concept	17
2.15	Kiến trúc Xception	18
2.16	(a) Standard CNN (b) Depthwise Separable	19
2.17	Dropout: (a) full network; (b) network after dropout	21
2.18	Bảng so sánh Xception với các model khác	23
2.19	Bảng so sánh Xception với Inception V3	23
2.20	Residual connection in Xception	23
2.21	Ảnh chụp màn hình hình ảnh thảo mộc hình dạng đầu vào và kết quả dự đoán trong ứng dụng di động HerbSnap	24

2.22	Hình ảnh phần mềm độc hại thang độ xám	25
2.23	Độ chính xác xác thực của các phương pháp khác nhau trên tập dữ liệu Malimg	25
2.24	Các bệnh thực vật chính	26
2.25	So sánh độ chính xác xác thực của bảy mô hình với L2 và L2M	26
2.26	Hình ảnh từ các lớp khác nhau	27
2.27	So sánh các mô hình cho tập dữ liệu ba lớp	27
3.1	Giao diện phần mềm	28
3.2	Xử lý dữ liệu train YOLOv8	30
3.3	Hình ảnh khi đã phân (trong 1 class)	30
3.4	Data sau khi được tổng hợp và up lên kaggel	31
3.5	Xử lý dữ liệu train Data Xception	32
3.6	Tổng hợp và label ảnh	32
3.7	Tạo 1 file class.csv tương ứng	32
3.8	Hướng dẫn sử dụng Kaggle	34
3.9	YOLO training flowchart	34
3.10	Tập dữ liệu sau khi Training YOLOv8	36
3.11	Dự đoán và hiển thị ảnh từ model YOLO8v đã training	37
3.12	Xception training flowchart	38
3.13	Output of Xception	46
3.14	Kiểm tra và đánh giá Model phân loại	47
3.15	Plot incorrect predictions	52
3.16	Plot correct predictions	53
3.17	Real time thực tế	56
3.18	Accuracy of YOLOv8	57
3.19	Accuracy of Xception	58
3.20	Hiệu suất khi Test	59
3.21	Số lượng dự đoán Correct và Incorrect trong tập Test	60
3.22	Đồ thị đánh giá Model sau training	60
4.1	Tốc độ Real time thực tế	61

4.2	Biến dạng nhỏ	62
4.3	Kết quả khi Test lại lần 2	62
4.4	Sự sai lệch trong dự đoán của Model khi thay đổi góc quay	63
5.1	Sự ảnh hưởng của góc máy và ánh sáng	64

Chương 1

Giới thiệu

Trong các khu công nghiệp, có rất nhiều ngành cần phân loại lon biến dạng theo mức độ để tiến hành tái chế hay sử dụng với mục đích khác. Tuy nhiên, việc nhận diện và phân loại sai mức độ m López của lon có thể dẫn đến nhiều hao phí trong quá trình sản xuất hay tái chế. Để giải quyết vấn đề này, việc sử dụng hệ thống giám sát camera kết hợp với công nghệ phân loại đối tượng (Classification Object) có thể giúp phân loại chính xác và dễ dàng nhận biết khi giám sát quá trình.

1.1 Lý do hình thành đề tài

Hiện nay, có rất nhiều ngành công nghiệp cần phân loại lon biến dạng theo mức độ trong quá trình công nghiệp. Nhưng vẫn còn những hệ thống camera giám sát chưa thực sự chính xác và tốc độ xử lý lâu khiến cho quá trình công nghiệp không thực sự tối ưu và đạt hiệu quả cao nhất. Do đó nhóm quyết định chọn đề tài **"Phân loại lon biến dạng theo mức độ - Classification Of Dented Cans By Degree"** để tạo ra một hệ thống phân loại hiệu quả hơn.

1.2 Mục tiêu

Mục tiêu của đề tài này bao gồm:

- Nâng cao độ chính xác: Sử dụng công nghệ phân loại đối tượng để đảm bảo độ chính xác khi phân loại trong điều kiện ánh sáng, góc độ camera khác nhau trong

nha máy.

- Tối ưu hóa quản lý nhân sự: Thay vì thuê nhân công để giám sát khu vực hạn chế, sử dụng hệ thống giám sát camera kết hợp với cảm biến giúp giám chi phí nhân công và tăng cường hiệu suất giám sát.
- Tăng cường tính tự động hóa: Bằng cách tích hợp công nghệ phân loại đối tượng vào hệ thống giám sát, có thể tạo ra một hệ thống tự động hóa thông minh hơn, giúp quản lý và chính xác một cách chủ động và hiệu quả.
- Cải thiện thời gian phản ứng: Nhờ vào khả năng phân loại chính xác và xử lý dữ liệu nhanh, hệ thống giám sát kết hợp với công nghệ phân loại đối tượng giúp cải thiện thời gian phân loại trong công nghiệp

1.3 Phạm vi nghiên cứu

Các hệ thống camera trong nhà máy công nghiệp. Do cơ sở dữ liệu còn hạn chế nên trước mắt nhóm sẽ chỉ nghiên cứu về phân loại lon sữa bằng nhôm.

1.4 Phương pháp thực hiện

Phương pháp thực hiện trong đề tài này sử dụng một phương pháp **"state two state"**, trong đó bao gồm hai giai đoạn chính.

Trong giai đoạn đầu tiên, sử dụng một mô hình nhận diện vật thể để phát hiện và xác định các vật thể trong hình ảnh từ dữ liệu camera. Các mô hình nhận diện vật thể như Faster R-CNN, YOLO, và SSD được áp dụng để xác định vị trí và loại của các vật thể như con người, xe cộ, và các đối tượng không mong muốn trong khung hình.

Tiếp theo, trong giai đoạn thứ hai, sử dụng một mô hình phân loại vật thể để nhận diện và phân loại hình ảnh dựa trên mạng Convolutional Neural Networks (CNN) để phân loại hình ảnh đầu vào thành các class hợp lý và chính xác. Các mô hình phục vụ bài toán phân loại hình ảnh (Classification) từ cơ bản đến phức tạp có thể kể đến như CNN, VGGNet, ResNet, Inception/GoogLeNet, Xception, EfficientNet, Vision Transformer (ViT), Swin Transformer, ConvNeXt,... tùy theo mức độ phức tạp, độ lớn của dữ liệu

mà chọn mô hình cho phù hợp.

Ở đây nhóm sử dụng mô hình **YOLOv8** chuyên về các bài toán Object Detection để nhận diện lon trên băng chuyền kết hợp với mô hình **Xception** cho bài toán phân loại lon theo mức độ biến dạng của sản phẩm để tối ưu độ chính xác nhất.

Chương 2

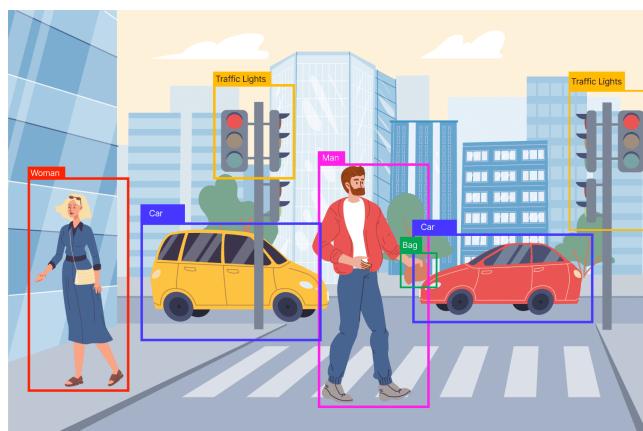
Cơ sở lý thuyết

2.1 Object Detection

Nhận diện vật thể (object detection) là một lĩnh vực trong trí tuệ nhân tạo và thị giác máy tính, chuyên về việc xác định danh tính và vị trí của các đối tượng trong hình ảnh hoặc video.

Trong khi đó, định vị hình ảnh là quá trình xác định vị trí chính xác của một hoặc nhiều đối tượng bằng cách sử dụng các hộp giới hạn, thường là các hình chữ nhật, để bao quanh các đối tượng đó.

Tuy nhiên, đôi khi quá trình này có thể bị nhầm lẫn với việc phân loại hình ảnh hoặc nhận dạng hình ảnh, mục tiêu của chúng là dự đoán lớp hoặc loại của một đối tượng trong hình ảnh và gán nó vào một trong các danh mục hoặc lớp đã xác định trước.

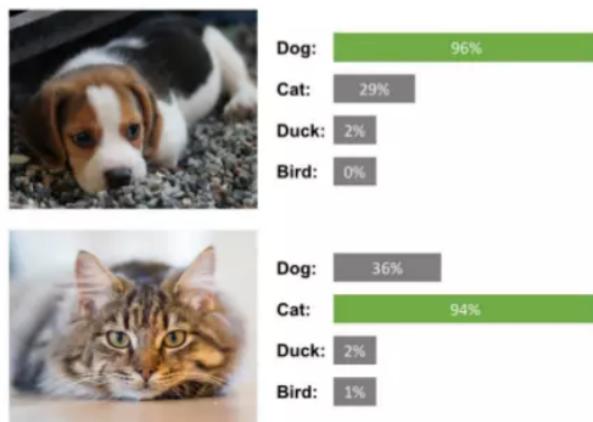


Hình 2.1: Ví dụ về Object Detection

2.2 Image Classification

Image Classification hay hiểu đơn giản là phân loại hình ảnh là một trong những nhiệm vụ phổ biến trong Computer Vision. Mục tiêu chính của bài toán này đó chính là phân loại một hình ảnh đầu vào (input) thành một nhãn (label) đầu ra (output). Một ví dụ đơn giản chúng ta cần phân biệt bức ảnh đầu vào là con chó hay con mèo chẳng hạn.

Có thể thấy Image Classification là một trong những nhiệm vụ cơ bản nhất của Computer Vision sẽ đưa ra một nhãn (còn gọi là class hay label) tương ứng với mỗi image kèm theo một chỉ số confidence (độ chắc chắn). Ứng dụng của nó có thể kể đến như phân loại động vật (chó mèo, gà vịt, ngan ngỗng...), phân loại giới tính, nhận dạng mặt người.



Hình 2.2: Ví dụ về Image Classification

2.3 Lựa chọn mô hình

Hiện tại có rất nhiều mô hình dùng để nhận diện khuôn mặt trong có có 3 yếu tố để chọn một mô hình:

- **Tốc độ:** Việc nhận diện vật thể và phản hồi trong thời gian thực luôn là yêu cầu hàng đầu. Hiệu suất nhận diện vật thể nhanh chóng giúp hệ thống dễ dàng phát hiện các yếu tố bất thường và thông báo kịp thời cho người dùng.
- **Độ chính xác:** Độ chính xác luôn là một trong những yếu tố hàng đầu giúp ta loại xác định đúng được vật thể và loại bỏ các yếu tố nhiễu.

- Tính tổng quát cao: Các mô hình cần có tính đa dạng và không nên quá phụ thuộc vào dữ liệu cụ thể, để có thể áp dụng linh hoạt trong nhiều tình huống khác nhau.

Network	Validation accuracy(L2) (%)	Validation accuracy (L2M) (%)	Change
AlexNet	56.31	57.31	1.00% (+)
ResNet50	78.64	79.34	0.7% (+)
Xception	92.23	93.85	1.62% (+)
SENet154	62.14	62.84	0.7% (+)
DenseNet169	82.90	80.58	2.32% (-)
HRNet-w48	78.13	78.00	0.13% (-)
MobileNetV3	65.69	66.01	0.32% (+)

Hình 2.3: Bảng so sánh độ chính xác

Từ các yếu tố trên, nhóm quyết định chọn YOLO cho ứng dụng nhận diện vật thể và Xception cho ứng dụng phân loại lon biển dạng. Bằng cách sử dụng kết hợp cả 2, việc nhận diện và phân loại có thể đạt được hiệu suất cao và tính linh hoạt trong việc áp dụng vào nhiều tình huống khác nhau.

2.4 Khái niệm về YOLO (you only look one)

2.4.1 Giới thiệu

YOLO là một mô hình mạng CNN cho việc phát hiện, nhận dạng, phân loại đối tượng. YOLO được tạo ra từ việc kết hợp giữa các convolutional layers và connected layers. Trong đó các convolutional layers sẽ trích xuất ra các feature của ảnh, còn full-connected layers sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.

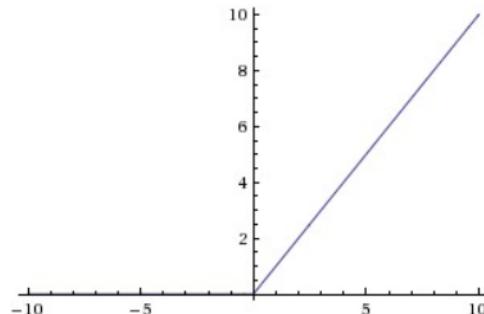
2.4.2 Kiến trúc mạng YOLO

Kiến trúc hoạt động như sau:

- Thay đổi kích thước ảnh đầu vào: Ảnh đầu vào được thay đổi kích thước thành kích thước cố định là 448x448 trước khi được xử lý bởi mạng tích chập. Bước thay đổi kích thước này giúp chuẩn hóa kích thước đầu vào và đảm bảo tính nhất quán trong đầu vào của mô hình.

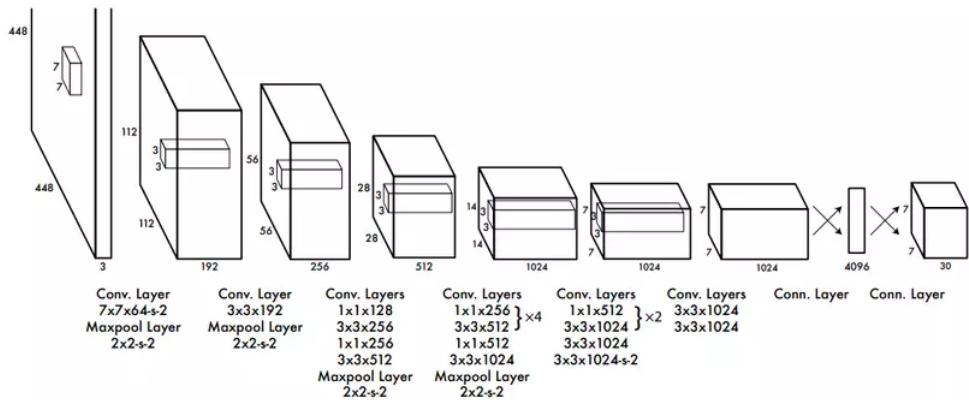
- Tích chập 1×1 tiếp theo bởi tích chập 3×3 : Ảnh đầu vào, sau khi được thay đổi kích thước, trải qua một loạt các phép tích chập. Ban đầu, một tích chập 1×1 được áp dụng để giảm số kênh, hiệu quả nén các bản đồ đặc trưng đầu vào. Tiếp theo, một tích chập 3×3 được thực hiện để tạo ra một đầu ra hình hộp, thu thập thông tin không gian trên các bản đồ đặc trưng đầu vào.
- Hàm kích hoạt: Trong toàn bộ mạng, hàm kích hoạt Rectified Linear Unit (ReLU) 2.4 được sử dụng, ngoại trừ lớp cuối cùng. ReLU là một hàm kích hoạt phổ biến được sử dụng để giới thiệu tính phi tuyến vào mô hình bằng cách đặt tất cả các giá trị âm thành không. Tuy nhiên, ở lớp cuối cùng, một hàm kích hoạt tuyến tính được sử dụng, cho phép mô hình xuất ra các giá trị thực không giới hạn, phù hợp cho các nhiệm vụ hồi quy.

$$f(x) = \begin{cases} x & \forall x > 0 \\ 0 & \forall x < 0 \end{cases}$$



Hình 2.4: Hàm kích hoạt ReLU

- Các kỹ thuật chính quy hóa: Chuẩn hóa theo batch và dropout được sử dụng như các kỹ thuật chính quy hóa bổ sung để ngăn chặn việc mô hình quá mức phù hợp. Chuẩn hóa theo batch chuẩn hóa các kích hoạt của mỗi lớp trên một mini-batch, làm cho quá trình huấn luyện ổn định hơn và tăng tốc quá trình hội tụ. Dropout ngẫu nhiên đặt một phần tử đầu vào thành không trong quá trình huấn luyện, giảm sự phụ thuộc giữa các nơ-ron và ngăn chặn mô hình phụ thuộc quá mức vào các đặc trưng cụ thể.

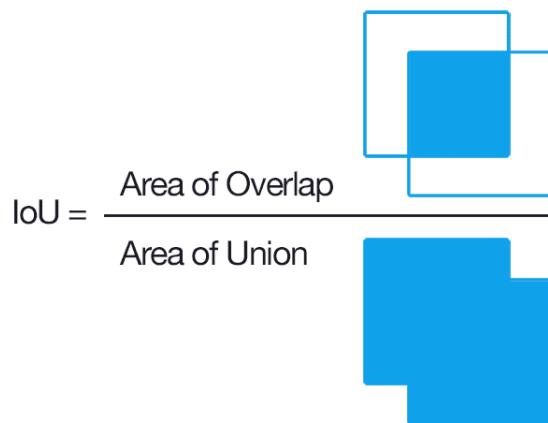


Hình 2.5: Kiến trúc YOLO

Tóm lại, kiến trúc này kết hợp các lớp tích chập, hàm kích hoạt và các kỹ thuật chính quy hóa để hiệu quả xử lý và trích xuất đặc trưng từ ảnh đầu vào trong khi giảm thiểu rủi ro của việc mô hình quá mức phù hợp (overfitting).

2.4.3 Khái niệm về IoU và hàm tính về IoU

IOU (INTERSECTION OVER UNION) là hàm đánh giá độ chính xác của object detector trên tập dữ liệu cụ thể. Trong đó Area of Overlap là diện tích phần giao nhau giữa predicted bounding box với ground-truth bounding box, còn Area of Union là diện tích phần hợp giữa predicted bounding box với ground-truth bounding box. Những bounding box được đánh nhãn bằng tay trong tập training set và test set.



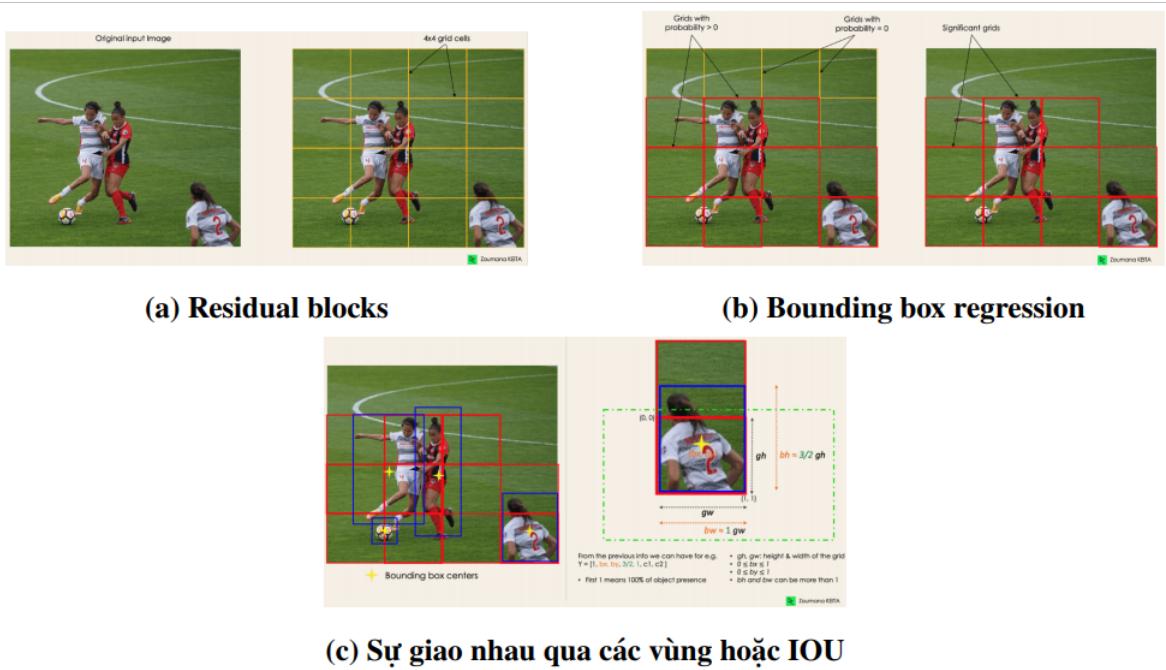
Hình 2.6: Công thức của IoU

2.4.4 Cách YOLO hoạt động

- **Các khối dư thừa:** Bước đầu tiên này bắt đầu bằng cách chia hình ảnh gốc (A) thành các ô lưới NxN có hình dạng bằng nhau. Mỗi ô trong lưới có trách nhiệm về việc xác định vị trí và dự đoán lớp của đối tượng mà nó bao phủ, cùng với giá trị xác suất.
- **Hồi quy hộp giới hạn:** Bước tiếp theo là xác định các hộp giới hạn tương ứng với các hình chữ nhật làm nổi bật tất cả các đối tượng trong hình ảnh. Từ đó có thể có nhiều hộp giới hạn như có bao nhiêu đối tượng trong một hình ảnh cụ thể. YOLO xác định các thuộc tính của các hộp giới hạn này bằng cách sử dụng một mô-đun hồi quy duy nhất theo định dạng sau, trong đó Y là biểu diễn vector cuối cùng cho mỗi hộp giới hạn:

$$Y = [pc, bx, by, bh, bw, c1, c2, \dots cn]$$

- Sự giao nhau qua các liên minh hoặc IOU: Hầu hết các thời điểm, một đối tượng duy nhất trong một hình ảnh có thể có nhiều ứng viên hộp lưới để dự đoán, ngay cả khi không phải tất cả đều có liên quan. Mục tiêu của IOU (một giá trị giữa 0 và 1) là loại bỏ các ô lưới như vậy để chỉ giữ lại những ô lưới có liên quan. Dưới đây là lý do logic đằng sau:
 - Người dùng xác định ngưỡng chọn IOU của mình, ví dụ chọn ngưỡng là 0.5.
 - Sau đó, YOLO tính toán IOU của mỗi ô lưới là diện tích giao chia cho Diện tích Liên minh.
 - Cuối cùng, nó bỏ qua việc dự đoán của các ô lưới có $IOU <= \text{ngưỡng}$ và xem xét những ô lưới có $IOU > \text{ngưỡng}$.
- Non-Max Suppression hoặc NMS: Việc thiết lập một ngưỡng cho IOU không luôn đủ vì một đối tượng có thể có nhiều hộp với IOU vượt qua ngưỡng, và việc bỏ đi tất cả các hộp đó có thể bao gồm nhiều. Đây là nơi có thể sử dụng NMS để chỉ giữ lại các hộp có điểm xác suất phát hiện cao nhất 2.7.



Hình 2.7: Quy trình tìm Bounding Box

2.5 YOLOv8

2.5.1 Giới thiệu

YOLOv8 là một trong những mô hình YOLO tiên tiến nhất có thể được sử dụng cho các tác vụ phát hiện đối tượng, phân loại hình ảnh và cung cấp hiệu suất tiên tiến về độ chính xác và tốc độ. YOLOv8 được phát triển bởi Ultralytics, công ty cũng đã tạo ra mô hình YOLOv5 có ảnh hưởng và định hình ngành. YOLOv8 bao gồm nhiều thay đổi và cải tiến về kiến trúc và trải nghiệm của nhà phát triển so với YOLOv5. Xây dựng dựa trên những tiến bộ của trước đây YOLO phiên bản, YOLOv8 giới thiệu các tính năng và tối ưu hóa mới giúp sản phẩm trở thành lựa chọn lý tưởng cho nhiều tác vụ phát hiện đối tượng trong nhiều ứng dụng khác nhau.

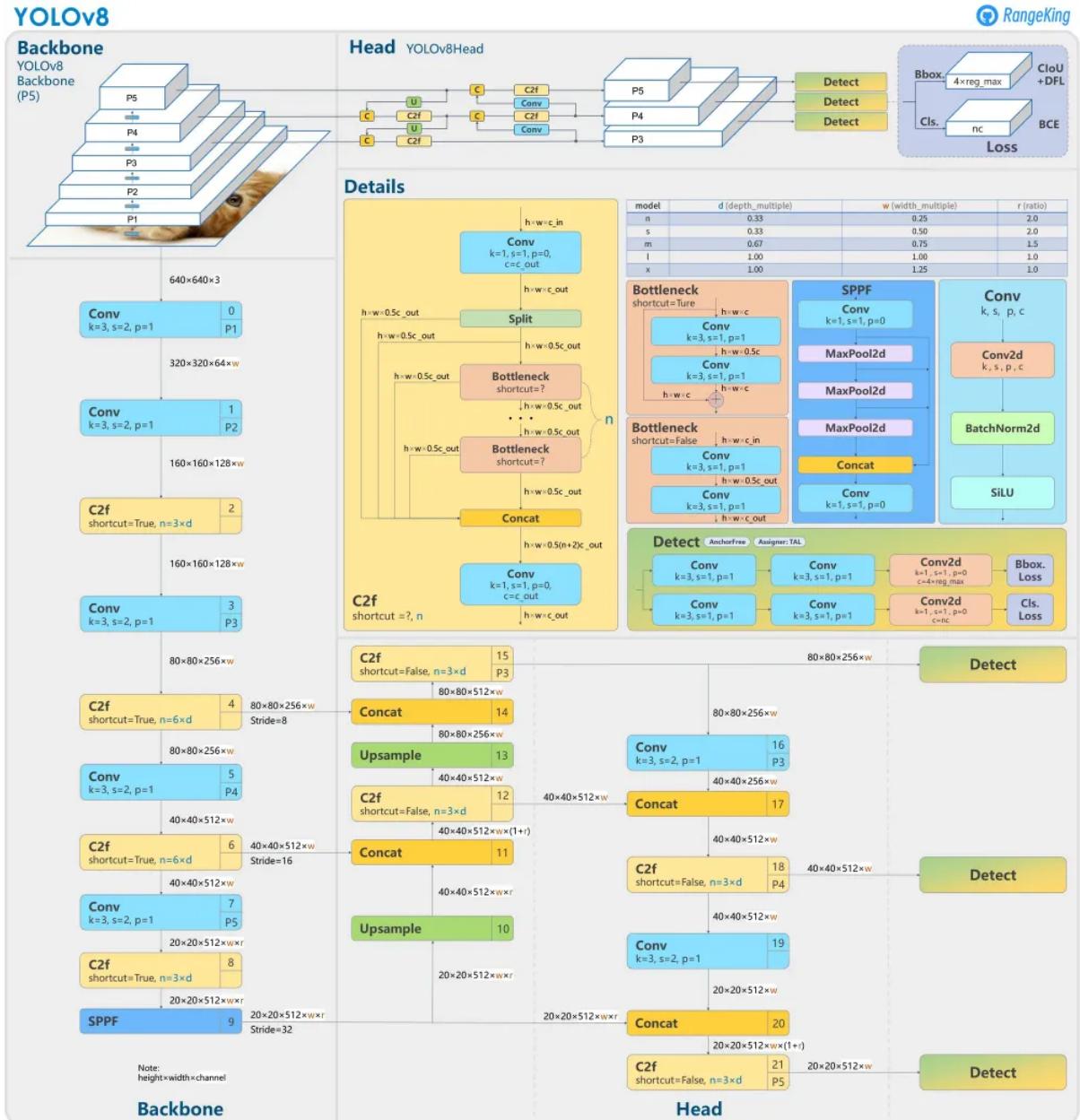
2.5.2 Kiến trúc mạng YOLOv8

YOLOv8 vẫn chưa có bài báo được công bố, vì vậy nhóm không có cái nhìn sâu sắc trực tiếp về phương pháp nghiên cứu trực tiếp và các nghiên cứu cắt bỏ được thực

hiện trong quá trình tạo ra nó.

Sau đây nhóm cung cấp bản tóm tắt nhanh về các cập nhật mô hình có tác động lớn và sau đó chúng ta sẽ xem xét đánh giá mô hình, điều này tự nói lên tất cả.

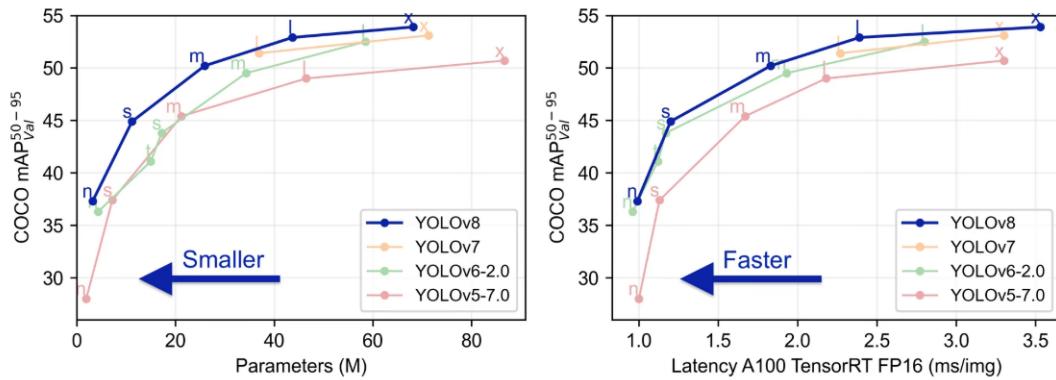
Hình ảnh sau đây do người dùng RangeKing trên GitHub tạo ra cho thấy hình ảnh trực quan chi tiết về kiến trúc mạng.



Hình 2.8: Kiến trúc YOLOv8, hình ảnh trực quan được thực hiện bởi người dùng GitHub RangeKing

2.5.3 So sánh với một số phiên bản khác

Ta có thể thấy phiên bản YOLOv8 có sự cân bằng về tốc độ và kích thước, điều này có thể giúp áp dụng trên một số thiết bị bị giới hạn về phần cứng lẫn tốc độ xử lý trong thời gian thực 2.9.



Hình 2.9: So sánh về kích thước và tốc độ

Model	size (pixels)	mAP _{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

- mAP_{val} values are for single-model single-scale on [COCO val2017](#) dataset.
Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- Speed averaged over COCO val images using an [Amazon EC2 P4d](#) instance.
Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

Hình 2.10: Các phiên bản pretrain models của YOLOv8

Ngoài ra, YOLOv8 có nhiều phiên bản pretrain model 2.10 với có kích thước của kiến trúc khác nhau, điều này giúp người dùng có thể linh động và tối ưu hóa hiệu năng trong từng công việc.

- Size: Kích thước ảnh đầu vào trước khi đưa vào mô hình xử lý.

- mAPval: Là một chỉ số đánh giá hiệu suất của mô hình trong tác vụ phát hiện đối tượng. mAP (mean Average Precision) là trung bình của các giá trị precision-recall (chính xác - độ phủ) cho tất cả các lớp. mAP cao hơn thường tượng trưng cho việc mô hình có khả năng phát hiện đối tượng hiệu quả.
- CPU ONNX Speed: Thời gian mà mô hình cần để xử lý mỗi hình ảnh trên một CPU khi sử dụng định dạng ONNX.
- Speed A100 TensorRT: Thời gian mà mô hình cần để xử lý mỗi hình ảnh trên một A100 GPU khi sử dụng TensorRT.
- Params (tham số): Số lượng tham số trong mô hình.
- FLOPs (Floating Point Operations - Phép toán dấu chấm động): Số lượng phép toán dấu chấm động cần thiết để thực hiện một lượt chạy qua mô hình.

2.5.4 Các điểm cải tiến của YOLOv8 so với các phiên bản khác

Phiên bản YOLOv8 có 2 điểm cải tiến chính so với các phiên bản còn lại:

- **Phát hiện không sử dụng Anchor Boxe:** Đây là các công cụ được sử dụng để phân loại hai đối tượng có cùng tâm. Trong YOLOv8, kiến trúc đã bỏ qua việc sử dụng Anchor Box vì một số lý do:
 - Thiếu tính tổng quát: Huấn luyện với các Anchor Box có sẵn làm cho mô hình cứng nhắc và khó phù hợp với dữ liệu mới.
 - Không có Anchor Box phù hợp cho các tình huống không đều: Các tình huống không đều không thể được ánh xạ một cách rõ ràng bằng các Anchor Box hình đa giác.
- **Kỹ thuật tăng cường dữ liệu Mosaic:**
 - Tăng cường Mosaic là một kỹ thuật đơn giản, trong đó bốn hình ảnh khác nhau được nối lại với nhau và đưa vào mô hình như đầu vào. Điều này giúp mô hình học được các đối tượng thực sự từ các vị trí khác nhau và trong trạng thái bị che khuất.

- Kỹ thuật tăng cường Mosaic được nhận định là làm giảm hiệu suất, vì vậy nó đã được sử dụng cho 10 epoch cuối cùng.

(Một Epoch được tính khi tất cả dữ liệu trong tập huấn luyện được đưa vào mạng neural network 1 lần. Ví dụ, nếu bạn có 10 triệu hình ảnh trong tập huấn luyện, và bạn đưa toàn bộ số hình ảnh đó vào mô hình 3 lần, bạn đã huấn luyện trong 3 epoch.

Khi dữ liệu quá lớn không thể đưa hết mỗi lần toàn bộ tập dữ liệu vào để huấn luyện được, vì bạn cần một siêu máy tính với lượng RAM và GPU RAM cực lớn để lưu trữ tất cả các hình ảnh, điều này là không khả thi đối với người dùng bình thường hoặc môi trường phòng thí nghiệm nhỏ. Do đó, cần phải chia nhỏ tập dữ liệu, và khái niệm batch được hình thành.)

2.6 Model Xception

2.6.1 Giới thiệu

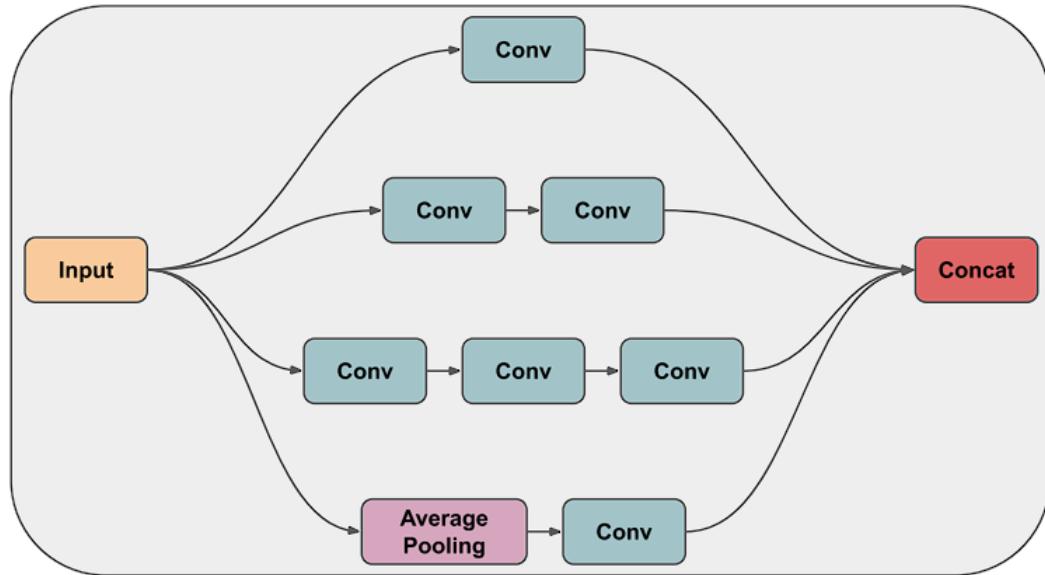
Xception (Extreme Inception) là một mô hình Học sâu được Francois Chollet tại Google phát triển và được giới thiệu vào năm 2017. Mô hình Xception dựa trên ý tưởng của Inception và sử dụng các Convolution Depthwise Separable để cải thiện hiệu suất và hiệu quả tính toán.

Kiến trúc Inception sử dụng các Module Inception, tuy nhiên, mô hình Xception thay thế nó bằng các lớp tích chập có thể tách theo chiều sâu, tổng cộng là 36 lớp. Khi chúng ta so sánh mô hình Xception với mô hình Inception V3, nó chỉ hoạt động tốt hơn một chút trên tập dữ liệu ImageNet, tuy nhiên, trên các tập dữ liệu lớn hơn bao gồm 350 triệu hình ảnh, Xception hoạt động tốt hơn đáng kể.

2.6.2 Model Inception

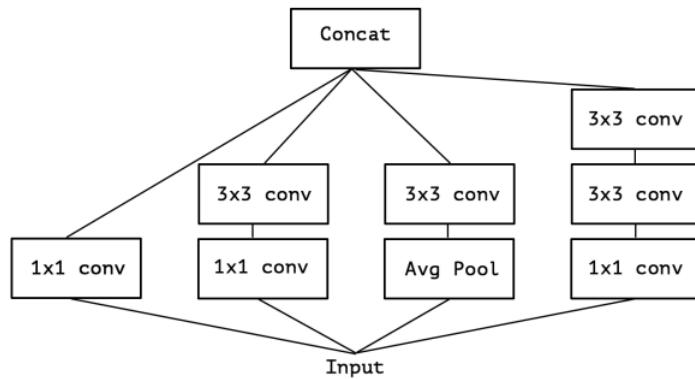
Convolution Layer là một phần quan trọng trong mạng neural, đặc biệt là trong việc xử lý dữ liệu như hình ảnh. Convolution Layer được sử dụng để học các đặc trưng từ dữ liệu. Các Convolution Layer có gắng tìm hiểu và gọc các bộ lọc (filter) trong một không gian 3D. Không gian 3D này gồm hai chiều không gian (spatial dimension) là chiều rộng,

chiều cao (tương quan không gian) và các kênh (channel dimension) (tương quan chéo kênh). Ví dụ, một ảnh màu RGB sẽ có ba kênh là đỏ (Red), xanh lá cây (Green) và xanh lam (Blue).



Hình 2.11: Inception Model

Tuy nhiên, Model Inception phân chia nhiệm vụ tương quan không gian và kênh chéo bằng cách sử dụng các bộ lọc có kích thước khác nhau (1×1 , 3×3 , 5×5) song song, đây là cách hiệu quả và tốt hơn để học bộ lọc.



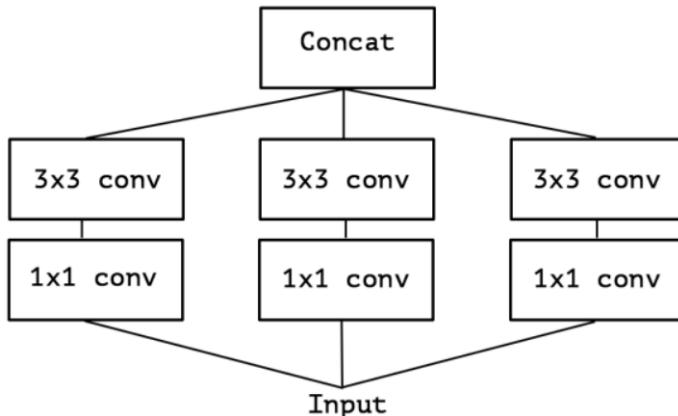
Hình 2.12: A canonical Inception module (Inception V3)

Module Inception được thiết kế để giúp quá trình học các tương quan giữa các Channel và Spatial hiệu quả hơn. Cụ thể, một Module Inception điển hình đầu tiên sử

dụng các Convolution (1×1) (như hình trên), lý do:

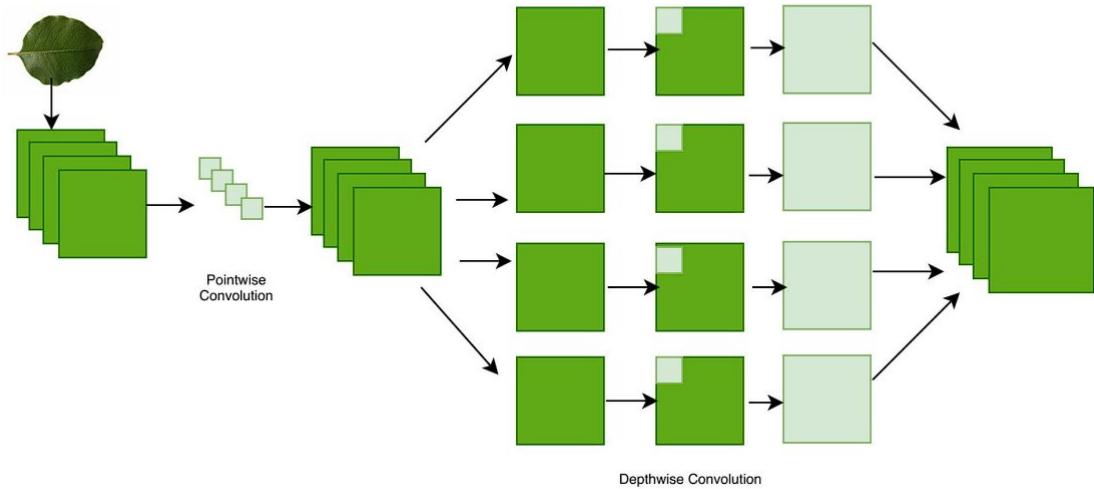
- Giảm số lượng kênh (Channel Reduction): Các Conv (1×1) giúp giảm số lượng kênh của input ban đầu. Việc này hữu ích trong việc giảm chi phí tính toán, giảm thiểu khả năng bị Overfitting (hiện tượng mô hình quá tập trung vào dữ liệu huấn luyện mà không tổng quát hóa tốt cho dữ liệu mới) và tạo ra một biểu diễn với chiều kênh thấp hơn, giúp tốn ít bộ nhớ hơn và tiết kiệm thời gian tính toán.
- Học tương quan giữa các kênh (Channel-wise Correlation): Sử dụng Conv (1×1) trên các kênh tương tự như việc thực hiện một loạt các phép nhân ma trận độc lập trên các kênh. Điều này cho phép mô hình học tương quan giữa các kênh nhưng không liên quan đến không gian. Điều này rất hữu ích vì Module Inception sau đó sử dụng các Conv lớn hơn như (3×3) hoặc (5×5) để học tương quan không gian trong không gian nhỏ hơn được tạo ra bởi các Conv (1×1).

Một phiên bản đơn giản hơn của module Inception là vẫn dùng Conv (1×1) và Conv (3×3) nhưng không dùng Avg pool như hình dưới:



Hình 2.13: A simplified Inception module

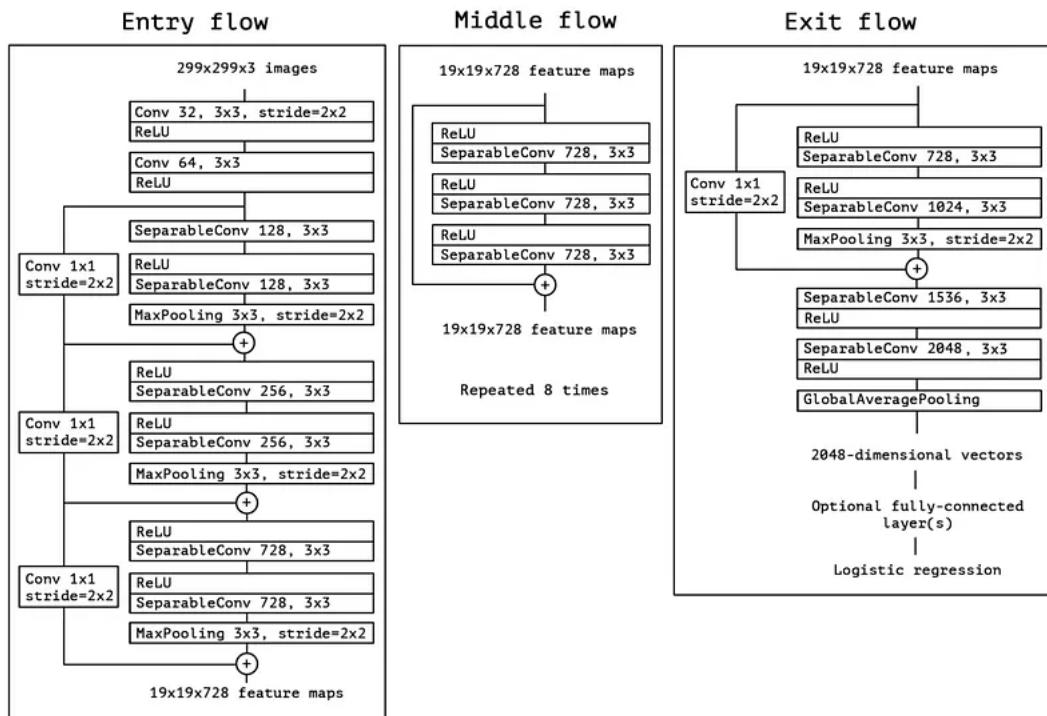
Mô hình Xception có cách tiếp cận thậm chí còn tích cực hơn vì nó tách biệt hoàn toàn nhiệm vụ của tương quan không gian và kênh chéo. Điều này đã mang lại cho nó cái tên Extreme Inception Model.



Hình 2.14: Xception Concept

2.6.3 Kiến trúc Xception

Kiến trúc tổng thể của Xception được mô tả tường minh trong hình dưới. Kiến trúc gồm 3 phần Entry flow, Middle flow, Exit flow. Có tổng cộng 14 module trong kiến trúc, trong đó Entry flow có 4 module, Middle flow có 8 module và Exit flow có 2 module. Mỗi module là tập hợp của Depthwise Seperable Convolution và Pooling Layer. Các Layer cuối của kiến trúc là các Fully Connected Layer.



Hình 2.15: Kiến trúc Xception

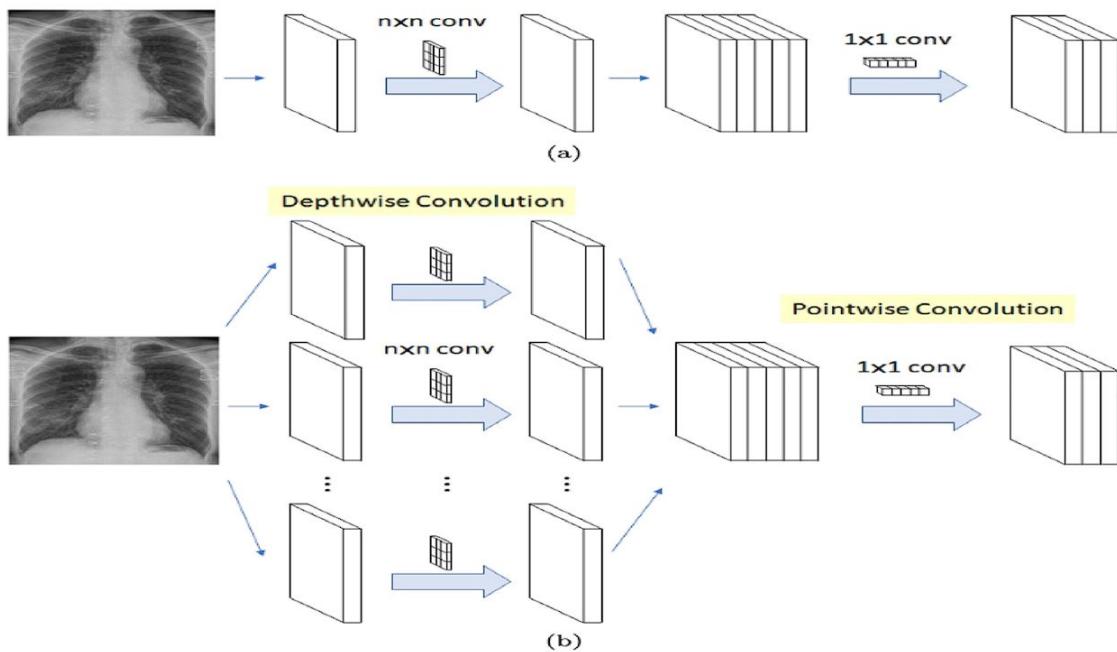
Lõi của mô hình Xception được tạo thành từ các phép tích chập có thể tách theo chiều sâu. Do đó, trước khi đi sâu vào từng thành phần của kiến trúc Xception, chúng ta hãy xem xét phép tích chập có thể tách theo chiều sâu.

Depthwise Separable Convolution

Tích chập tiêu chuẩn học các bộ lọc trong không gian 3D, trong đó mỗi Kernel học chiều rộng, chiều cao và kênh.

Trong khi đó, phép tích chập tách theo chiều sâu chia quá trình thành hai quá trình riêng biệt bằng cách sử dụng phép tích chập theo chiều sâu (Depthwise Convolution) và phép tích chập theo điểm (Pointwise convolution):

- Tích chập theo chiều sâu (Depthwise Convolution): Ở đây, một bộ lọc duy nhất được áp dụng cho từng kênh đầu vào riêng biệt. Ví dụ, nếu một hình ảnh có ba kênh màu (đỏ, lục và lam), một bộ lọc riêng biệt được áp dụng cho từng kênh màu.
- Tích chập theo điểm (Pointwise Convolution): Sau khi thực hiện depthwise convolution, một pointwise convolution được áp dụng. Đây là bộ lọc (1×1) kết hợp đầu ra của depthwise convolution thành một bản đồ đặc trưng duy nhất.



Hình 2.16: (a) Standard CNN (b) Depthwise Separable

Mô hình Xception sử dụng một phiên bản được sửa đổi đôi chút của mô hình này. Trong phép tích chập tách biệt theo chiều sâu ban đầu, trước tiên chúng ta thực hiện tích chập theo chiều sâu (Depthwise Convolution), sau đó là tích chập theo điểm (Pointwise Convolution). Mô hình Xception thực hiện tích chập theo điểm trước (1×1), sau đó thực hiện tích chập theo chiều sâu bằng nhiều bộ lọc ($n \times n$) khác nhau.

Ba phần của kiến trúc Xception

Toàn bộ kiến trúc Xception chia thành ba phần chính: luồng vào (Entry Flow), luồng giữa (Middle Flow) và luồng ra (Exit Flow), với các kết nối bù qua xung quanh 36 lớp.

- Entry Flow:
 - Hình ảnh đầu vào có kích thước (299×299) pixel với 3 kênh (RGB).
 - Lớp tích chập (3×3) được sử dụng với 32 bộ lọc và bước tiến (2×2). Điều này làm giảm kích thước hình ảnh và trích xuất các đặc điểm cấp thấp. Để đưa vào tính phi tuyến tính, hàm kích hoạt ReLU được áp dụng.
 - Tiếp theo là lớp tích chập (3×3) khác với 64 bộ lọc và ReLU.

- Sau khi trích xuất tính năng cấp thấp ban đầu, lớp tích chập có thể tách theo chiều sâu đã sửa đổi được áp dụng, cùng với lớp tích chập 1×1 . Tổng hợp tối đa $[(3 \times 3), stride = 2]$ làm giảm kích thước của bản đồ tính năng.
- Middle Flow:
 - Khối này được lặp lại tám lần.
 - Mỗi lần lặp lại bao gồm:
 - * Tích chập phân tách theo chiều sâu với 728 bộ lọc và Kernel (3×3) .
 - * ReLU activation.
 - Bằng cách lặp lại tám lần, luồng ở giữa sẽ dần trích xuất các đặc điểm cấp cao hơn từ hình ảnh.
- Exit Flow:
 - Tích chập tách rời với các bộ lọc 728, 1024, 1536 và 2048, tất cả đều có hạt nhân (3×3) để trích xuất thêm các tính năng phức tạp.
 - Global Average Pooling tóm tắt toàn bộ bản đồ đặc điểm thành một vector duy nhất.
 - Cuối cùng, một lớp được kết nối đầy đủ với Hồi quy Logistic sẽ phân loại hình ảnh.

Regularization Techniques

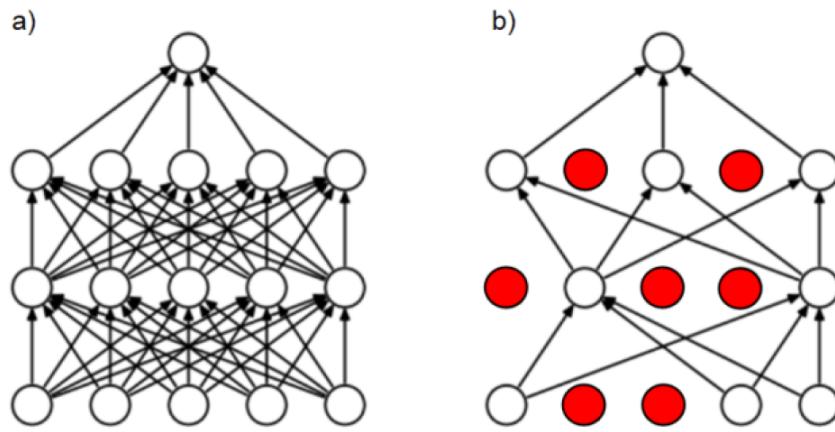
Các mô hình học sâu hướng đến mục tiêu khái quát hóa (khả năng thích ứng phù hợp của mô hình với dữ liệu mới, chưa từng thấy trước đó), trong khi quá trình học quá mức ngăn mô hình khái quát hóa.

Overfitting: khi một mô hình học nhiều từ dữ liệu đào tạo hoặc học quá mức dữ liệu đào tạo. Các kỹ thuật chính quy hóa giúp ngăn ngừa quá khớp trong các mô hình học máy. Mô hình Xception sử dụng các kỹ thuật chính quy hóa (Regularization Techniques) giảm trọng số (weight decay) và drop out.

- Giảm trọng số (Weight Decay): Còn được gọi là chính quy hóa L2, hoạt động bằng cách thêm penalties vào các trọng số lớn hơn. Điều này giữ cho kích thước của

các trọng số nhỏ (khi các trọng số nhỏ, mỗi tính năng đóng góp ít hơn vào quyết định chung của mô hình, điều này làm cho mô hình ít nhạy cảm hơn với các biến động trong dữ liệu đầu vào). Nếu không giảm trọng số, trọng số có thể tăng theo cấp số nhân, dẫn đến tình trạng Overfitting.

- Drop out:



Hình 2.17: Dropout: (a) full network; (b) network after dropout

Kỹ thuật điều chỉnh này hoạt động bằng cách bỏ qua ngẫu nhiên một số neuron nhất định trong quá trình đào tạo, trong quá trình truyền tới và truyền lui. Tỷ lệ bỏ qua kiểm soát khả năng một neuron nhất định sẽ bị loại bỏ. Do đó, đối với mỗi đợt đào tạo, một tập hợp con neuron khác nhau sẽ được kích hoạt, dẫn đến quá trình học tập mạnh mẽ hơn.

Residual Connections

Mô hình Xception có một số kết nối bỏ qua trong toàn bộ kiến trúc của nó.

Khi đào tạo một Mạng nơ-ron rất sâu, các gradient được sử dụng trong quá trình đào tạo để cập nhật trọng số trở nên nhỏ và thậm chí đôi khi biến mất. Đây là một vấn đề lớn mà tất cả các mô hình Deep Learning đều phải đối mặt. Để khắc phục điều này, các nhà nghiên cứu đã đưa ra các kết nối còn lại trong bài báo của họ vào năm 2016 về mô hình ResNet.

Kết nối còn lại (Residual Connections), còn được gọi là kết nối bỏ qua (Skip Connections), hoạt động bằng cách cung cấp kết nối giữa các lớp trước đó trong mạng với các lớp sâu

hơn hoặc cuối cùng trong mạng. Các kết nối này giúp dòng chảy của các gradient mà không bị mất đi, vì chúng bỏ qua các lớp trung gian (the intermediate layers).

Khi sử dụng Residual Learning, các lớp học cách ước tính sự khác biệt (hoặc δ) giữa đầu vào và đầu ra, do đó, hàm ban đầu $H(x)$ trở thành $H(x) = F(x) + x$.

Lợi ích của Residual Connections:

- **Mạng sâu hơn (Deeper Networks):** Cho phép đào tạo các mạng sâu hơn nhiều.
- **Luồng gradient được cải thiện (Improved Gradient Flow):** Bằng cách cung cấp đường dẫn trực tiếp để gradient chảy ngược về các lớp trước đó, chúng tôi giải quyết được vanishing gradient .
- **Hiệu suất tốt hơn (Better Performance).**

Ngày nay, ResNet là một thành phần tiêu chuẩn trong kiến trúc học sâu.

2.6.4 Hiệu suất và Điểm chuẩn

Bài báo gốc về mô hình Xception sử dụng hai tập dữ liệu khác nhau: ImageNet và JFT. ImageNet là một tập dữ liệu phổ biến, bao gồm 15 triệu hình ảnh được gắn nhãn với 20.000 danh mục. Kiểm tra sử dụng một tập hợp con của ImageNet chứa khoảng 1,2 triệu hình ảnh đào tạo và 1.000 danh mục.

JFT là một tập dữ liệu lớn bao gồm hơn 350 triệu hình ảnh có độ phân giải cao được chú thích bằng nhãn của 17.000 lớp.

Nhóm so sánh mô hình Xception với Inception V3 do số lượng tham số tương tự. Điều này đảm bảo rằng bất kỳ sự khác biệt về hiệu suất nào giữa hai mô hình đều là kết quả của hiệu quả kiến trúc chứ không phải kích thước của nó.

Kết quả thu được cho ImageNet cho thấy sự khác biệt nhỏ giữa hai mô hình, tuy nhiên với tập dữ liệu lớn hơn như JFT, mô hình Xception cho thấy sự cải thiện tương đối 4,3%. Hơn nữa, mô hình Xception vượt trội hơn các mô hình ResNet-152 và VGG-16 .

Bảng dưới so sánh Xception với các model trước đó trên tập dữ liệu Imagenet.

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	0.790	0.945

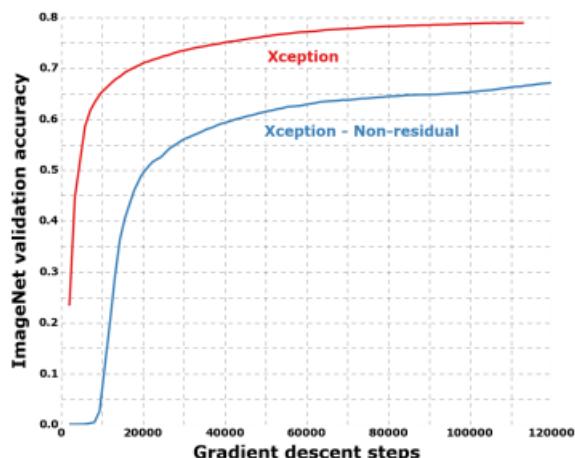
Hình 2.18: Bảng so sánh Xception với các model khác

Xception cũng thể hiện kết quả vượt trội so với Inception V3.

FastEval14k MAP@100	
Inception V3 - no FC layers	6.36
Xception - no FC layers	6.70
Inception V3 with FC layers	6.50
Xception with FC layers	6.78

Hình 2.19: Bảng so sánh Xception với Inception V3

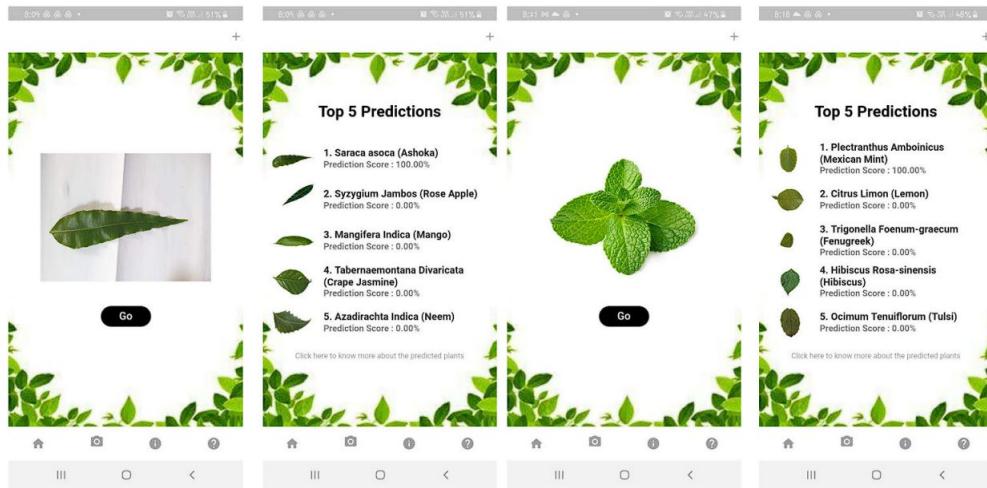
Việc sử dụng Residual connections làm cho độ chính xác của Xception tăng đáng kể.



Hình 2.20: Residual connection in Xception

2.6.5 Ứng dụng của mô hình Xception

Nhận diện kế hoạch



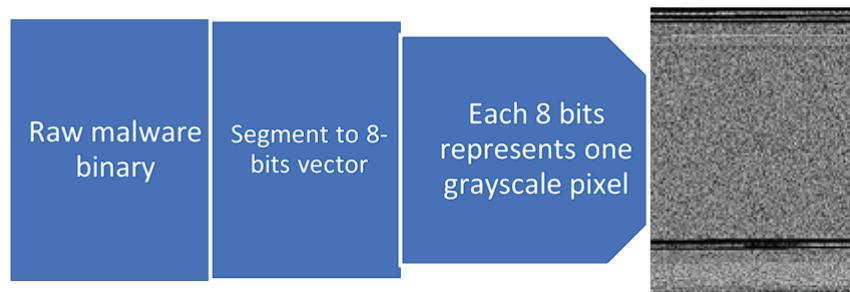
Hình 2.21: Ảnh chụp màn hình hình ảnh thảo mộc hình dạng đầu vào và kết quả dự đoán trong ứng dụng di động HerbSnap

Các nhà nghiên cứu đã phát triển ứng dụng DeepHerb, một hệ thống tự động nhận dạng cây thuốc bằng các kỹ thuật học sâu. Bộ dữ liệu DeepHerb bao gồm 2515 hình ảnh lá từ 40 loài thảo mộc Ấn Độ.

Các nhà nghiên cứu đã sử dụng nhiều kiến trúc mạng nơ-ron tích chập (CNN) được đào tạo trước như VGG16, VGG19, InceptionV3 và Xception. Mô hình có hiệu suất tốt nhất là mô hình Xception đạt độ chính xác 97,5%. Ứng dụng di động HerbSnap cung cấp khả năng nhận dạng thảo mộc với thời gian dự đoán là 1 giây.

Phát hiện phần mềm độc hại

Các nhà nghiên cứu đã sử dụng Xception Network để phân loại phần mềm độc hại bằng cách sử dụng học chuyển giao. Đầu tiên, họ chuyển đổi các tệp phần mềm độc hại thành hình ảnh thang độ xám và sau đó phân loại chúng bằng mô hình Xception được đào tạo trước được tinh chỉnh để phát hiện phần mềm độc hại. Hai tập dữ liệu đã được sử dụng cho nhiệm vụ này: Malimg Dataset (9.339 hình ảnh thang độ xám phần mềm độc hại, 25 họ phần mềm độc hại) và Microsoft Malware Dataset (10.868 hình ảnh thang độ



Hình 2.22: Hình ảnh phần mềm độc hại thang độ xám

xám phần mềm độc hại, 10.873 mẫu thử nghiệm, 9 họ phần mềm độc hại).

Mô hình Xception thu được đạt độ chính xác (99,04% trên Malimg, 99,17% trên Microsoft) so với các phương pháp khác như VGG16.

Các nhà nghiên cứu cũng cải thiện độ chính xác hơn nữa bằng cách tạo ra một Mô hình tổng hợp kết hợp các kết quả dự đoán từ hai loại tệp phần mềm độc hại (.asm và .bytes). Mô hình tổng hợp kết quả đạt được độ chính xác là 99,94%.

Malimg Dataset Validation Accuracy	
<i>Method</i>	<i>Validation Accuracy</i>
M-CNN (VGG16) [9]	98.52%
GIST + SVM [9]	92.23%
KNN [8]	97.18%
Xception	99.03%

Hình 2.23: Độ chính xác xác thực của các phương pháp khác nhau trên tập dữ liệu Malimg

Phát hiện bệnh lá

Một nghiên cứu đã phân tích các bệnh khác nhau được tìm thấy ở đào và phân loại của chúng bằng các mô hình học sâu khác nhau. Các mô hình học sâu bao gồm MobileNet , ResNet , AlexNet , v.v. Trong số tất cả các mô hình này, mô hình Xception với chính quy hóa L2M đạt điểm cao nhất là 93,85%, khiến nó trở thành mô hình hiệu quả nhất trong nghiên cứu đó để phân loại bệnh đào.

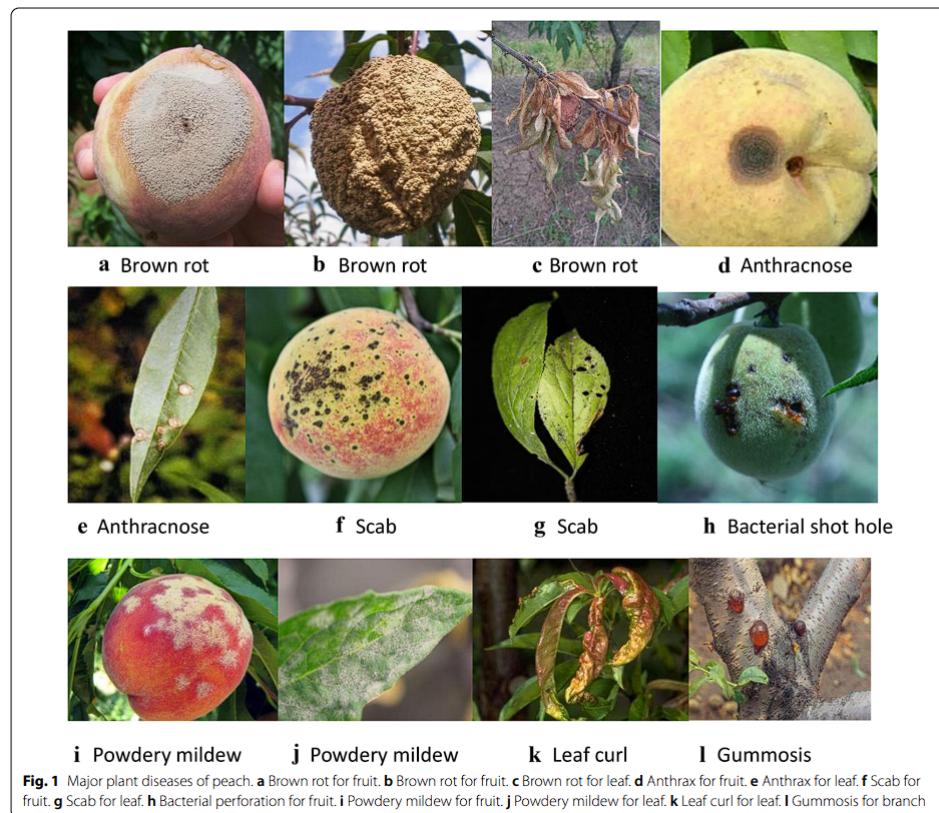


Fig. 1 Major plant diseases of peach. **a** Brown rot for fruit. **b** Brown rot for fruit. **c** Brown rot for leaf. **d** Anthrax for fruit. **e** Anthracnose for leaf. **f** Scab for fruit. **g** Scab for leaf. **h** Bacterial perforation for fruit. **i** Powdery mildew for fruit. **j** Powdery mildew for leaf. **k** Leaf curl for leaf. **l** Gummosis for branch

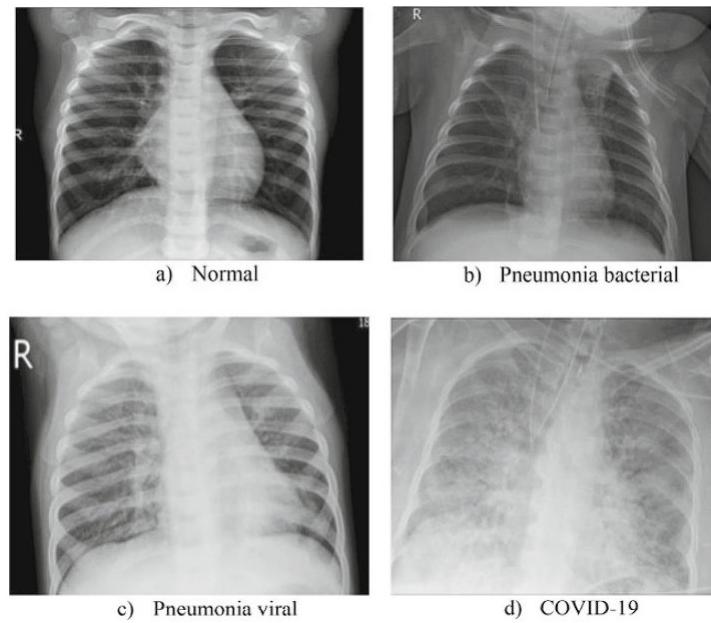
Hình 2.24: Các bệnh thực vật chính

Network	Validation accuracy(L2) (%)	Validation accuracy (L2M) (%)	Change
AlexNet	56.31	57.31	1.00% (+)
ResNet50	78.64	79.34	0.7% (+)
Xception	92.23	93.85	1.62% (+)
SENet154	62.14	62.84	0.7% (+)
DenseNet169	82.90	80.58	2.32% (-)
HRNet-w48	78.13	78.00	0.13% (-)
MobileNetV3	65.69	66.01	0.32% (+)

Hình 2.25: So sánh độ chính xác xác thực của bảy mô hình với L2 và L2M

Phát hiện COVID-19

Các nhà nghiên cứu đã phát triển một mô hình dựa trên Xception cải tiến bằng cách sử dụng các kỹ thuật toán di truyền để tối ưu hóa mạng. Mô hình Xception kết quả đạt được kết quả có độ chính xác cao trên hình ảnh X-quang—99,6% cho hai điểm



Hình 2.26: Hình ảnh từ các lớp khác nhau

cass và 98,9% cho ba lớp, vượt trội đáng kể so với các phương pháp học sâu khác (như DenseNet169, HRNet-w48 và AlexNet) được sử dụng trong nghiên cứu.

Model	Number of parameters	Accuracy
VGG19 (Apostolopoulos & Mpesiana, 2020)	143 million	0.935
Xception (Apostolopoulos & Mpesiana, 2020)	33 million	0.929
Darknet (Ozturk et al., 2020)	1.1 million	0.870
Coronet (Khan et al., 2020)	33 million	0.896
This study	2 million	0.989

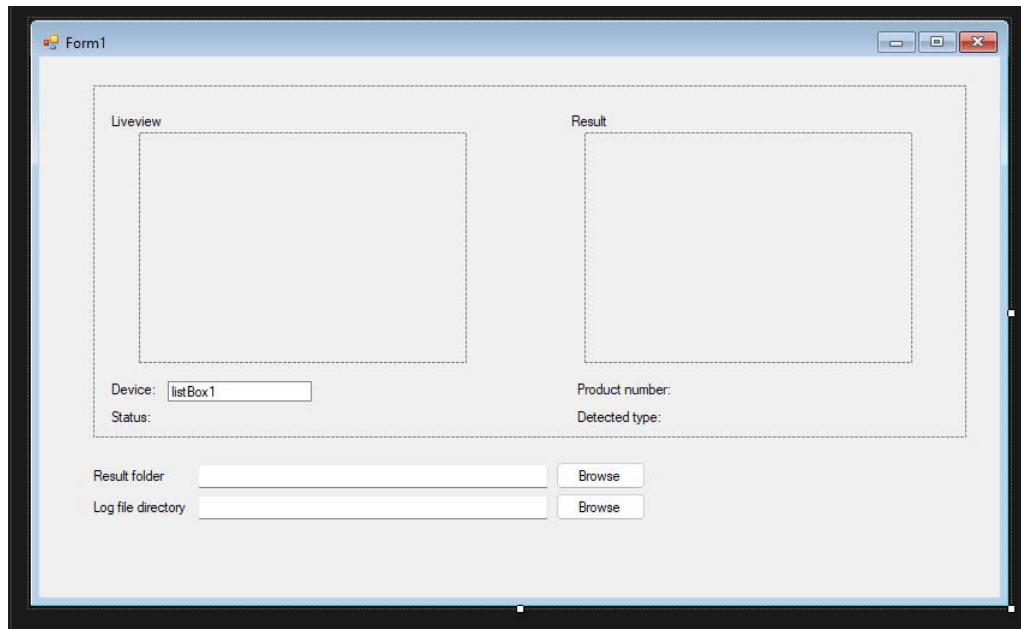
Hình 2.27: So sánh các mô hình cho tập dữ liệu ba lớp

Chương 3

Nội dung thực hiện

Vì thời gian có hạn nên nhóm chỉ làm GUI và Training model

3.1 GUI



Hình 3.1: Giao diện phần mềm

Các chức năng trên GUI:

- **Liveview:** Hiển thị hình ảnh trực tiếp từ thiết bị (camera hoặc nguồn cấp dữ liệu) để người dùng có thể quan sát.

- **Result:** Hiển thị kết quả sau khi xử lý dữ liệu từ "Liveview".
- **Device:** Chứa các thiết bị đầu vào có sẵn.
- **Status:** Hiển thị trạng thái của thiết bị hoặc hệ thống:
 - Connected: Đã kết nối với thiết bị.
 - Disconnected: Chưa kết nối.
 - Processing: Đang xử lý dữ liệu.
 - Error: Có lỗi xảy ra.
- **Product number:** Hiển thị số hiệu hoặc ID của sản phẩm được xử lý hoặc quét.
- **Detected type:** Cho biết kết quả nhận dạng là thân (Body) hay nắp (Lid) và thuộc loại nào sau đây
 - No Defect.
 - Minor Defect.
 - Major Defect.
 - Critical Defect.
- **Result folder:** Chọn thư mục để lưu kết quả đầu ra sau khi xử lý.
- **Log file directory:** Chọn thư mục để lưu trữ file nhật ký (log file) ghi lại thông tin hoạt động của hệ thống.

3.2 DATASET

3.2.1 Lựa chọn tập dữ liệu DATASET

Nhóm đã tìm được một project trên roboflow có sự tương đồng với nhóm tuy số lượng dữ liệu không nhiều nhưng nhóm sẽ sử dụng bộ dữ liệu và tiến hành xử lý lại bộ dữ liệu để training Model. Nguồn: <https://s.pro.vn/CKip>

3.2.2 Xử lý DATASET

Nhóm tiến hành xử lý dữ liệu chia làm 2 loại:

DATASET training YOLOv8

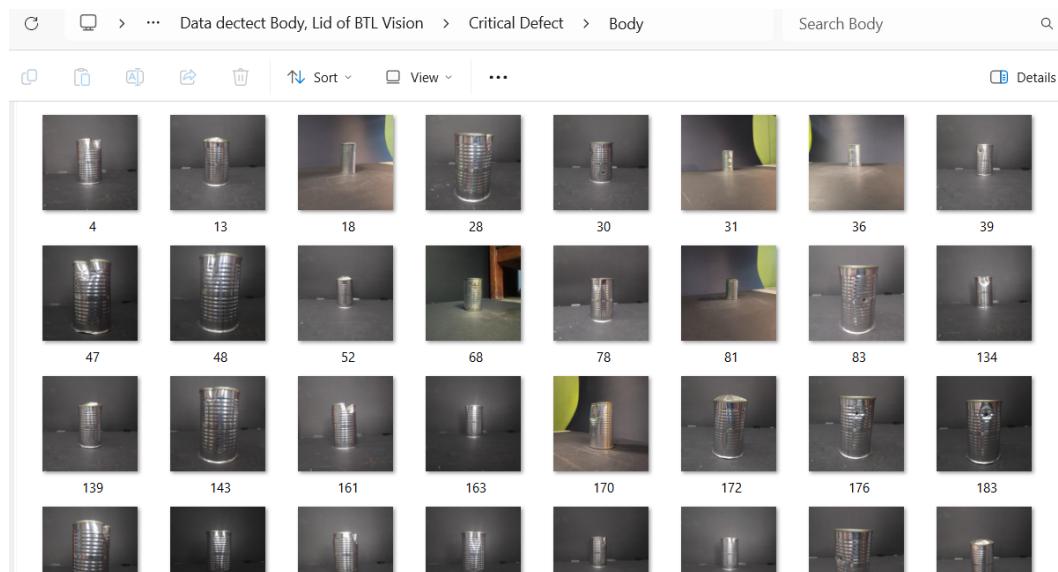
Phục vụ cho việc Detect Lon trong băng chuyền bao gồm cả thân lon và nắp lon. Nhóm tiến hành phân ảnh theo 4 Class hay 4 mức mức độ biến dạng của lon, ở tại mỗi Class nhóm tiếp tục phân thành 2 nhóm là Body và Lid để tiến hành train YOLOv8 trên Roboflow.

📁 Critical Defect	11/15/2024 10:21 AM	File folder
📁 Major Defect	11/15/2024 10:19 AM	File folder
📁 Minor Defect	11/15/2024 10:16 AM	File folder
📁 No Defect	11/15/2024 10:11 AM	File folder
📁 Body	11/15/2024 10:21 AM	File folder
📁 Lid	11/15/2024 10:22 AM	File folder

(a) Data train YOLOv8

(b) Phân thành 2 loại trong mỗi Class

Hình 3.2: Xử lý dữ liệu train YOLOv8



Hình 3.3: Hình ảnh khi đã phân (trong 1 class)

Do YOLOv8 chỉ phục vụ cho việc nhận diện lon, vẽ bounding box và chạy real time nên không cần số lượng hình quá nhiều để training giúp tiết kiệm thời gian tính toán. Tiếp theo tiến hành tổng hợp và phân thành 3 tệp test, train, valid theo tỷ lệ 70:20:10 để tiến hành training như hình 3.4

Class	No Defect		Minor Defect		Major Defect		Critical Defect	
	Body	Lid	Body	Lid	Body	Lid	Body	Lid
	50	50	50	33	50	9	50	50
Sum	100		83		59		100	
Total	Body = 200		Lid = 142		Body + Lid = 342			

Bảng 3.1: Bảng thống kê số lượng Data train YOLOv8



Hình 3.4: Data sau khi được tổng hợp và up lên kaggel

DATASET training Xception

Sau khi tiến hành train Model YOLOv8 detect Lon, cần 1 bộ Data đủ lớn để training cho Model Xception trong việc phân loại Lon giữa các Class. Vì giữa các Class độ khác nhau không quá lớn nên cần 1 bộ Data đủ lớn và cân bằng giữa các Class để tránh hiện tượng Overfitting cũng như để Model "học" một cách chính xác nhất. Nhóm phân thành 3 : Train, Valid, Test theo tỷ lệ 70:20:10 để đảm bảo quá trình training một cách tối ưu nhất. Ở trong mỗi folder tiến hành phân thành tệp ảnh riêng, gắn nhãn các ảnh (label) và quan trọng là tạo 1 tệp class.csv.

valid	12/16/2024 10:45 PM	File folder	labels	12/16/2024 10:44 PM	File folder
train	12/16/2024 10:44 PM	File folder	images	12/16/2024 10:44 PM	File folder
test	12/16/2024 10:44 PM	File folder	_Earlier this year		

(a) Data train Xception

(b) Xử lý bên trong mỗi tệp

Hình 3.5: Xử lý dữ liệu train Data Xception

IMG_20221108_114229.jpg.rf.e7553ced1...	10/11/2024 9:42 AM	JPG File	829 KB	IMG_20221108_114237.jpg.rf.0f94babd0...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114231.jpg.rf.88a6dd1ea...	10/11/2024 9:42 AM	JPG File	836 KB	IMG_20221108_114241.jpg.rf.790923813...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114236.jpg.rf.047f02f5be...	10/11/2024 9:42 AM	JPG File	760 KB	IMG_20221108_114242.jpg.rf.e63cbc6f53...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114237.jpg.rf.0f94babd0...	10/11/2024 9:42 AM	JPG File	812 KB	IMG_20221108_114244.jpg.rf.736fa82f6e...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114241.jpg.rf.790923813...	10/11/2024 9:42 AM	JPG File	774 KB	IMG_20221108_114245.jpg.rf.e63cbc6f53...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114242.jpg.rf.e63cbc6f53...	10/11/2024 9:42 AM	JPG File	802 KB	IMG_20221108_114246.jpg.rf.79983979a...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114244.jpg.rf.736fa82f6e...	10/11/2024 9:42 AM	JPG File	799 KB	IMG_20221108_114248.jpg.rf.875b094f58...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114245.jpg.rf.e63cbc6f53...	10/11/2024 9:42 AM	JPG File	758 KB	IMG_20221108_114251.jpg.rf.a17a8dd1e...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114246.jpg.rf.57983897a...	10/11/2024 9:42 AM	JPG File	763 KB	IMG_20221108_114254.jpg.rf.57b06952d...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114248.jpg.rf.875b094f58...	10/11/2024 9:42 AM	JPG File	867 KB	IMG_20221108_114257.jpg.rf.e2de30333...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114251.jpg.rf.a17a8dd1e...	10/11/2024 9:42 AM	JPG File	762 KB	IMG_20221108_114259.jpg.rf.7d59f465...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114254.jpg.rf.57b06952d...	10/11/2024 9:42 AM	JPG File	760 KB	IMG_20221108_114259.jpg.rf.a0d00bf92...	10/11/2024 9:46 AM	Text Document	1 KB
IMG_20221108_114257.jpg.rf.62de30333...	10/11/2024 9:42 AM	JPG File	746 KB	IMG_20221108_114301.jpg.rf.r4b0f6cd41...	10/11/2024 9:46 AM	Text Document	1 KB

(a) images

(b) labels

Hình 3.6: Tôp hợp và label ảnh

.classes - Excel																	
File Home Insert Page Layout Formulas Data Review View Help Torabox Tell me what you want to do																	
Font																	
POSSIBLE DATA LOSS: Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.																	
Don't show again Save As...																	
A1																	
1	Ellesmere	Critical	Def.	Major	Def.	Minor	Def.	No	defect								
2	IMG_2022	1	0	0	0	0	0										
3	IMG_2022	0	1	0	0	0	0										
4	IMG_2022	0	0	0	1	0	0										
5	IMG_2022	0	0	0	1	0	0										
6	IMG_2022	0	1	0	0	0	0										
7	IMG_2022	0	0	0	0	0	1										
8	IMG_2022	0	0	1	0	0	0										
9	IMG_2022	0	1	0	0	0	0										
10	IMG_2022	1	0	0	0	0	0										
11	IMG_2022	0	0	0	0	1	0										
12	IMG_2022	0	0	0	0	0	1										
13	IMG_2022	0	1	0	0	0	0										
14	IMG_2022	0	0	1	0	0	0										
15	IMG_2022	0	1	0	0	0	0										
16	IMG_2022	0	0	0	1	0	0										
17	IMG_2022	0	0	0	0	0	1										
18	IMG_2022	0	0	0	0	0	1										
19	IMG_2022	0	1	0	0	0	0										
20	IMG_2022	1	0	0	0	0	0										
21	IMG_2022	0	0	0	1	0	0										
22	IMG_2022	0	1	0	0	0	0										
23	IMG_2022	0	0	0	1	0	0										
24	IMG_2022	1	0	0	0	0	0										
25	IMG_2022	0	0	0	0	0	1										
26	IMG_2022	1	0	0	0	0	0										
27	IMG_2022	0	0	0	0	0	1										

Hình 3.7: Tạo 1 file class.csv tương ứng

Phải đảm bảo hình ảnh 3 tệp được phân theo tỷ lệ 70:20:10 để quá trình training diễn ra tốt nhất

	Train	Valid	Test
Images	5658	1609	829
Total	8096		

Bảng 3.2: Bảng thống kê Data train Xception

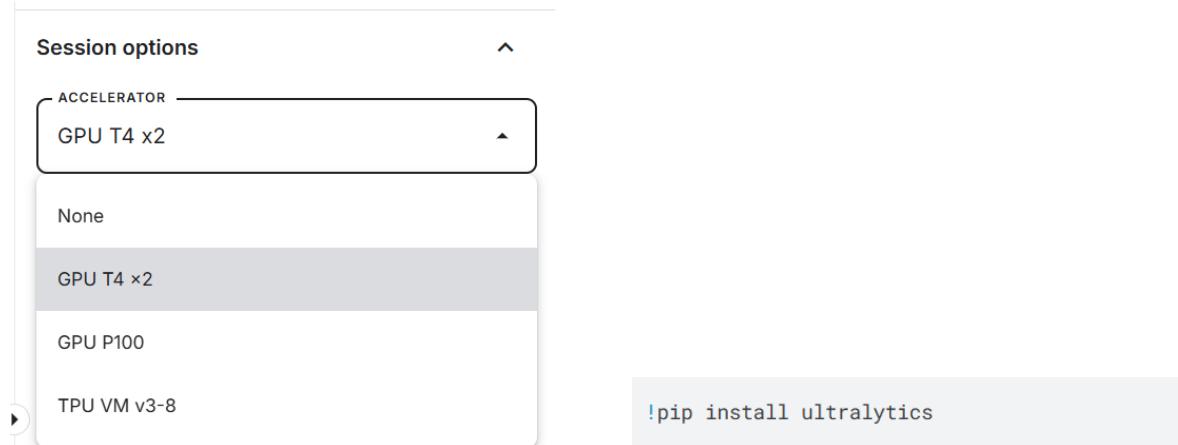
Lưu ý: mỗi tệp train, valid, test đều phải chứa hình ảnh 4 class để Model "học" chính xác và tránh hiện tượng Overfitting như 3.7 có đủ hình ảnh 4 class.

3.3 Training Model

3.3.1 Code Environment

Kaggel là một nền tảng dựa trên đám mây được cung cấp bởi Google cho phép người dùng viết và thực thi mã Python. Những lý do phù hợp cho dự án là:

- GPU miễn phí: Kaggel cung cấp truy cập vào tài nguyên GPU miễn phí để chạy các tác vụ học máy (ML) và học sâu. Điều này đặc biệt hữu ích cho các tác vụ yêu cầu tài nguyên tính toán nặng, như việc huấn luyện các mạng nơ-ron sâu.
- Các gói ML phổ biến đã được cài sẵn: Kaggel đi kèm với nhiều thư viện học máy và học sâu phổ biến đã được cài đặt sẵn, bao gồm TensorFlow, PyTorch, scikit-learn, Keras, và nhiều hơn nữa. Điều này làm cho việc bắt đầu làm việc trên các dự án ML trở nên thuận tiện mà không cần phải cài đặt các gói này thủ công.
- Giao diện Notebook: Google Colab cung cấp một giao diện sổ ghi chép tương tự như Jupyter Notebooks, cho phép người dùng viết và thực thi mã trong một môi trường cấu trúc và tương tác. Các sổ ghi chú được tổ chức thành các ô, trong đó mỗi ô có thể chứa mã, văn bản hoặc hình ảnh. Giao diện này giúp người dùng viết và thử nghiệm mã, hình dung dữ liệu và tài liệu công việc của họ trong một môi trường duy nhất.



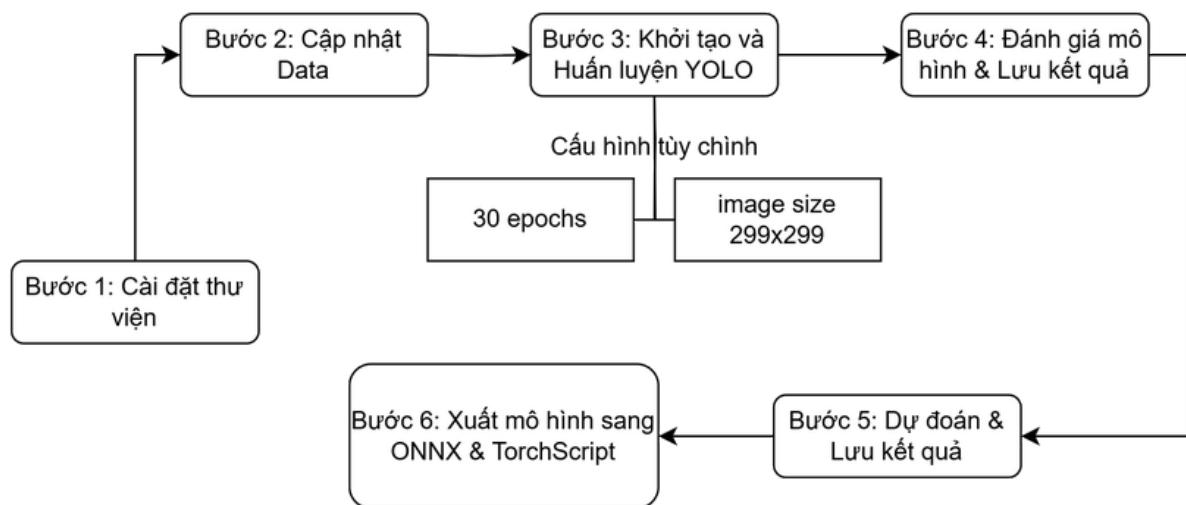
(a) Thiết lập GPU trong Kaggle

(b) Cài đặt các gói thư viện có sẵn

Hình 3.8: Hướng dẫn sử dụng Kaggle

3.3.2 Training YOLOv8

Dưới đây là lưu đồ thuật toán huấn luyện YOLOv8, hình 3.9.



Hình 3.9: YOLO training flowchart

Code:

```
##Environment
import os
import pandas as pd
!pip install ultralytics
```

```

##Training

import yaml
from ultralytics import YOLO


# Step 1: Update data.yaml file
original_yaml_path = "/kaggle/input/can-body-lid/Can-Body-Lid/data.yaml"
updated_yaml_path = "/kaggle/working/data.yaml"

# Load and modify the YAML file
with open(original_yaml_path, "r") as file:
    data = yaml.safe_load(file)

# Update paths in the YAML file
data['train'] = "/kaggle/input/can-body-lid/Can-Body-Lid/train/images"
data['val'] = "/kaggle/input/can-body-lid/Can-Body-Lid/valid/images"
data['test'] = "/kaggle/input/can-body-lid/Can-Body-Lid/test/images"

# Save the updated YAML to /kaggle/working
with open(updated_yaml_path, "w") as file:
    yaml.safe_dump(data, file)

print(f"Updated data.yaml saved to: {updated_yaml_path}")

# Step 2: Train YOLOv8 Model
model = YOLO('yolov8n.pt')

# Train the model with 200 epochs and image size 299x299
model.train(data=updated_yaml_path, epochs=30, imgsz=299,
            project='/kaggle/working', name='yolov8_can_body_lid')

# Step 3: Validate the Model
metrics = model.val()

```

```

# Extract Metrics from the Results
print("\nValidation Metrics:")
results = metrics.results_dict
print(f"mAP@0.5: {results['mAP50']:.4f}")
print(f"mAP@0.5:0.95: {results['mAP50-95']:.4f}")
print(f"Precision: {results['precision']:.4f}")
print(f"Recall: {results['recall']:.4f}")

# Step 4: Predict on the Test Dataset
test_images_path = "/kaggle/input/can-body-lid/Can-Body-Lid/test/images"
test_results = model.predict(source=test_images_path, save=True,
    save_txt=True, project='/kaggle/working',
    name='yolov8_test_predictions')

print("\nTest Results:")
print(f"Number of Predictions: {len(test_results)}")
print("Bounding boxes saved with images in
    '/kaggle/working/yolov8_test_predictions/.'")

# Step 5: Export the Model to ONNX and TorchScript
export_formats = ['onnx', 'torchscript'] # Export formats
for format in export_formats:
    export_path = model.export(format=format, project='/kaggle/working',
        name=f'yolov8_can_body_lid_{format}')
    print(f"Model exported to {format.upper()} format at: {export_path}")

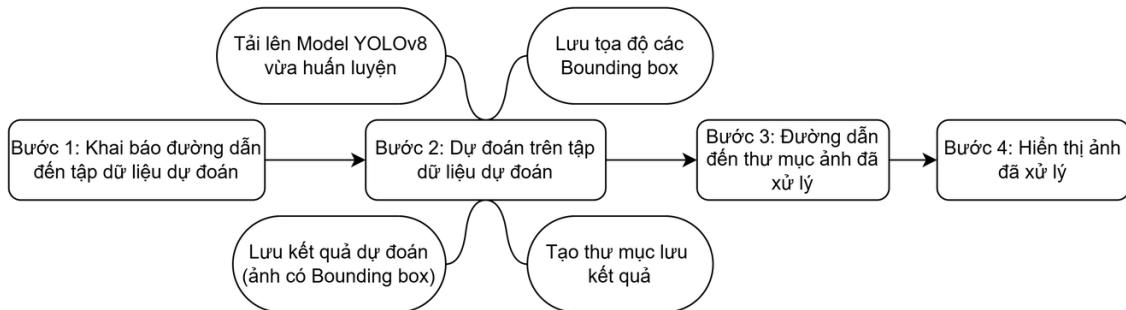
```

Sau khi Training model Kaggel sẽ hiện output là tập hợp các trọng số weight. Tuy nhiên, Kaggel sẽ tự động xóa kết quả trong 40' nếu không tiến hành download, canh thời điểm download tập dữ liệu xuống để sử dụng cho quá trình dự đoán và hiển thị ảnh tiếp theo.

 best.pt 11/20/2024 4:28 PM PT File 6,063 KB

Hình 3.10: Tập dữ liệu sau khi Training YOLOv8

Tiến hành sử dụng tập dữ liệu cho việc dự đoán và vẽ Bounding box. Dưới đây là lưu đồ dự đoán và hiển thị ảnh từ mô hình YOLOv8 vừa huấn luyện, hình 3.11.



Hình 3.11: Dự đoán và hiển thị ảnh từ model YOLO8v đã training

Code:

```

import os
from PIL import Image
import matplotlib.pyplot as plt
from ultralytics import YOLO

# Step 1: Define Test Images Path
test_images_path = "/kaggle/input/can-body-lid/Can-Body-Lid/test/images"

# Step 2: Predict on the Test Dataset
model = YOLO('yolov8n.pt') # Load the trained model (ensure it's trained
                            # before this step)
test_results = model.predict(
    source=test_images_path,
    save=True,
    save_txt=True,
    project='/kaggle/working',
    name='yolov8_test_predictions'
)

# Step 3: Path to the Folder with Processed Images
processed_images_path = "/kaggle/working/yolov8_test_predictions/predict/"

```

```

# Step 4: Display All Images with Bounding Boxes

image_files = [f for f in os.listdir(processed_images_path) if
f.endswith('.jpg')]

print("Displaying test images with bounding boxes:")

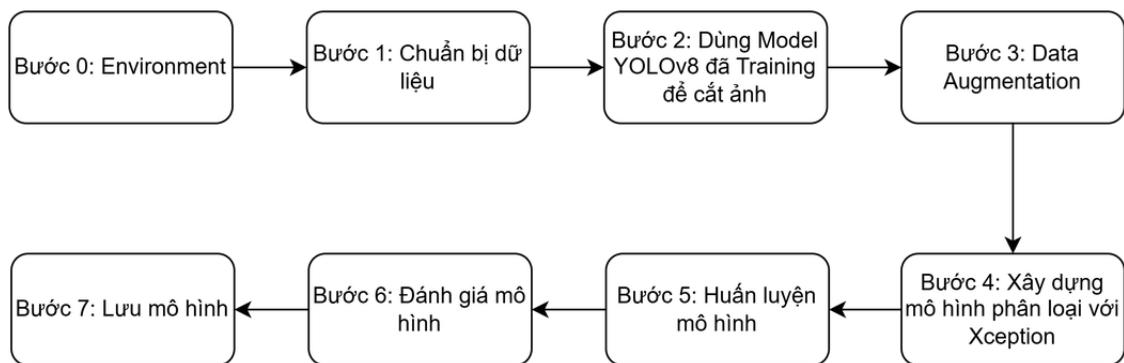
for image_file in image_files:
    # Load the image
    image_path = os.path.join(processed_images_path, image_file)
    img = Image.open(image_path)

    # Display the image with bounding boxes
    plt.figure(figsize=(8, 8))
    plt.imshow(img)
    plt.axis('off')
    plt.title(image_file)
    plt.show()

```

3.3.3 Training Xception

Dưới đây là lưu đồ thuật toán huấn luyện Xception, hình 3.12.



Hình 3.12: Xception training flowchart

Code:

```
!pip install ultralytics

import os
import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import Xception
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
    Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from sklearn.utils import class_weight
from sklearn.metrics import precision_recall_curve, f1_score
from tqdm import tqdm

# Parameters
img_height, img_width = 299, 299
batch_size = 32
l2_lambda = 0.001 # L2 regularization strength

# Load the CSV files
train_csv = '/kaggle/input/candataset/train/_classes.csv'
valid_csv = '/kaggle/input/candataset/valid/_classes.csv'
train_df = pd.read_csv(train_csv)
valid_df = pd.read_csv(valid_csv)

# Remove extra whitespace from column names
train_df.columns = train_df.columns.str.strip()
valid_df.columns = valid_df.columns.str.strip()
```

```

# Columns for the labels
y_columns = ['Critical Defect', 'Major Defect', 'Minor Defect', 'No
defect']

# Load YOLO model for cropping
yolo_model = YOLO('/kaggle/input/can-detection/pytorch/default/1/best.pt')

def crop_image_using_yolo(image_path):
    """
    Detect and crop can from image using YOLOv8
    """
    image = cv2.imread(image_path)
    if image is None:
        return None

    results = yolo_model(image)

    if len(results[0].boxes) > 0:
        box = results[0].boxes[0]
        confidence = float(box.conf)

        if confidence > 0.5:
            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])
            cropped_image = image[y1:y2, x1:x2]

            if cropped_image.size > 0:
                return cv2.resize(cropped_image, (img_width, img_height))

    return None

def process_and_crop_images(df, source_dir):
    """
    Process all images in the dataframe using YOLO detection silently.

```

```

"""
updated_filenames = []
cropped_count = 0
total_count = len(df)

for _, row in df.iterrows():
    image_filename = row['filename']
    image_path = os.path.join(source_dir, image_filename)

    cropped_image = crop_image_using_yolo(image_path)

    if cropped_image is not None:
        cv2.imwrite(image_path, cropped_image)
        cropped_count += 1

    updated_filenames.append(image_filename)

df['filename'] = updated_filenames
return df, cropped_count, total_count

# Process and crop the training and validation datasets
print("Processing training images...")
train_df, train_cropped_count, train_total_count =
    process_and_crop_images(train_df, '/kaggle/input/candataset/train')
print("\nProcessing validation images...")
valid_df, valid_cropped_count, valid_total_count =
    process_and_crop_images(valid_df, '/kaggle/input/candataset/valid')

# Print summary of cropping results
print(f"\nTraining set: Cropped {train_cropped_count} out of
{train_total_count} images")
print(f"Validation set: Cropped {valid_cropped_count} out of
{valid_total_count} images")

```

```

# Data augmentation for training data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    horizontal_flip=True,
    fill_mode='nearest'
)
valid_datagen = ImageDataGenerator(rescale=1./255)

# Prepare generators for training and validation
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    directory='/kaggle/input/candataset/train',
    x_col='filename',
    y_col=y_columns,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='raw'
)
valid_generator = valid_datagen.flow_from_dataframe(
    dataframe=valid_df,
    directory='/kaggle/input/candataset/valid',
    x_col='filename',
    y_col=y_columns,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='raw'
)

# Build the Xception model for classification with regularization
base_model = Xception(weights='imagenet', include_top=False,
    input_shape=(img_height, img_width, 3))
model = Sequential([
    base_model,

```

```

        GlobalAveragePooling2D(),
        BatchNormalization(), # Batch normalization for better training
        stability
        Dense(1024, activation='relu', kernel_regularizer=l2(l2_lambda)), # L2
        regularization applied here
        Dropout(0.5), # Dropout for regularization, helps to prevent
        overfitting
        Dense(4, activation='softmax', kernel_regularizer=l2(l2_lambda)) # L2
        regularization applied to output layer
    )
    base_model.trainable = False

    # Class balancing
    class_weights = class_weight.compute_class_weight('balanced',
        classes=np.unique(train_generator.y), y=train_generator.y)
    class_weight_dict = dict(zip(np.unique(train_generator.y), class_weights))

    # Compile the model with regularization in mind
    model.compile(
        optimizer=Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    # Train the model with class weights
    print("\nTraining with frozen base layers...")
    history = model.fit(
        train_generator,
        epochs=4,
        validation_data=valid_generator,
        class_weight=class_weight_dict,
        verbose=1
    )

```

```

# Fine-tune the model
print("\nFine-tuning the model...")
base_model.trainable = True
model.compile(
    optimizer=Adam(learning_rate=0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

fine_tune_history = model.fit(
    train_generator,
    epochs=3,
    validation_data=valid_generator,
    class_weight=class_weight_dict,
    verbose=1
)

# Evaluate the model
y_pred = model.predict(valid_generator)
precision, recall, thresholds = precision_recall_curve(valid_generator.y,
    y_pred, pos_label=1)
f1_scores = [2 * p * r / (p + r + 1e-8) for p, r in zip(precision,
    recall)]
best_threshold = thresholds[np.argmax(f1_scores)]
print(f"Best threshold: {best_threshold:.2f}")

y_pred_binary = (y_pred[:, 1] > best_threshold).astype(int)
valid_f1 = f1_score(valid_generator.y, y_pred_binary)
print(f"Validation F1-score: {valid_f1:.4f}")

# Save the model and weights
model_save_path = '/kaggle/working/candefect_model.h5'
model.save(model_save_path)
print(f"Model saved at {model_save_path}")

```

```
weights_save_path = '/kaggle/working/candefect_weights.weights.h5'  
model.save_weights(weights_save_path)  
print(f"Weights saved at {weights_save_path}")
```

Khi Training việc lựa chọn mô hình kiến trúc cũng cần quan tâm đến số lớp và số lượng tham số của mô hình để đánh giá độ phức tạp và để có thể tinh chỉnh thay đổi kiến trúc mô đồ cho riêng từng loại dự án. Khi xây dựng mô hình Xception nhóm đã thiết kế theo hình thức:

- Phần đầu của Xception (base model) được tải và train tên Data từ ImageNet và không được huấn luyện lại (frozen). Do đã được train sẵn nên Xception sẽ nhận biết được nhiều góc cạnh khác nhau.
- Phần đầu ra được thêm các lớp (Sequential):
 - **GlobalAveragePooling2D:** Chuyển đổi đặc trưng thành vector đầu ra. Do Max Pooling tập trung vào đặc trưng nổi bật (prominent features) trong vùng giữ lại các giá trị quan trọng nhất, thường là các cạnh, góc, hoặc các đặc trưng mạnh trong ảnh nên vì thế có thể bỏ qua thông tin liên quan từ các giá trị nhỏ hơn ảnh, dẫn đến mất một số thông tin chi tiết. Do đó chọn AveragePooling sẽ tập trung vào toàn bộ thông tin tổng quát trong vùng cũng không nhấn mạnh vào đặc trưng nổi bật mà thay vào đó, làm mềm các giá trị đầu ra.
 - **BatchNormalization:** Giúp cải thiện training stability và tăng tốc độ huấn luyện của mô hình học sâu nhờ:
 - * Giảm hiện tượng "Internal Covariate Shift" làm ổn định quá trình huấn luyện
 - * Cho phép learning rate lớn hơn.
 - * Hạn chế overfitting.
 - * Tăng độ nhạy với các giá trị khởi tạo.
 - * Giảm vấn đề "vanishing/exploding gradients".
 - **Dense với L2 regularization và activation là ReLu:** Sử dụng hàm ReLu (hàm phi tuyến tính) để mạng neuron học được các quan hệ phức tạp hơn và

apply L2 nhằm giảm overfitting bằng cách thêm penalty weight.

- **Dropout:** Ngăn overfitting bằng cách loại bỏ ngẫu nhiên các neuron.
- **Dense với L2 regularization và activation là Softmax:** Do cần phân loại 4 lớp nên sẽ có 4 lớp Dense đầu ra đồng thời sử dụng activation Softmax chủ yếu để cho phân loại đa lớp đồng thời vẫn apply L2 nhằm giảm overfitting bằng cách thêm penalty weight.

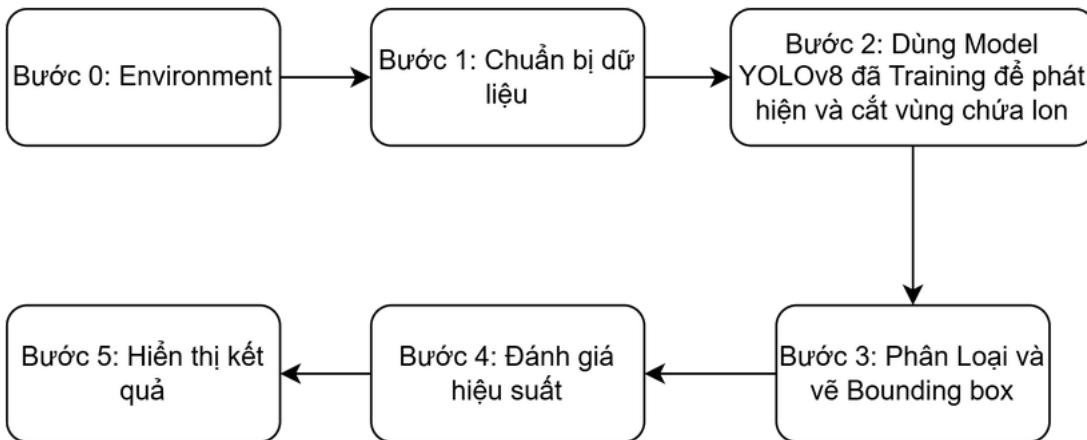
Khi tiến hành Training hiện tượng frozen xảy ra (*base_model.trainable = False*) nhằm tiết kiệm tài nguyên tính toán, tránh Overfitting và đóng băng Xception chỉ huấn luyện các lớp cuối chủ yếu chỉ đặc trưng tổng thể và càng về sau mọi thứ càng trễ nên trùu tượng. Do khi build Model càng về sau càng nhận diện đặc trưng của ảnh hơn và những lớp đầu tiên chỉ tập trung chủ yếu vào các chi tiết như góc nhọn, cạnh, hình tròn,... Tại các lớp đầu Xception thường rất tốt nên cho False để tập trung các lớp cuối để nhận diện đây là cái Lon. Khi frozen cũng sử dụng class weights để cân bằng dữ liệu không đồng đều.

Sau 4 tập train mở khóa lại (unfreeze) (*base_model.trainable = True*) toàn bộ mạng Xception để huấn luyện lại toàn bộ các lớp lúc này sẽ tập trung lại các chi tiết nhỏ của Lon như cạnh Lon, góc,... Mô hình sẽ được "fine-tune" bằng cách mở khóa các lớp cơ sở và huấn luyện lại chúng với một learning rate rất nhỏ (thường nhỏ hơn so với learning rate ban đầu để tránh làm xáo trộn quá nhiều các trọng số đã học được). Sau khi hoàn thành quá trình Training ta có thể sử dụng các bộ output để tiến hành phân loại.

 candefect_weights.weights.h5	11/28/2024 8:24 PM	H5 File	269,207 KB
 candefect_model.h5	11/28/2024 8:24 PM	H5 File	269,197 KB

Hình 3.13: Output of Xception

Dưới đây là lưu đồ kiểm tra và phân loại Lon từ Model Training, hình 3.14.



Hình 3.14: Kiểm tra và đánh giá Model phân loại

Code:

```

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from ultralytics import YOLO
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm

# Paths and constants
test_csv = '/kaggle/input/candataset/test/_classes.csv'
test_dir = '/kaggle/input/candataset/test'
model_path = '/kaggle/working/candefect_model.h5'
yolo_model_path = '/kaggle/input/can-detection/pytorch/default/1/best.pt'

img_height, img_width = 299, 299
y_columns = ['Critical Defect', 'Major Defect', 'Minor Defect', 'No
defect']

```

```

# Load the CSV file
test_df = pd.read_csv(test_csv)
test_df.columns = test_df.columns.str.strip()

# Load trained Xception model
model = load_model(model_path)

# Load YOLO model for object detection
yolo_model = YOLO(yolo_model_path)

# Data generator for test data (apply the same rescale factor as used
# during training)
test_datagen = ImageDataGenerator(rescale=1./255)

# Function to crop image using YOLO
def crop_image_using_yolo(image_path):
    """
    Detect and crop can from image using YOLOv8.
    Resizes the cropped region to the desired input size for the
    classifier.
    """
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Image at {image_path} not found.")
        return None, None

    # Run YOLO prediction to find objects in the image
    results = yolo_model(image)

    if len(results[0].boxes) > 0:
        # Take the first detected box
        box = results[0].boxes[0]
        confidence = float(box.conf)

```

```

if confidence > 0.5:

    # Extract bounding box coordinates
    x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
    x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

    # Crop the image using bounding box coordinates
    cropped_image = image[y1:y2, x1:x2]

    if cropped_image.size > 0:

        # Resize cropped image to match model input size
        cropped_image = cv2.resize(cropped_image, (img_width,
                                                    img_height))

        return cropped_image, (x1, y1, x2, y2)

    # If no valid detection, return None
    print(f"No valid detection for {image_path}")
    return None, None

# Predict and draw bounding boxes
def predict_and_draw_bboxes(test_df, source_dir, model):

    """
    Predict defects using the Xception model and draw bounding boxes.
    """

    true_labels = []
    predicted_labels = []
    incorrect_indices = []
    correct_indices = []

    with tqdm(total=len(test_df), desc="Processing test images") as pbar:
        for idx, row in test_df.iterrows():

            image_filename = row['filename']
            true_label = row[y_columns].values # Ground truth labels
            image_path = os.path.join(source_dir, image_filename)

```

```

# Crop the image using YOLO
cropped_image, bbox_coords = crop_image_using_yolo(image_path)

if cropped_image is not None:
    # Rescale the image after resizing (value range 0-1)
    cropped_image = cropped_image / 255.0
    cropped_image = np.expand_dims(cropped_image, axis=0)

    # Predict using the trained Xception model
    prediction = model.predict(cropped_image)
    predicted_label = np.argmax(prediction, axis=1)[0]
    true_label_idx = np.argmax(true_label)

    # Record true and predicted labels
    true_labels.append(true_label_idx)
    predicted_labels.append(predicted_label)

    # Determine correct or incorrect prediction
    if true_label_idx == predicted_label:
        correct_indices.append(idx)
    else:
        incorrect_indices.append(idx)

    # Draw bounding box on the original image if available
    if bbox_coords is not None:
        x1, y1, x2, y2 = bbox_coords
        image = cv2.imread(image_path)
        cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
        label_text = f"Pred: {y_columns[predicted_label]}, True: {y_columns[true_label_idx]}"
        cv2.putText(image, label_text, (x1, y1 - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Save the image with bounding box if needed

```

```

        output_path = f"/kaggle/working/results/{image_filename}"
        cv2.imwrite(output_path, image)

pbar.update(1)

return true_labels, predicted_labels, correct_indices,
incorrect_indices

# Process test images and make predictions
print("Processing test images and making predictions...")
true_labels, predicted_labels, correct_indices, incorrect_indices =
predict_and_draw_bboxes(test_df, test_dir, model)

# Calculate accuracy and other metrics
accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Accuracy on the test set: {accuracy:.4f}")

# Generate a classification report
print("\nClassification Report:")
print(classification_report(true_labels, predicted_labels,
target_names=y_columns))

# Function to plot images
def plot_images(indices, title, test_df, directory):
    plt.figure(figsize=(16, 16))
    for i, idx in enumerate(indices[:9]):
        filename = test_df.iloc[idx]['filename']
        image_path = os.path.join(directory, filename)
        image = cv2.imread(image_path)
        if image is not None:
            plt.subplot(3, 3, i + 1)
            plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
            plt.title(f"{title}\n{filename}")
            plt.axis('off')

```

```

plt.tight_layout()
plt.show()

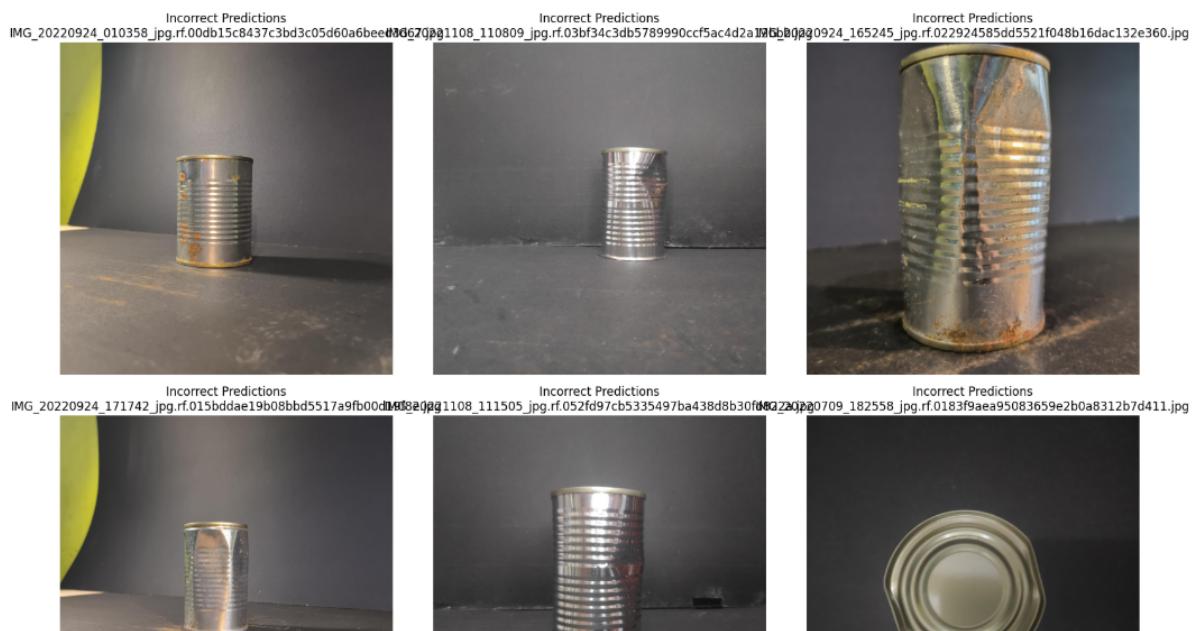
# Plot incorrect predictions
print("\nPlotting incorrect predictions...")
plot_images(incorrect_indices, "Incorrect Predictions", test_df, test_dir)

# Plot correct predictions
print("\nPlotting correct predictions...")
plot_images(correct_indices, "Correct Predictions", test_df, test_dir)

```

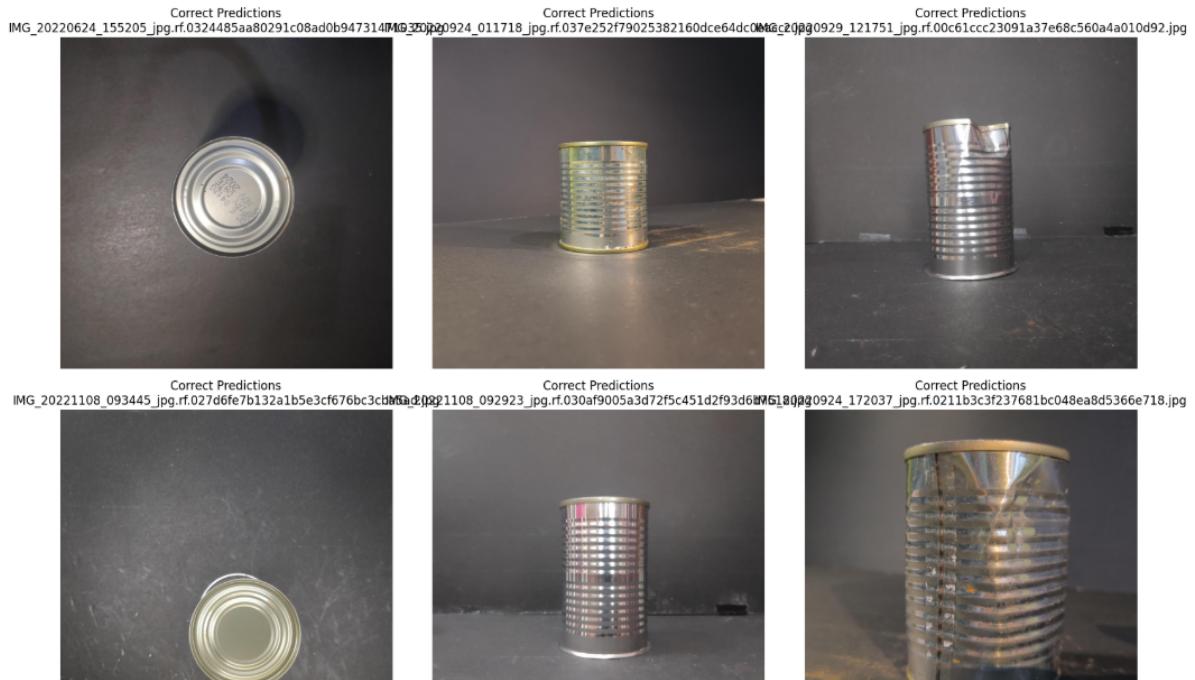
Khi run xong ta sẽ thu được plot của những predict incorrect và correct.

Plotting incorrect predictions...



Hình 3.15: Plot incorrect predictions

Plotting correct predictions...



Hình 3.16: Plot correct predictions

3.3.4 Real time và thực tế

Sau khi trainning và kiểm tra model, tiến hành code real time (đã hỗ trợ sẵn ở YOLOv8) và xuất output (Bounding Box, Body hay Lid, Thuộc lớp nào) trong realtime để cho người dùng dễ dàng phân biệt kết quả.

```
import torch
import cv2
import numpy as np
from ultralytics import YOLO
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array

# Load the YOLO detection model
model_path = '..\\BTL Vision\\best.pt'
detection_model = YOLO(model_path)

# Load the defect classification model
```

```

classification_model = load_model('..\BTL Vision\candefect_model.h5')

# Define classification classes
classes = ['Critical Defect', 'Major Defect', 'Minor Defect', 'No Defect']

# Start the webcam (or any connected camera)
video_capture = cv2.VideoCapture(0) # Use default camera (0 is the camera
ID)

if not video_capture.isOpened():
    print("Unable to open the camera")
    exit()

# Process the video stream
while True:
    ret, frame = video_capture.read() # Read each frame
    if not ret:
        print("Unable to read frames from the camera")
        break

    # Resize the frame to improve processing speed
    scale_percent = 50 # Reduce frame size by 50%
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized_frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)

    # Perform object detection on the frame
    results = detection_model(resized_frame)

    # Process each detected object
    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy().astype(int) # Bounding box
        coordinates

```

```

classes_detected = result.boxes.cls.cpu().numpy().astype(int) #

Detected classes

confidences = result.boxes.conf.cpu().numpy() # Confidence scores

for box, cls, conf in zip(boxes, classes_detected, confidences):
    x1, y1, x2, y2 = box

    # Extract the detected object from the frame
    detected_object = resized_frame[y1:y2, x1:x2]

    # Classify the object if it has a valid size
    if detected_object.size > 0:
        # Resize the object to 299x299 to fit the classification
        # model
        resized_object = cv2.resize(detected_object, (299, 299))
        processed_object = img_to_array(resized_object)
        processed_object = np.expand_dims(processed_object, axis=0)
        processed_object = processed_object / 255.0 # Normalize
        # pixel values

        # Classify the object
        prediction = classification_model.predict(processed_object)
        class_index = np.argmax(prediction) # Class with the
        # highest probability
        defect_class = classes[class_index] # Class name
        confidence_score = prediction[0][class_index] # Probability
        # of the class

        # Create a label with detection and classification results
        label = f'{detection_model.names[cls]} - {defect_class}'
        # {confidence_score:.2f}'

    else:
        # If classification fails, show detection results only
        label = f'{detection_model.names[cls]} {conf:.2f}'
```

```

# Draw the bounding box
cv2.rectangle(resized_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Add the label to the frame
cv2.putText(resized_frame, label, (x1, y1 - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

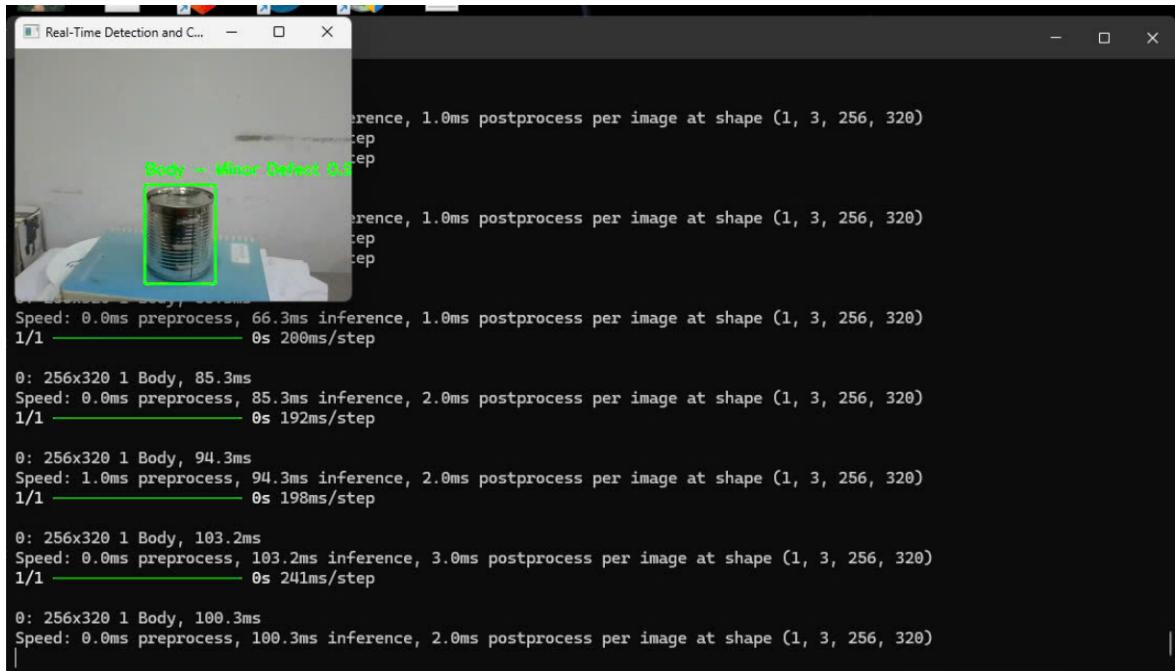
# Display the processed frame
cv2.imshow('Real-Time Detection and Classification', resized_frame)

# Exit the program when the 'q' key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release resources
video_capture.release()
cv2.destroyAllWindows()

```

Tiến hành kiểm tra với camera rời thực tế và ghi nhận kết quả, hình 3.17



Hình 3.17: Real time thực tế

3.4 Đánh giá hiệu suất

3.4.1 Model YOLOv8

Accuracy là một chỉ số quan trọng được sử dụng để đánh giá hiệu suất của mô hình học máy (machine learning). Nó cho biết tỷ lệ dự đoán chính xác so với tổng số dự đoán mà mô hình thực hiện. Accuracy thường được sử dụng trong các bài toán phân loại. Công thức tính Accuracy:

$$Accuracy = \frac{\text{Số lượng dự đoán đúng}}{\text{Tổng số dự đoán}}$$

Lưu ý khi sử dụng Accuracy:

- **Mất cân bằng dữ liệu (Imbalanced Data):** Nếu dữ liệu mất cân bằng (ví dụ: một lớp chiếm 95%, lớp còn lại chiếm 5%), thì một mô hình chỉ cần dự đoán tất cả là lớp chiếm đa số cũng đạt Accuracy cao, nhưng thực tế mô hình không hiệu quả. Trong trường hợp này, bạn nên sử dụng các chỉ số khác như Precision, Recall, F1-score, hoặc ROC-AUC.
- **Không phản ánh chi tiết:** Accuracy không cung cấp thông tin chi tiết về các lỗi của mô hình (False Positive và False Negative).

Kết quả thu được khi training Model YOLOv8:

```
Class      Images  Instances      Box(P)      R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:00<00:00,  5.25it/s]

      all      51      51      0.997      1      0.995      0.983
      Body     35      35      0.998      1      0.995      0.98
      Lid      16      16      0.996      1      0.995      0.986
Speed: 0.3ms preprocess, 6.1ms inference, 0.0ms loss, 2.5ms postprocess per image
Results saved to /kaggle/working/yolov8_can_body_lid2
```

Hình 3.18: Accuracy of YOLOv8

Với Accuracy = 99.7%, cụ thể Body = 99.8% và Lid = 99.6% đánh giá một cách tổng quan thì Model đang cho thấy độ chính xác trong việc nhận diện lon, crop và vẽ Bounding box.

3.4.2 Model Xception với Validation

Tiến hành kết hợp YOLOv8 vào training tập dữ liệu cho Xception nhóm ghi nhận được kết quả:

```
- val_loss: 0.8065
Epoch 10/10
177/177 444s 2s/step - accuracy: 0.9823 - loss: 0.7598 - val_accuracy: 0.9695
- val_loss: 0.7959
51/51 89s 2s/step - accuracy: 0.9723 - loss: 0.7865

Final validation accuracy: 0.9695
Model saved at /kaggle/working/candefect_model.h5
Weights saved at /kaggle/working/candefect_weights.weights.h5
```

Hình 3.19: Accuracy of Xception

Qua đó có thể thấy Model đang thực hiện rất tốt bài toán Classification với Accuracy = 96.95%. Có thể tiến hành đem Model đi predict với tập test.

3.4.3 Model Xception với Test

Precision (Độ chính xác) là tỷ lệ số lượng dự đoán đúng của một lớp nhất định so với tổng số lượng dự đoán vào lớp đó. Nói cách khác, precision cho biết phần trăm các kết quả dương tính được dự đoán mà thực sự là dương tính. Precision được tính bằng công thức:

$$Precision = \frac{TP}{TP + FP}$$

trong đó TP là số lượng true positives và FP là số lượng false positives.

Recall (Độ nhạy) còn được gọi là Sensitivity hoặc True Positive Rate, là tỷ lệ số lượng dự đoán đúng của một lớp nhất định so với tổng số trường hợp thực tế của lớp đó. Recall cho ta biết một mô hình có khả năng nhận diện bao nhiêu phần trăm các trường hợp dương tính. Công thức tính Recall là:

$$Recall = \frac{TP}{TP + FN}$$

trong đó FN là số lượng false negatives.

F1 Score là trung bình điều hòa của Precision và Recall. Nó được sử dụng khi bạn cần cân bằng giữa Precision và Recall, nhất là khi có sự không đồng đều trong phân phối

lớp (class imbalances). F1 Score được tính bằng công thức:

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Ta có được số liệu sau:

```
Accuracy on the test set: 0.7045
```

```
Classification Report:
```

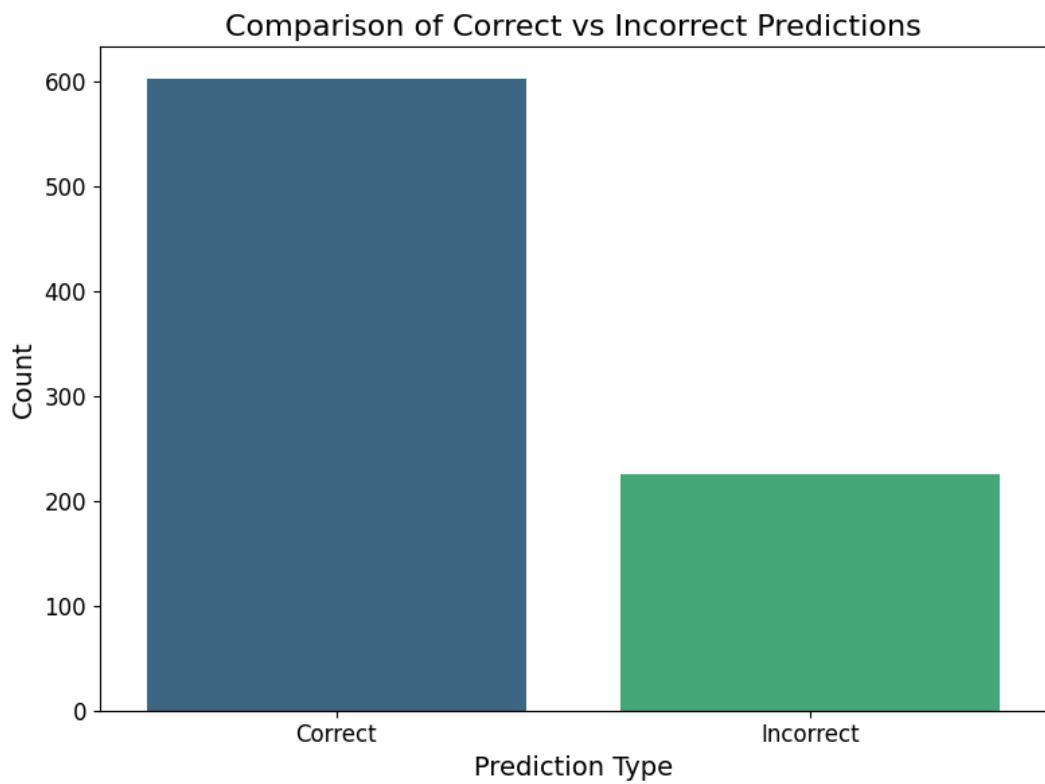
	precision	recall	f1-score	support
Critical Defect	0.63	0.93	0.75	203
Major Defect	0.68	0.78	0.73	200
Minor Defect	0.70	0.29	0.41	227
No defect	0.83	0.88	0.86	199
accuracy			0.70	829
macro avg	0.71	0.72	0.69	829
weighted avg	0.71	0.70	0.68	829

```
Plotting incorrect predictions...
```

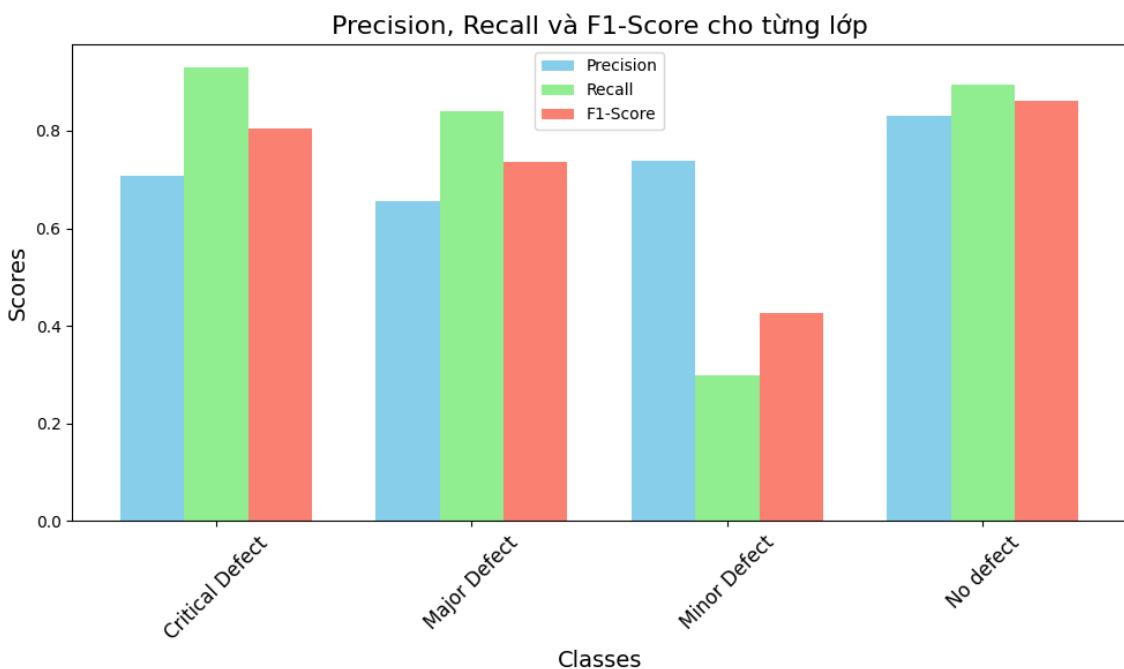
Hình 3.20: Hiệu suất khi Test

Từ số liệu thu được có thể thấy:

- Độ chính xác hay Accuracy của Model với tập Test chưa thực sự cao chỉ 70%. Accuracy của Model khi Test giảm rõ rệt so với khi train Validation. Cho thấy Model còn những hạn chế cần phải khắc phục
- Khả năng nhận diện (Recall) của 3 lớp: Critical, Major, No tương đối cao tuy nhiên phần trăm các kết quả dự đoán đúng (Precision) không quá cao đặc biệt là Major khi precision chỉ 68% và Critical với 63%. Tuy nhiên khả năng nhận diện lớp Minor chưa thực sự tốt với Recall chỉ 29%.



Hình 3.21: Số lượng dự đoán Correct và Incorrect trong tập Test



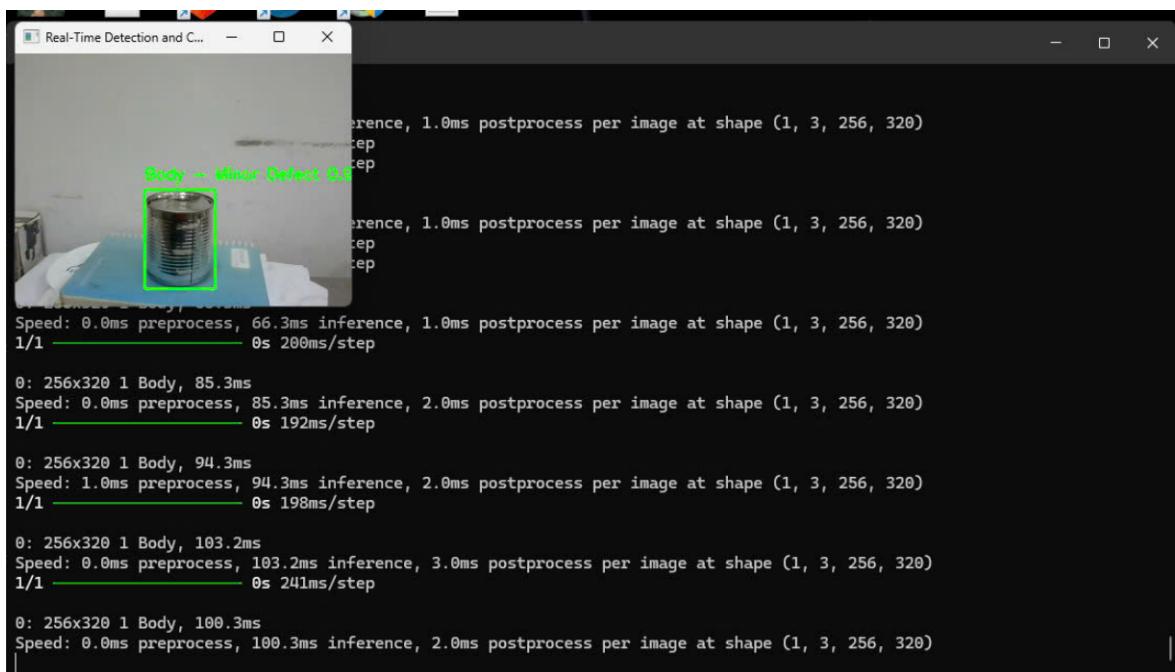
Hình 3.22: Đồ thị đánh giá Model sau training

Chương 4

Ưu điểm và Nhược điểm

4.1 Ưu điểm

- Tốc độ xử lý Real time rất nhanh



Hình 4.1: Tốc độ Real time thực tế

- Phân loại tương đối tốt giữa 4 mức độ biến dạng.
- Có thể nhận ra những biến dạng nhỏ như hình 4.2.
- Có thể sử dụng để phân loại với đầu vào là hình ảnh.



Hình 4.2: Biến dạng nhỏ

- Không cần người canh băng chuyên chụp từng ảnh vì giờ đây bài toán Real time đã được xử lý.
- Có thể theo dõi song song camera và output dự đoán thông qua giao diện GUI, hình 3.1.
- Khả năng phát triển và ứng dụng lớn trong công nghiệp.

4.2 Nhược điểm

- Độ chính xác của Model chưa thực sự cao. Nhóm đã thử Test lại model với bộ test mới và ghi nhận lại kết quả như hình 4.3. Dù đã tiến hành thử lại với Model, qua số liệu trên và thực tế ta có thể thấy Accuracy của Model chưa thực sự cao khi chỉ dao động 70%-80%.

Classification Report:				
	precision	recall	f1-score	support
Critical Defect	0.71	0.93	0.80	203
Major Defect	0.66	0.84	0.74	200
Minor Defect	0.74	0.30	0.43	227
No defect	0.83	0.89	0.86	199
accuracy			0.73	829
macro avg	0.73	0.74	0.71	829
weighted avg	0.73	0.73	0.70	829

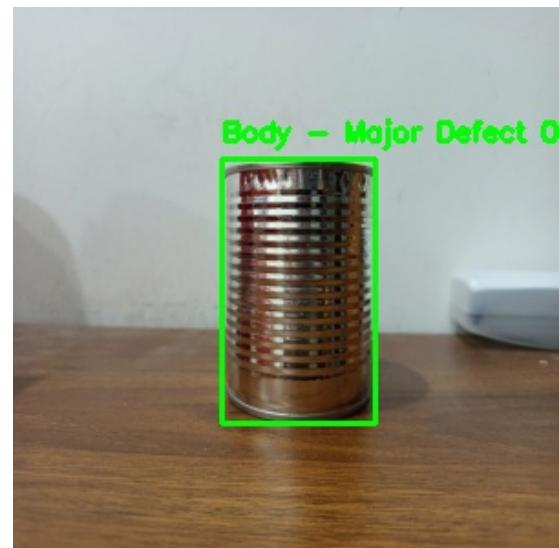
Hình 4.3: Kết quả khi Test lại lần 2

- Qua số liệu train 2 lần với các tập Test khác nhau ta có thể nhận thấy việc phân loại

biến dạng chưa thực sự quá cao. Đặc biệt là phân loại Minor và Major chưa thực sự tốt vì giữa 2 lớp rất khó để phân biệt lẫn nhau.



(a) Minor Defect



(b) Major Defect

Hình 4.4: Sự sai lệch trong dự đoán của Model khi thay đổi góc quay

Chương 5

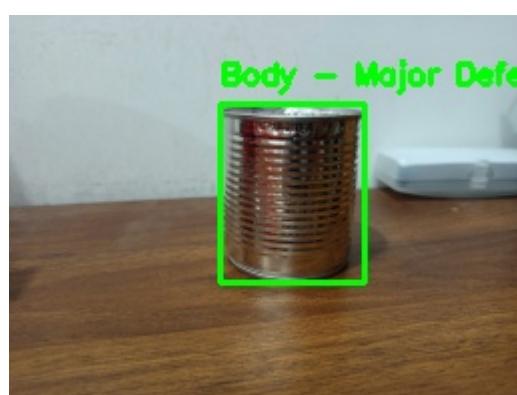
Giải pháp và Kết luận

5.1 Giải pháp

5.1.1 Nguyên nhân

Nguyên nhân của nhược điểm trên có thể là do:

- Data chưa đủ lớn.
- Sự chênh lệch dữ liệu giữa các lớp quá lớn. Ví dụ: Critical:2000, Minor:1500, Major:150,...
- Chưa giải quyết được sự ảnh hưởng của ánh sáng, góc quay đối với Model. Vì khi góc quay làm cho các vằn giữa Lon tối đi Model ngay lập tức hiểu đó là biến dạng và trả hình ảnh output sai khi quan sát trên app.



Hình 5.1: Sự ảnh hưởng của góc máy và ánh sáng

5.1.2 Hướng giải pháp

Có thể tìm cách gia tăng thêm số lượng Data và đảm bảo sự cân bằng giữa các class. Nhưng cách này khó vì yêu cầu là ảnh "thật" trong công nghiệp sẽ rất tốn công chụp hoặc sẽ không thể gia tăng Data nếu như Data khách hàng đưa ra chỉ có vậy.

Model này chủ yếu dựa vào hình dạng của lon để phân loại hình dạng của lon nhưng chưa dựa vào tình trạng của lon như thế nào, đa phần những ảnh mà model dự đoán sai là do lon đã bị rỉ sét nặng, nên phải là Major defect, nhưng model đa phần dự đoán là minor defect có thể là sẽ đúng về mặt hình dạng nhưng về mức độ biến dạng thì chưa thực sự chuẩn lắm trong công nghiệp. Có thể sử dụng Canny để phát hiện những bất thường trên bề mặt lon, Grad-CAM để hiển thị khu vực mà model đang chú ý có thể giúp nhận ra rằng model có thể không chú ý đến khu vực bị rỉ sét,...

Ý tưởng xử lý khác:

- Đầu tiên lọc hết màu của ảnh chỉ lấy hình dạng Lon.
- Cho 1 tham số để Model vote thuộc Class nào.
- Thêm màu vô lại

5.2 Kết luận

Trong nghiên cứu này, nhóm đã xây dựng thành công một hệ thống dựa trên thị giác máy tính để phân loại lon m López theo mức độ biến dạng. Sử dụng các công nghệ hiện đại như phát hiện đối tượng với YOLO và phân loại chi tiết bằng các mô hình học sâu như Xception, khung giải pháp đề xuất đã chứng minh tiềm năng trong việc tự động hóa quy trình kiểm soát chất lượng trong các ngành công nghiệp.

Nhóm đã đạt được các kết quả như sau:

- **Độ chính xác tương đối cao trong nhận diện Lon:** Mô hình YOLO nhận diện và cắt chính xác hình ảnh lon hay nắp lon từ các ảnh đầu vào, đảm bảo chất lượng tiền xử lý cho giai đoạn phân loại.
- **Phân loại hiệu quả:** Mô hình Xception được tinh chỉnh đã đạt độ tin cậy tương đối trong việc phân loại lon vào các nhóm: lỗi nghiêm trọng, lỗi lớn, lỗi nhỏ hoặc

không có lỗi hay phân biệt Lon hay Nắp Lon phù hợp với tiêu chuẩn chất lượng công nghiệp.

- **Tự động hóa và khả năng mở rộng:** Quy trình tự động hóa từ đầu đến cuối có khả năng triển khai thời gian thực, dễ dàng tích hợp vào dây chuyền sản xuất.

Hệ thống giải quyết được nhiều thách thức thực tế trong sản xuất:

- **Tăng sự hiệu quả:** Phân loại tự động giúp giảm sự phụ thuộc vào kiểm tra thủ công, tăng năng suất và giảm chi phí.
- **Cải thiện kiểm soát chất lượng:** Nhận diện chính xác các lon bị lỗi giúp ngăn chặn sản phẩm kém chất lượng đến tay người tiêu dùng, bảo vệ thương hiệu và an toàn khách hàng.
- **Cung cấp thông tin từ dữ liệu:** Hệ thống cung cấp dữ liệu chi tiết, hỗ trợ phân tích nguyên nhân gốc rễ của lỗi, từ đó cải thiện quy trình sản xuất.

Nghiên cứu này mở ra hướng phát triển các ứng dụng thị giác máy tính trong kiểm soát chất lượng công nghiệp, góp phần thúc đẩy tự động hóa và nâng cao hiệu quả sản xuất.

Chương 6

Source

6.1 DATASET

Data training YOLOv8. Link Terabox: <https://terabox.com/s/1trLq19hFG0xVEU118CNAg>

Data Training Xception. Link Terabox: <https://terabox.com/s/1zMG92fFUWoyfxjECk-0qbQ>

6.2 Code

Code Training YOLOv8. Link Github: <https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/Train%20YOLOv8.ipynb>

Code Training Xception. Link Github: <https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/TrainXceptionKaggel.ipynb>

Code Training Xception. Link Github: <https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/Real%20time.py>

Solution For Training Xception. Link Github: <https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/Solution.py>

Github chứa Code. Link Github: [Minh-Nguyen-CEAA](https://github.com/Minh-Nguyen-CEAA)

6.3 Weight

Kết quả training các Model. Link Github:

- Weight Of YOLOv8: <https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/best.pt>.
- Weight Of Xception: https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/candefect_model.h5 và https://github.com/Minh-Nguyen-CEAA/Minh/blob/main/candefect_weights.weights.h5

Danh mục tham khảo

- [1] Viblo AI. *Paper reading | Xception phiên bản nâng cấp của Inception V3*. Truy cập: 2024-11-10. 2023. URL: <https://viblo.asia/p/paper-reading-xception-phien-ban-nang-cap-cua-inception-v3-MkNLr1ZoJgA>.
- [2] Gaudenz Boesch. *Xception Model: Analyzing Depthwise Separable Convolutions*. Truy cập: 2024-11-01. 2024. URL: <https://viso.ai/deep-learning/xception-model/>.
- [3] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. Truy cập: 2024-11-12. 2017. URL: <https://arxiv.org/abs/1610.02357>.
- [4] Francesco Jacob Solawetz. *What is YOLOv8? A Complete Guide*. Truy cập: 2024-11-15. 2024. URL: <https://blog.roboflow.com/what-is-yolov8/>.
- [5] Ultralytics. *Ultralytics YOLOv8*. Truy cập: 2024-11-15. 2024. URL: <https://docs.ultralytics.com/vi/models/yolov8/#overview>.