

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN – ĐIỆN TỬ**

**BỘ MÔN VIỄN THÔNG**

-----oo-----



**ĐÒ ÁN TỐT NGHIỆP ĐẠI HỌC  
ĐỀ TÀI: NHẬN DIỆN VẬT THỂ  
DƯỚI NƯỚC SỬ DỤNG DEEP LEARNING**

**UNDERWATER OBJECT DETECTION USING  
DEEP LEARNING**

**SVTH : Đào Nhựt Nam**

**MSSV : 1914214**

**GVHD : PGS. TS Hà Hoàng Kha**

**TP. HỒ CHÍ MINH, THÁNG 01 NĂM 2025**

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI  
TRƯỜNG ĐẠI HỌC BÁCH KHOA –ĐHQG -HCM**

Cán bộ hướng dẫn Khóa luận tốt nghiệp : PGS.TS. Hà Hoàng Kha  
(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Cán bộ chấm nhận xét 1 : TS. Phạm Quang Thái  
(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Cán bộ chấm nhận xét 2 : ThS. Đinh Quốc Hùng  
(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Khóa luận tốt nghiệp được bảo vệ tại Trường Đại học Bách Khoa, ĐHQG  
Tp.HCM ngày 03 tháng 01 năm 2025

Thành phần Hội đồng đánh giá khóa luận tốt nghiệp gồm:  
(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ khóa luận tốt  
nghiệp)

1. PGS.TS. Hà Hoàng Kha
2. TS. Phạm Quang Thái
3. ThS. Đinh Quốc Hùng

Xác nhận của Chủ tịch Hội đồng đánh giá khóa luận tốt nghiệp và Chủ  
nhiệm Bộ môn sau khi luận văn đã được sửa chữa (nếu có).

**CHỦ TỊCH HỘI ĐỒNG      CHỦ NHIỆM BỘ MÔN VIỄN THÔNG**

-----☆-----

-----☆-----

Số: \_\_\_\_\_ /BKĐT  
Khoa: **Điện - Điện tử**  
Bộ Môn: **Viễn Thông**

## NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. HỌ VÀ TÊN : Đào Nhựt Nam MSSV: 1914214  
2. NGÀNH: **Điện - Điện tử** LỚP : DD19DV5  
CHUYÊN NGÀNH: Kỹ thuật Điện tử - Viễn thông  
3. Đề tài: Nhận diện vật thể dưới nước sử dụng deep learning  
4. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):  
- Nghiên cứu và thu thập dữ liệu vật thể dưới nước  
- Xây dựng phần mềm có khả năng nhận diện các loại vật thể dưới nước  
- Hoàn thành thiết kế phần cứng kết hợp với phần mềm nhận diện  
- Xây dựng trang web stream thời gian thực  
5. Ngày giao nhiệm vụ luận văn: 20/09/2024  
6. Ngày hoàn thành nhiệm vụ: 03/01/2025  
7. Họ và tên người hướng dẫn: Phần hướng dẫn  
PGS. TS Hà Hoàng Kha  
BM Viễn Thông, Khoa Điện - Điện Tử 100%

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp.HCM, ngày 08 tháng 01 năm 2025

**CHỦ NHIỆM BỘ MÔN**

**NGƯỜI HƯỚNG DẪN CHÍNH**

### PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):.....

Đơn vị:.....

Ngày bảo vệ : .....

Điểm tổng kết: .....

Nơi lưu trữ luận văn: .....

## LỜI CẢM ƠN

Lời đầu tiên em xin chân thành cảm ơn thầy Hà Hoàng Kha đã giúp em trong quá trình thực hiện đồ án

Trong quá trình thực hiện đồ án không tránh khỏi một số sai sót. Mong nhận được sự góp ý của thầy để hoàn thiện hơn

Trong quá trình thực hiện đồ án, Thầy đã tận tình hướng dẫn, chỉ bảo em từ những bước đầu tiên cho đến khi hoàn thiện. Những kiến thức quý báu, kinh nghiệm thực tiễn và sự nhiệt tình của thầy đã giúp em vượt qua nhiều khó khăn, thử thách. Mỗi lời khuyên, góp ý của thầy không chỉ giúp em nâng cao chất lượng đồ án mà còn mở ra cho em những hướng đi mới trong tư duy và nghiên cứu.

Sự thành công của đồ án không thể thiếu sự đồng hành và hỗ trợ của thầy.

Tp. Hồ Chí Minh, ngày 08 tháng 01 năm 2025 .

**Sinh viên**

**Đào Nhựt Nam**

## **LỜI CAM ĐOAN**

Tôi tên: Đào Nhựt Nam, là sinh viên cao học chuyên ngành Kỹ thuật Điện tử - Viễn thông, khóa 2019, tại Đại học Quốc gia thành phố Hồ Chí Minh – Trường Đại học Bách Khoa. Tôi xin cam đoan những nội dung sau đều là sự thật:

- (i) Công trình nghiên cứu này hoàn toàn do chính tôi thực hiện;
- (ii) Các tài liệu và trích dẫn trong đồ án này được tham khảo từ các nguồn thực tế, có uy tín và độ chính xác cao;
- (iii) Các số liệu và kết quả của công trình này được tôi tự thực hiện một cách độc lập và trung thực.

*TP. Hồ Chí Minh, ngày 08 tháng 01 năm 2025*

**Sinh viên**

**Đào Nhựt Nam**

# TÓM TẮT ĐỒ ÁN

Trong những năm gần đây, việc phát hiện đối tượng dưới nước đã trở thành một lĩnh vực nghiên cứu quan trọng do ứng dụng rộng rãi trong hải dương học, khảo cổ học dưới nước và an ninh quốc phòng. Tuy nhiên, môi trường dưới nước đặt ra nhiều thách thức như ánh sáng yếu, độ mờ cao và sự biến dạng màu sắc, khiến cho việc phát hiện và nhận dạng đối tượng trở nên phức tạp.

Đồ án này tập trung vào nghiên cứu và phát triển phương pháp phát hiện đối tượng dưới nước bằng cách sử dụng nhiều kiến trúc học sâu khác nhau như YOLO và Faster R-CNN. Chúng tôi triển khai và so sánh hiệu suất của các mô hình này trên tập dữ liệu hình ảnh dưới nước được gán nhãn chính xác. Mục tiêu là tìm ra mô hình tối ưu về độ chính xác và tốc độ xử lý cho ứng dụng thực tế.

Để cải thiện chất lượng dữ liệu đầu vào, chúng tôi áp dụng các kỹ thuật tiền xử lý hình ảnh như cân bằng màu sắc, giảm nhiễu và tăng cường độ tương phản nhằm giảm thiểu ảnh hưởng của môi trường dưới nước. Các mô hình sau đó được huấn luyện với tập dữ liệu đã qua xử lý để nâng cao hiệu suất phát hiện.

Kết quả thực nghiệm cho thấy YOLO đạt được tốc độ xử lý nhanh với độ chính xác cao, phù hợp cho các ứng dụng yêu cầu thời gian thực. Trong khi đó, R-CNN cung cấp độ chính xác cao hơn trong việc nhận dạng và phân loại đối tượng nhưng yêu cầu thời gian xử lý lâu hơn. Việc lựa chọn mô hình phù hợp sẽ tùy thuộc vào yêu cầu cụ thể của từng ứng dụng.

Đồ án này không chỉ cung cấp một giải pháp hiệu quả cho việc phát hiện đối tượng dưới nước mà còn đóng góp vào việc hiểu rõ ưu, nhược điểm của các kiến trúc học sâu hiện đại trong lĩnh vực này. Những kết quả đạt được mở ra hướng nghiên cứu và ứng dụng mới cho các hệ thống quan sát và khai thác dưới nước trong tương lai.

# MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU	9
1.1 Tổng quan	9
1.2 Phạm vi và phương pháp nghiên cứu	10
1.3 Nhiệm vụ luận văn	10
CHƯƠNG 2: LÝ THUYẾT	11
2.1 Giới thiệu về mạng neuron nhân tạo và deep learning	11
2.1.1 Định nghĩa	11
2.1.2 Cấu tạo của một neuron trong Mạng Neural	11
2.1.3 Một số hàm kích hoạt phổ biến (active function)	13
2.2 Giới thiệu về Multi-layers Perceptron (MLP)	14
2.2.1 Kiến trúc của MLP	14
2.2.2 Nguyên lý hoạt động MLP	15
2.3 Convolutional Neural Network (Mạng neuron tích chập)	17
2.3.1 Các thành phần của một CNN	17
2.3.1.1 Convolutional Layer (Lớp tích chập)	17
2.3.1.2 Nonlinear activation function (Hàm phi tuyén)	20
2.3.1.3 Pooling Layer (lớp tổng hợp)	21
2.3.1.4 Fully connected layer (lớp kết nối đầy đủ)	21
2.3.2 Điểm Nổi Bật của CNN	22
2.4 Transfer Learning	23
2.5 Feature Pyramid Networks (FPN)	23
2.6 YOLO11	24
2.6.1 YOLO11 Variants	24
2.6.2 Các thành phần trong kiến trúc YOLO11	27
2.6.2.1 Convolutional Block (Conv Block)	27
2.6.2.2 Bottleneck Block	29
2.6.2.3 C3K2 và C3K Block	31
2.6.2.4 Spatial Pyramid Pooling Fast (SPPF) Block:	32
2.6.2.5 C2PSA	34
2.6.2.6 Detect Block	35
2.6.3 Giải thích Kiến trúc YOLO11	36
2.7 SORT (SIMPLE ONLINE AND REALTIME TRACKING)	38
2.7.1 Giải thuật Hungary	39
2.7.2 Kalman Filter	40
2.7.3 Mô tả SORT	43
2.7.4 Deep SORT	44

2.7.4.1 Kiến trúc trích xuất đặc trưng	45
2.7.4.2 Khoảng cách Mahalanobis và Khoảng cách cosine	45
2.7.4.3 Matching Cascade	47
2.7.4.4 Track Lifecycle	47
2.7.5 Mô tả DEEP SORT	48
2.8 Kết luận chương 2	50
<b>CHƯƠNG 3: THIẾT KẾ VÀ THỰC HIỆN</b>	<b>51</b>
3.1 Yêu cầu thiết kế	51
3.2 Thực hiện	51
3.2.1. Chuẩn bị dataset	51
3.2.2 Train mô hình	53
3.2.3 Triển khai DEEP SORT	57
3.3 Kết luận chương 3	58
<b>CHƯƠNG 4: KẾT QUẢ THỰC HIỆN</b>	<b>60</b>
<b>CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>	<b>64</b>
<b>CHƯƠNG 6: TÀI LIỆU THAM KHẢO</b>	<b>65</b>

## **DANH SÁCH BẢNG SỐ LIỆU**

Bảng 2.1 Model Variant	26
Bảng 3.1 So sánh giữa video tự quay và có sẵn	52
Bảng 3.2 Thông tin về các class	52
Bảng 3.3 Time inference	57

# DANH SÁCH HÌNH ẢNH MINH HỌA

Hình 1.1 Môi trường dưới nước với nhiều nhiễu	9
Hình 1.2 Sự vượt trội của YOLO1 so với các phiên bản trước	10
Hình 2.1 Cấu tạo mạng nơron	11
Hình 2.2 Một số hàm kích hoạt phổ biến	14
Hình 2.3 Mạng nhiều lớp	15
Hình 2.4 Các loại kernel	18
Hình 2.5 Giải thích kernel	19
Hình 2.6 Giải thích Stride	19
Hình 2.7 Giải thích padding	20
Hình 2.8 Hàm phi tuyến	21
Hình 2.9 Giải thích pooling	21
Hình 2.10 Kiến trúc FPN	24
Bảng 2.1 Model Variant	26
Hình 2.11 Conv Block	27
Hình 2.12 Bottleneck Block	29
Hình 2.13 C3K2 Block và C3K Block	31
Hình 2.14 SPPF Block	32
Hình 2.15 C2PSA Block	34
Hình 2.16 Detect Block	35
Hình 2.17 Sơ đồ YOLO11	36
Hình 2.18 Multi-Object Tracking	38
Hình 2.19 Mô tả SORT	43
Hình 2.20 Mô tả Deep SORT	44
Hình 2.21 Mạng WRN trong Deep SORT	45
Hình 2.22 Mã giả Matching Cascade	47
Hình 2.23 Track lifecycle	48
Hình 2.24 Mô tả Deep SORT	49
Bảng 3.1 So sánh giữa video tự quay và có sẵn	52
Bảng 3.2 Thông tin về các class	52
Hình 3.1 Tỷ lệ train, valid, test trong dataset	53
Hình 3.2 Kết quả train với variant nano	54
Hình 3.3 Kết quả train với variant small	54
Hình 3.4 Kết quả train với variant medium	55
Hình 3.5 So sánh thời gian train	56
Bảng 3.3 Time inference	57
Hình 3.6 Tổng quan hệ thống	60

Hình 3.7 Lưu đồ giải thuật trên raspberry	61
Hình 3.8 Lưu đồ giải thuật ở sever host	62
Hình 3.9 Web stream	63

# CHƯƠNG 1: GIỚI THIỆU

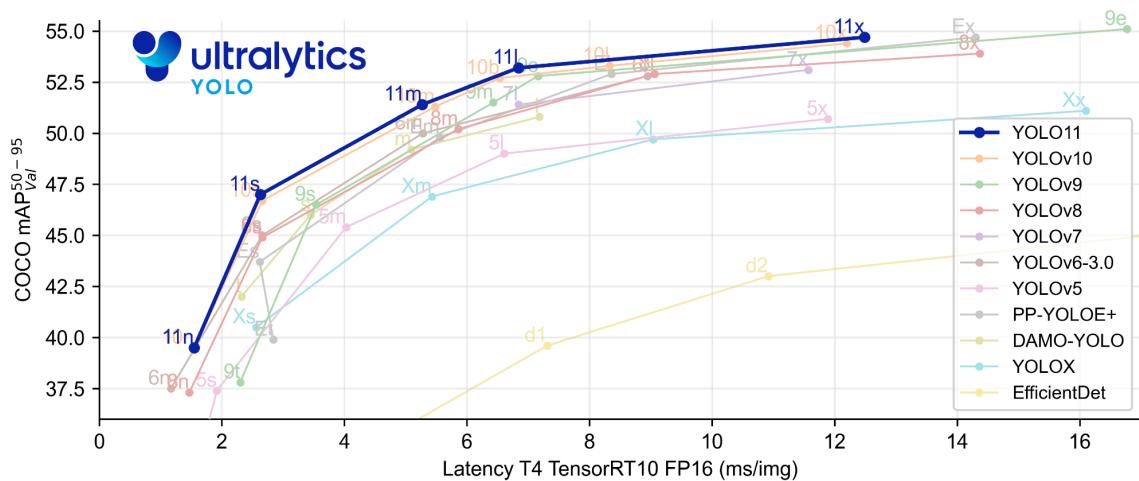
## 1.1 Tổng quan

Trong những năm gần đây, việc phát hiện và nhận dạng đối tượng dưới nước đã trở thành một lĩnh vực nghiên cứu quan trọng với nhiều ứng dụng trong hải dương học, khảo cổ học dưới nước, khai thác tài nguyên biển và an ninh quốc phòng. Sự gia tăng các hoạt động kinh tế và nghiên cứu khoa học dưới biển đòi hỏi các hệ thống quan sát và phát hiện đối tượng hiệu quả. Tuy nhiên, môi trường dưới nước đặt ra nhiều thách thức đặc thù như ánh sáng yếu, độ mờ cao, tán xạ và hấp thụ ánh sáng, cùng với biến dạng màu sắc, khiến cho việc phát hiện và nhận dạng đối tượng trở nên phức tạp hơn so với môi trường trên cạn.



*Hình 1.1 Môi trường dưới nước với nhiều nhiễu*

Hiện nay, nhiều phương pháp đã được đề xuất để giải quyết vấn đề này, bao gồm các kỹ thuật xử lý ảnh truyền thống và các mô hình học sâu. Các phương pháp xử lý ảnh truyền thống thường gặp khó khăn trong việc thích ứng với môi trường dưới nước biến đổi và không đạt được độ chính xác cao. Trong khi đó, các mô hình học sâu như R-CNN, YOLO và đặc biệt là phiên bản mới nhất YOLOv11 đã chứng tỏ hiệu quả vượt trội trong việc phát hiện đối tượng trên cạn. Tuy nhiên, việc áp dụng trực tiếp các mô hình này cho môi trường dưới nước vẫn còn hạn chế do sự khác biệt lớn về đặc tính dữ liệu và môi trường hoạt động.



*Hình 1.2 Sự vượt trội của YOLOv11 so với các phiên bản trước*

Do đó, cần có một nghiên cứu chi tiết để tùy chỉnh và tối ưu hóa các mô hình học sâu hiện đại nhằm phù hợp với đặc thù của môi trường dưới nước. Luận văn này sẽ tập trung vào việc sử dụng và so sánh hai kiến trúc phổ biến là YOLOv11 và trong việc phát hiện đối tượng dưới nước.

## 1.2 Phạm vi và phương pháp nghiên cứu

- Làm thế nào để tùy chỉnh và tối ưu hóa mô hình YOLOv11 để phù hợp với đặc thù của dữ liệu hình ảnh dưới nước?
- Các kỹ thuật tiền xử lý và tăng cường dữ liệu nào hiệu quả trong việc cải thiện chất lượng hình ảnh dưới nước cho quá trình huấn luyện mô hình?
- So sánh hiệu suất giữa YOLOv11 trong việc phát hiện đối tượng dưới nước, mô hình nào đạt được sự cân bằng tốt hơn giữa độ chính xác và tốc độ xử lý cho các ứng dụng thực tế?

## 1.3 Nhiệm vụ luận văn

Luận văn này có các đóng góp như sau:

- Hiện thực hóa
- Mô phỏng
- So sánh

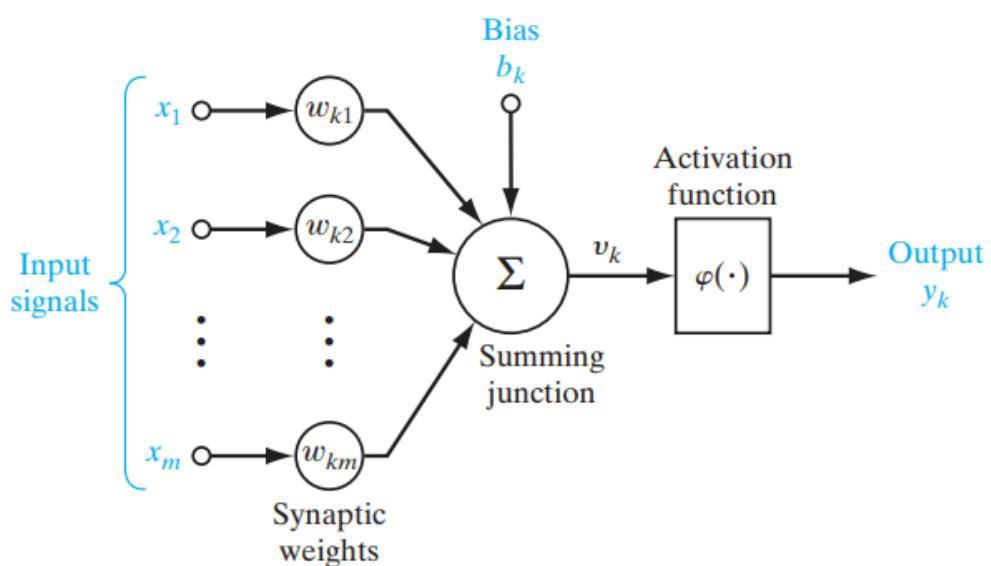
## CHƯƠNG 2: LÝ THUYẾT

### 2.1 Giới thiệu về mạng neuron nhân tạo và deep learning

#### 2.1.1 Định nghĩa

Lấy ý tưởng từ cách các neuron hoạt động trong não người, mạng neural cơ bản gồm ba phần chính là lớp đầu vào, các lớp ẩn và lớp đầu ra. Trong mỗi lớp này, các neuron liên kết chặt chẽ với nhau, cho phép mô hình có khả năng học hỏi và nắm bắt các mối quan hệ phức tạp của dữ liệu. Mỗi neuron được liên kết qua các trọng số, và chính sự phối hợp của chúng tạo thành một hệ thống hợp nhất để giải quyết những bài toán cụ thể. Bản chất của deep learning nằm ở việc điều chỉnh các trọng số giữa các neuron này. Các neuron cơ bản trong mạng còn được gọi là Perceptron, nơi thông tin được xử lý và xuất ra tại lớp đầu ra. Nhiều đơn vị như vậy ghép lại với nhau mô phỏng hệ thần kinh sinh học của con người, từ đó hình thành nên cái tên Mạng Neural (Neural Networks). Đôi khi, hệ thống này còn được biết đến dưới tên gọi Mạng Neuron Nhân Tạo (Artificial Neural Networks - ANN).

#### 2.1.2 Cấu tạo của một neuron trong Mạng Neural



Hình 2.1 Cấu tạo mạng neuron

Các thành phần chính của một mạng neuron nhân tạo bao gồm:

- **Tập đầu vào:** Đây là những tín hiệu được đưa vào neuron, thường được biểu diễn dưới dạng một vector có N chiều.
- **Tập các trọng số kết nối:** Mỗi kết nối giữa các neuron được đặc trưng bởi một trọng số, gọi là "trọng số liên kết" hay Synaptic weight. Trọng số này biểu thị độ mạnh của kết nối giữa đầu vào thứ j và neuron k, được ký hiệu là  $w_{kj}$ . Khi bắt đầu, các trọng số này thường được khởi tạo ngẫu nhiên và sẽ liên tục được điều chỉnh trong quá trình học của mạng.
- **Hàm tổng:** Thường được sử dụng để tính tổng các giá trị đầu vào nhân với trọng số của chúng, từ đó đưa ra kết quả tổng hợp.
- **Bias (độ lệch):** Đây là thành phần điều chỉnh nhằm giúp mô hình dự đoán chính xác hơn bằng cách giảm thiểu sai số giữa giá trị dự đoán và giá trị thực tế.
- **Hàm kích hoạt (hàm truyền):** Được sử dụng để chuẩn hóa tín hiệu đầu ra của neuron vào một khoảng giá trị cụ thể hoặc một tập hợp giá trị xác định.
- **Đầu ra:** Đây là tín hiệu cuối cùng từ neuron, thường mỗi neuron chỉ có một đầu ra duy nhất.

Những thành phần này kết hợp để xây dựng một mạng neuron nhân tạo có khả năng học và tối ưu hóa. Được mô tả bằng biểu thức:

$$u_k = \sum_{j=1}^m w_{kj}x_j \quad (1)$$

$$v_k = u_k + b_k \quad (2)$$

$$y_k = \varphi(v_k) \quad (3)$$

Trong đó:  $x_1, x_2, \dots, x_m$  là các tín hiệu đầu vào;  $w_{k1}, w_{k2}, \dots, w_{km}$  là các trọng số liên kết của neuron thứ k;  $u_k$  là hàm tổng hợp;  $b_k$  là giá trị độ lệch;  $\varphi(v_k)$  là hàm kích hoạt và  $y_k$  là tín hiệu đầu ra từ neuron có nhãn k.

### 2.1.3 Một số hàm kích hoạt phổ biến (active function)

Hàm kích hoạt (activation function) là một thành phần quan trọng trong mạng neuron nhân tạo, giúp điều chỉnh tín hiệu đầu ra của mỗi neuron, đưa tín hiệu này vào một khoảng giá trị cụ thể. Đây là bước không thể thiếu trong quá trình học và phân loại của mạng neuron, bởi các hàm này quyết định cách thức mô hình xử lý thông tin phi tuyến tính từ dữ liệu. Một số hàm kích hoạt phổ biến bao gồm:

**Hàm Sigmoid:** Hàm Sigmoid thường được sử dụng trong các mô hình phân loại vì đầu ra của nó nằm trong khoảng từ 0 đến 1, phù hợp cho việc dự đoán xác suất. Giúp làm mịn và chuẩn hóa đầu ra nhưng có hạn chế là dễ gây hiện tượng "vanishing gradient" khi giá trị  $\bar{x}$  lớn hoặc nhỏ.

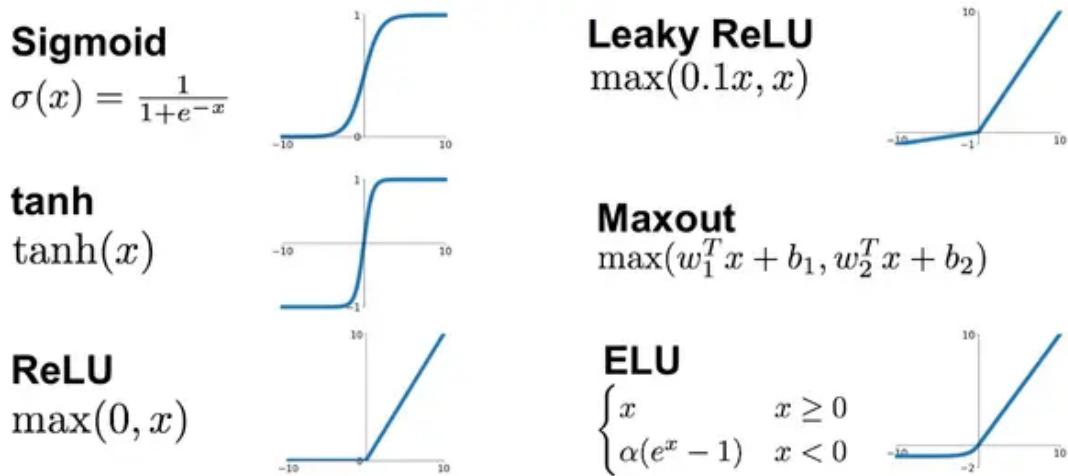
**Hàm Tanh (Hyperbolic Tangent):** đưa tín hiệu đầu ra vào khoảng từ -1 đến 1. So với Sigmoid, Tanh giúp giảm bớt hiện tượng mất gradient, nhờ vào đầu ra tập trung hơn quanh giá trị 0, phù hợp cho các mô hình cần xác định giá trị âm và dương rõ ràng.

**Hàm ReLU (Rectified Linear Unit):** Đây là hàm kích hoạt phổ biến nhất trong các mạng neuron hiện đại, đặc biệt là mạng sâu (Deep Neural Networks). Mọi giá trị âm được đưa về 0 và giá trị dương giữ nguyên. ReLU giúp tăng tốc độ huấn luyện do tính toán đơn giản, nhưng có thể gây ra vấn đề "dying ReLU" khi quá nhiều neuron đầu ra là 0, đặc biệt là trong các lớp sâu.

**Hàm Leaky ReLU:** Để khắc phục vấn đề của ReLU, hàm Leaky ReLU cho phép một lượng nhỏ giá trị âm đi qua với hệ số dốc nhỏ. Hàm này giúp giảm khả năng chết của neuron và cải thiện hiệu suất trong một số bài toán.

**Hàm Maxout:** hoạt động như một dạng tổng quát hóa của ReLU và Leaky ReLU. Thay vì áp dụng một phép toán đơn giản lên đầu vào, Maxout tính toán giá trị đầu ra bằng cách chọn giá trị lớn nhất từ một nhóm các tuyến tính có trọng số khác nhau. Với khả năng phi tuyến mạnh mẽ và giảm vấn đề vanishing gradient.

**ELU (Exponential Linear Unit)** là một hàm kích hoạt phi tuyến khác được thiết kế để khắc phục một số hạn chế của ReLU, đặc biệt là hiện tượng dying ReLU (neuron bị chết).



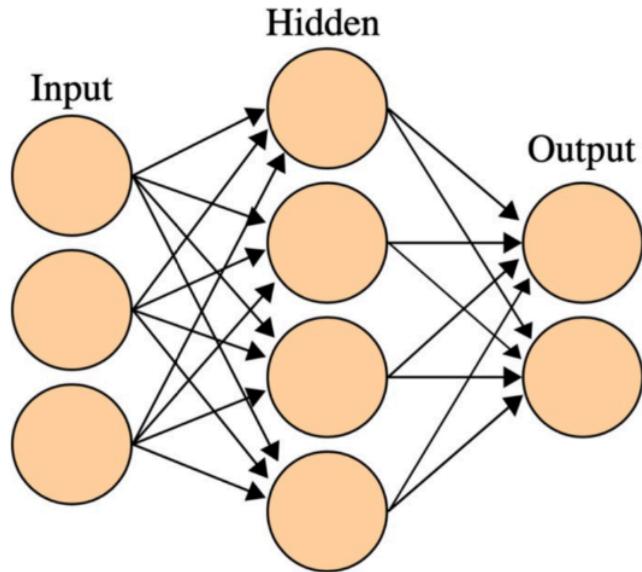
Hình 2.2 Một số hàm kích hoạt phổ biến

## 2.2 Giới thiệu về Multi-layers Perceptron (MLP)

### 2.2.1 Kiến trúc của MLP

**Multi-Layer Perceptron (MLP)**, hay mạng Perceptron nhiều lớp, là một dạng mạng neuron nhân tạo phổ biến và được sử dụng rộng rãi trong các bài toán học máy, đặc biệt là học có giám sát (supervised learning). MLP được xem là mô hình cơ bản của học sâu (deep learning), nhờ cấu trúc nhiều lớp cho phép học các mối quan hệ phi tuyến tính và phức tạp trong dữ liệu. MLP gồm ba loại lớp chính:

- **Lớp đầu vào (Input Layer)**: Đây là lớp nhận dữ liệu đầu vào từ ngoài môi trường. Mỗi neuron trong lớp đầu vào đại diện cho một đặc trưng (feature) của dữ liệu, và số lượng neuron bằng với số chiều của vector đầu vào.
- **Các lớp ẩn (Hidden Layers)**: Đây là lớp trung gian nằm giữa lớp đầu vào và lớp đầu ra. MLP có thể có một hoặc nhiều lớp ẩn tùy theo độ phức tạp của bài toán. Mỗi lớp ẩn gồm nhiều neuron và mỗi neuron trong lớp này được kết nối với tất cả các neuron trong lớp liền trước. Lớp ẩn là nơi các tính toán phức tạp xảy ra, giúp mạng học các đặc trưng phi tuyến tính của dữ liệu.
- **Lớp đầu ra (Output Layer)**: Đây là lớp đưa ra kết quả cuối cùng của mạng. Số lượng neuron trong lớp đầu ra phụ thuộc vào loại bài toán, ví dụ, một neuron cho bài toán phân loại nhị phân hoặc nhiều neuron cho bài toán phân loại đa lớp.



Hình 2.3 Mạng nhiều lớp

### 2.2.2 Nguyên lý hoạt động MLP

Nguyên lý hoạt động của Multi-Layer Perceptron (MLP) được xây dựng dựa trên các phép toán cơ bản, trong đó các tín hiệu đầu vào được lan truyền qua các lớp neuron với các phép tính tích chập, cộng, và áp dụng hàm kích hoạt. Quá trình này có thể được tóm tắt qua ba bước chính là lan truyền tiến (forward propagation), lan truyền ngược (backward propagation), và cập nhật trọng số. Dưới đây là cách MLP hoạt động về mặt toán học.

#### Lan truyền tiến (Forward Propagation)

Trong giai đoạn lan truyền tiến, dữ liệu đầu vào sẽ lần lượt truyền qua từng lớp của MLP, bao gồm lớp đầu vào, các lớp ẩn, và lớp đầu ra, để tính toán đầu ra cuối cùng.

##### Bước 1: Tổng hợp tín hiệu

Với một đầu vào  $x = (x_1, x_2, \dots, x_m)$ , mỗi neuron trong lớp ẩn sẽ tính tổng trọng số của các đầu vào, kết hợp với một giá trị bias. Đối với neuron  $j$  trong lớp ẩn  $l$ , tổng hợp tín hiệu  $z_j^{(l)}$  có thể được tính như sau:

$$z_j^{(l)} = \sum_{i=1}^m w_k w_{ij}^{(l)} x_i + b_j^{(l)} \quad (1)$$

trong đó:

-  $w_{ij}^{(l)}$  là trọng số kết nối giữa đầu vào  $x_i$  và neuron  $j$  trong lớp  $l$ ,

-  $b_j^{(l)}$  là giá trị bias của neuron  $j$ .

### Bước 2: Áp dụng hàm kích hoạt

Để tạo ra đầu ra của neuron  $j$  trong lớp  $l$ , ta áp dụng một hàm kích hoạt  $\varphi$  lên tổng hợp tín hiệu  $z_j^{(l)}$ :

$$y_j^{(l)} = \varphi(z_j^{(l)}) \quad (2)$$

Hàm kích hoạt  $\varphi$  có thể là Sigmoid, ReLU, Tanh, hoặc các hàm khác, tùy thuộc vào yêu cầu của bài toán.

Quá trình này được lặp lại qua từng lớp, từ lớp ẩn đầu tiên đến lớp ẩn cuối cùng và sau đó là lớp đầu ra. Đầu ra cuối cùng của MLP, thường ký hiệu là  $\bar{y}$ , được tính dựa trên các lớp trước đó và phụ thuộc vào hàm kích hoạt của lớp đầu ra.

### Lan truyền ngược (Backward Propagation)

Lan truyền ngược là quá trình tính toán và phân phối lỗi trở ngược từ lớp đầu ra đến các lớp trước đó, để từ đó điều chỉnh trọng số. Lan truyền ngược dựa vào Gradient Descent và được tính toán như sau:

#### Bước 1: Tính độ lỗi ở lớp đầu ra

Đầu tiên, tính độ lỗi của lớp đầu ra. Giả sử hàm mất mát là  $L(\bar{y}, y)$ , với  $\bar{y}$  là đầu ra dự đoán và  $y$  là giá trị thực tế. Độ lỗi ở neuron  $k$  trong lớp đầu ra có thể được tính bằng đạo hàm của hàm mất mát:

$$\delta_k^{(L)} = \partial L / \partial a_k^{(L)} \quad (3)$$

trong đó  $L$  là lớp cuối cùng của MLP.

#### Bước 2: Tính gradient ở các lớp ẩn

Để tính gradient tại lớp ẩn, ta sử dụng quy tắc chuỗi để tính đạo hàm của hàm mất mát đối với từng trọng số và bias trong các lớp trước. Đối với mỗi lớp  $l$  và mỗi neuron  $j$ , gradient của trọng số  $w_{ij}^{(l)}$  được tính như:

$$\delta_k^{(l)} = \varphi'(z_j^{(l)}) \sum_k \delta_k^{(l+1)} w_{ij}^{(l+1)} \quad (4)$$

trong đó  $\varphi'$  là đạo hàm của hàm kích hoạt, và  $\delta_k^{(l+1)}$  là lỗi lan truyền ngược từ lớp sau.

### Cập nhật trọng số (Weight Update)

Dựa trên gradient được tính từ lan truyền ngược, trọng số và bias trong mạng sẽ được cập nhật bằng phương pháp Gradient Descent hoặc các biến thể như Stochastic Gradient Descent. Công thức cập nhật trọng số  $w_{ij}^{(l)}$  và bias  $b_b^{(l)}$  là:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \cdot \partial L / \partial w_{ij}^{(l)} \quad (5)$$

$$b_b^{(l)} = b_b^{(l)} - \eta \cdot \partial L / \partial b_b^{(l)} \quad (6)$$

trong đó  $\eta$  là tốc độ học (learning rate) và  $\partial L / \partial w_{ij}^{(l)}$  là gradient của hàm mất mát theo trọng số  $w_{ij}^{(l)}$ .

Quá trình này được lặp lại qua nhiều vòng huấn luyện để tối ưu hóa các trọng số và bias trong mạng, giúp MLP đạt được độ chính xác cao trên dữ liệu huấn luyện.

## 2.3 Convolutional Neural Network (Mạng neuron tích chập)

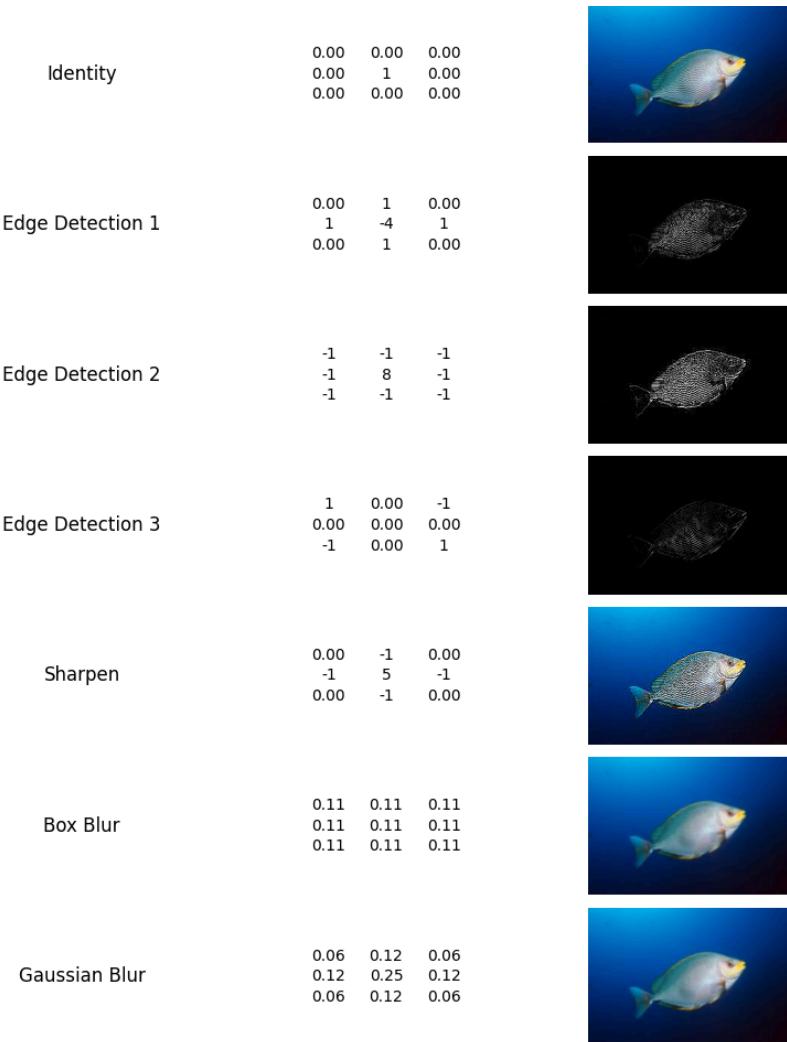
Mạng Neural Tích chập (Convolutional Neural Network - CNN) là một loại mạng neural nhân tạo được phát triển dựa trên ý tưởng học cách trích xuất các đặc trưng không gian từ dữ liệu hình ảnh. Hiện nay, CNN là một công cụ phổ biến trong các lĩnh vực như nhận dạng hình ảnh, phân loại đối tượng, và xử lý ngôn ngữ tự nhiên. CNN khác biệt so với các mạng neural truyền thống ở chỗ nó có khả năng trích xuất thông tin không gian từ hình ảnh qua các lớp tích chập và lớp pooling, giúp giảm thiểu số lượng tham số cần học và tăng cường khả năng học đặc trưng.

### 2.3.1 Các thành phần của một CNN

#### 2.3.1.1 Convolutional Layer (Lớp tích chập)

Lớp tích chập (**Convolutional Layer**) là thành phần quan trọng nhất của CNN. Nó sử dụng một số lượng bộ lọc (kernel) nhỏ di chuyển trên toàn bộ dữ liệu đầu vào, giúp trích xuất các đặc trưng cục bộ từ hình ảnh. Một đặc điểm nổi bật của lớp tích chập là khả năng học các đặc trưng không gian cục bộ, thay vì học toàn bộ

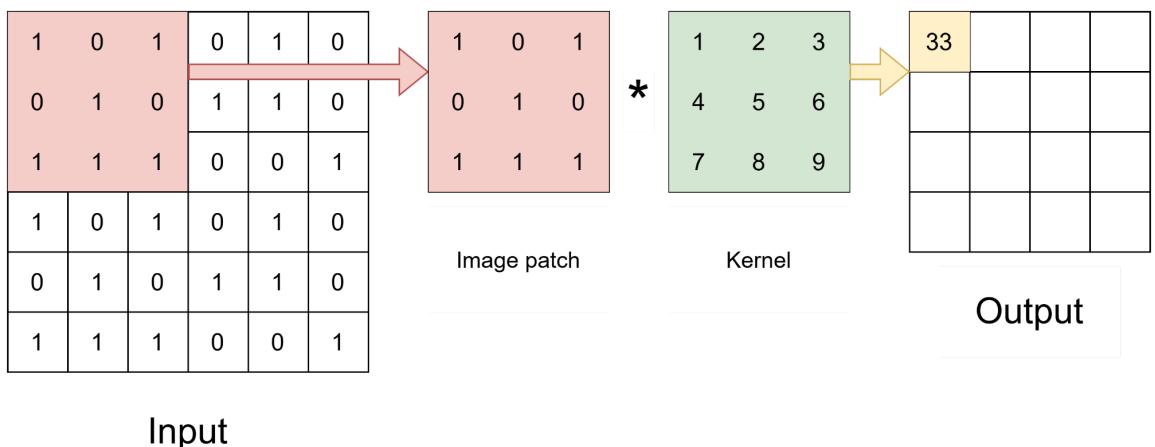
dữ liệu như trong các mạng neural thông thường. Điều này giúp CNN giảm bớt số lượng tham số và tăng khả năng tổng quát hóa cho các bài toán thị giác máy tính. Các bộ lọc trong lớp tích chập thường được học thông qua quá trình huấn luyện, nơi mỗi bộ lọc học cách phát hiện một kiểu đặc trưng riêng biệt.



*Hình 2.4 Các loại kernel*

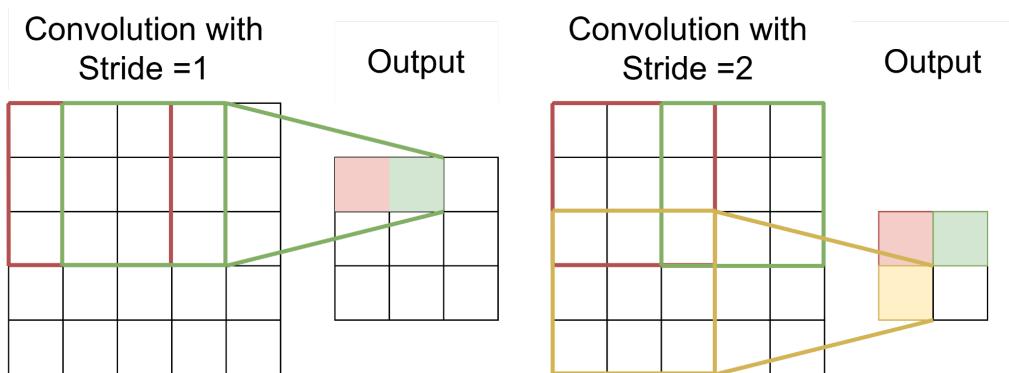
**Kernel** là ma trận . **Convolutional Layer** có được khi ta nhân từng bộ lọc với một phần của đầu vào và tạo ra một ma trận đầu ra, được gọi là feature map. Những bộ lọc này cho phép mạng học cách nhận diện các đặc trưng như cạnh, góc, và hình dạng từ hình ảnh gốc. Giả sử có một đầu vào  $X$  với kích thước  $H \times W$  và một kernel  $K$  kích thước  $h \times w$ , đầu ra  $Y$  của lớp tích chập sẽ được tính như sau:

$$Y(i, j) = \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} (i + m, j + n) \bullet K(m, n) \quad (1)$$



*Hình 2.5 Giải thích kernel*

**Stride** là một tham số quan trọng trong lớp tích chập của mạng nơ-ron tích chập (CNN), xác định khoảng cách di chuyển của bộ lọc (kernel) trên đầu vào trong quá trình thực hiện phép tích chập. Nói cách khác, stride điều khiển cách bộ lọc "truột" qua dữ liệu đầu vào. Stride ảnh hưởng trực tiếp đến kích thước không gian (chiều cao và chiều rộng) của đầu ra.

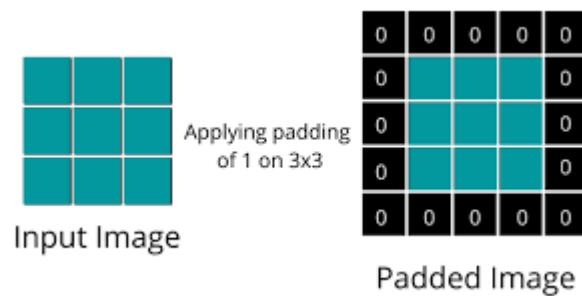


*Hình 2.6 Giải thích Stride*

**Padding** là một kỹ thuật quan trọng trong lớp tích chập của mạng nơ-ron tích chập, được sử dụng để kiểm soát kích thước không gian của đầu ra và giữ lại thông

tin ở biên của dữ liệu đầu vào. Quá trình này thêm các giá trị (thường là zero) xung quanh biên của dữ liệu đầu vào trước khi thực hiện phép tích chập.

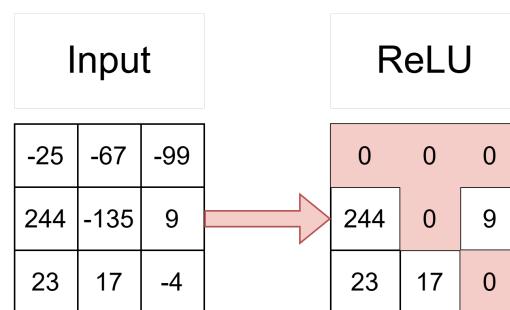
- Zero Padding: Thêm các giá trị zero.
- Reflect Padding: Phản chiếu giá trị của biên.
- Constant Padding: Thêm một giá trị hằng số cụ thể.



Hình 2.7 Giải thích padding

### 2.3.1.2 Nonlinear activation function (Hàm phi tuyến)

Bởi vì hình ảnh là dữ liệu không âm nên lớp tích chập sử dụng các hàm phi tuyến như ReLU với  $f(x) = \max(0, x)$  đầu ra là ảnh xám (0 đến 255) cho hiệu suất tốt nhất hoặc hàm phi tuyến sigmoid với  $f(x) = \frac{1}{1+e^{-x}}$  cho ra đen trắng (0 hoặc 1)



Hình 2.8 Hàm phi tuyến

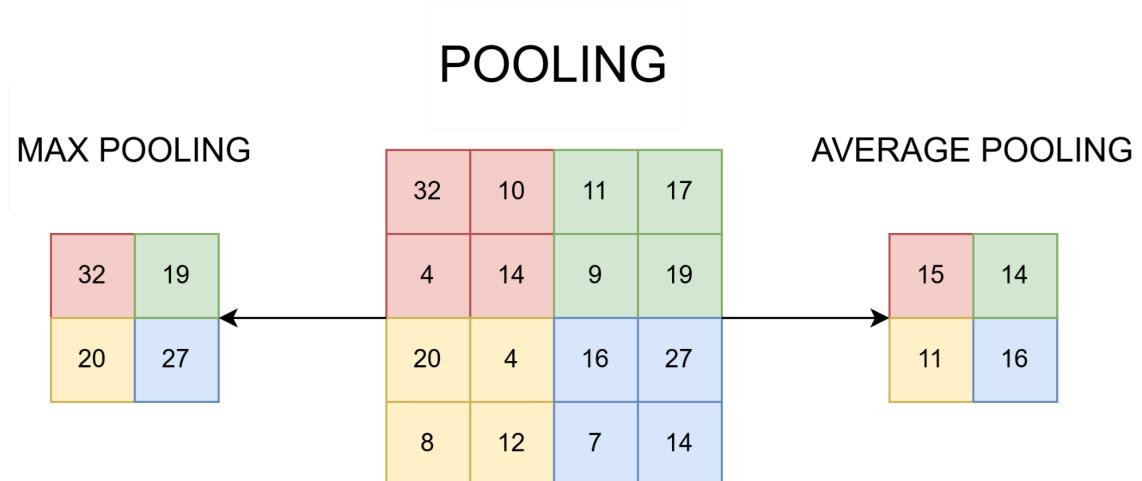
### 2.3.1.3 Pooling Layer (lớp tổng hợp)

Lớp pooling được sử dụng để giảm kích thước của các feature map sau lớp tích chập, từ đó giảm số lượng tham số và tránh hiện tượng overfitting.

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling sẽ chọn phần tử lớn nhất trong ma trận vùng ảnh, trong khi average pooling sẽ lấy giá trị trung bình, và sum pooling sẽ tính tổng của tất cả các phần tử trong map.

Max pooling thường được sử dụng để bảo toàn các đặc trưng quan trọng trong khi giảm độ phân giải của feature map. Việc giảm kích thước dữ liệu không chỉ giúp giảm chi phí tính toán mà còn giúp tăng tính bát biến của mô hình đối với các thay đổi nhỏ về vị trí của đối tượng trong hình ảnh.



Hình 2.9 Giải thích pooling

### 2.3.1.4 Fully connected layer (lớp kết nối đầy đủ)

Lớp kết nối đầy đủ là lớp cuối cùng của CNN, nơi các đặc trưng đã được trích xuất từ các lớp trước được sử dụng để phân loại hoặc dự đoán. Lớp này có cấu

trúc tương tự như mạng neural truyền thống, với mỗi neuron kết nối với tất cả các neuron của lớp trước. Kết quả từ các lớp tích chập và pooling sẽ được đưa vào lớp kết nối dày đủ để tạo ra dự đoán cuối cùng.

### 2.3.2 Điểm Nổi Bật của CNN

Mạng CNN bao gồm nhiều lớp Convolution xếp chồng lên nhau, kết hợp các hàm kích hoạt phi tuyến để kích hoạt trọng số. Mỗi lớp sau khi được kích hoạt sẽ tạo ra một kết quả trừu tượng, trở thành đầu vào cho lớp kế tiếp. Mỗi lớp tiếp theo là sự biểu diễn của lớp trước đó. Trong quá trình huấn luyện, các lớp CNN tự động học các đặc điểm thông qua các filter.

CNN có tính bất biến và tính kết hợp cục bộ (Location Invariance và Compositionality). Với một đối tượng duy nhất, nếu nó được chiếu qua các góc nhìn khác nhau (dịch chuyển, xoay, co giãn), độ chính xác của thuật toán sẽ bị ảnh hưởng. Lớp Pooling giúp duy trì tính bất biến với các phép dịch chuyển, xoay và co giãn.

Tính kết hợp cục bộ cho phép mô hình biểu diễn thông tin từ các đặc trưng cấp thấp đến các đặc trưng cấp cao và trừu tượng hơn thông qua các filter. Đây là lý do CNN đạt độ chính xác cao. Tương tự như cách con người phân biệt đối tượng trong tự nhiên, chúng ta có thể nhận ra một con chó và một con mèo dựa trên các đặc điểm từ mức độ thấp (chân, đuôi) đến mức độ cao hơn (dáng đi, hình thể, màu lông).

Cấu trúc cơ bản của CNN gồm 3 phần chính: trường tiếp nhận cục bộ, trọng số và độ lệch chia sẻ, và lớp Pooling:

- **Trường tiếp nhận cục bộ (Local receptive field):** giúp tách lọc thông tin của ảnh, xác định vùng có giá trị sử dụng cao nhất.
- **Trọng số và độ lệch chia sẻ (Shared weights and bias):** giảm số lượng tham số tối đa bằng cách chia sẻ trọng số giữa các feature map, giúp nhận diện các đặc điểm khác nhau trong ảnh.

- **Lớp Pooling:** dùng để giảm kích thước của các feature map sau lớp tích chập, giảm số lượng tham số và hạn chế hiện tượng overfitting.

## 2.4 Transfer Learning

Transfer Learning dựa trên việc sử dụng lại một mô hình đã huấn luyện trước trên một nhiệm vụ khác, giúp tiết kiệm thời gian và tài nguyên, đặc biệt là khi dữ liệu hạn chế. Quá trình này bao gồm chọn mô hình nguồn, chuyển giao kiến thức, và fine-tuning để tối ưu hóa mô hình cho nhiệm vụ mới. Các phương pháp phổ biến của Transfer Learning gồm trích xuất đặc trưng, fine-tuning toàn bộ mô hình, và đóng băng các lớp đầu. Tùy vào nhiệm vụ và dữ liệu, mỗi phương pháp sẽ mang lại hiệu quả khác nhau, nhưng đều tận dụng kiến thức đã học từ trước để cải thiện hiệu suất.

**Feature Extraction (trích xuất đặc trưng):** Trong phương pháp này, các đặc trưng học được từ mô hình nguồn sẽ được giữ nguyên, và chỉ có các lớp cuối cùng (classifier) sẽ được huấn luyện lại. Đây là cách tiếp cận hiệu quả khi dữ liệu của nhiệm vụ mới không quá lớn hoặc có những đặc điểm tương đồng với dữ liệu gốc.

**Fine-tuning toàn bộ mô hình:** Phương pháp này cho phép huấn luyện lại toàn bộ mạng nơ-ron để tối ưu hóa cho nhiệm vụ mới. Cách tiếp cận này thường được áp dụng khi dữ liệu huấn luyện mới đủ lớn và có khả năng làm thay đổi các đặc trưng từ dữ liệu nguồn.

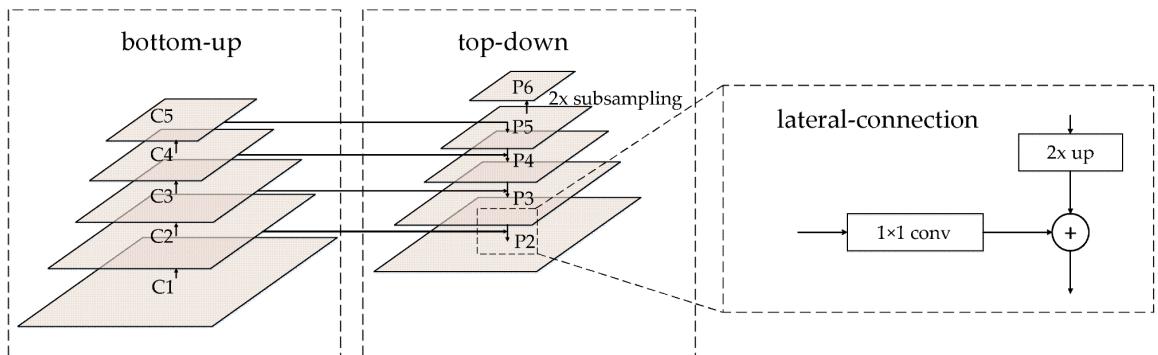
**Đóng băng lớp:** Đây là một kỹ thuật trong đó một số lớp đầu của mô hình nguồn được “đóng băng” và giữ nguyên, chỉ cho phép các lớp cuối cùng được huấn luyện lại. Điều này giúp tiết kiệm tài nguyên tính toán và tránh làm thay đổi kiến thức cơ bản của mô hình.

## 2.5 Feature Pyramid Networks (FPN)

FPN dựa trên một mạng nơ-ron tích chập cơ bản (Convolutional Neural Network - CNN), để tạo ra một chuỗi các đặc trưng ở các cấp độ khác nhau, mỗi cấp độ mang thông tin không gian và ngữ cảnh đặc trưng. Kiến trúc FPN được thiết kế để tối ưu hóa việc học các đặc trưng ở từng cấp độ, đảm bảo rằng các đặc trưng

từ các cấp độ cao (độ phân giải thấp) và các đặc trưng từ các cấp độ thấp (độ phân giải cao) đều có thể được sử dụng một cách hiệu quả. Quá trình này bao gồm hai phần chính: **bottom-up pathway** và **top-down pathway**.

- **Bottom-up pathway:** Đây là phần đầu tiên của kiến trúc, nơi các đặc trưng được trích xuất theo từng lớp trong mạng gốc. Các đặc trưng ở các cấp độ này chứa thông tin không gian chi tiết hơn ở các lớp thấp và thông tin ngữ cảnh tổng quan hơn ở các lớp cao.
- **Top-down pathway:** Sau khi có các đặc trưng từ bottom-up pathway, FPN sử dụng top-down pathway để xây dựng một "kim tự tháp đặc trưng". Các đặc trưng ở cấp độ cao sẽ được lấy mẫu lại (upsampling) và kết hợp với các đặc trưng ở các cấp độ thấp tương ứng. Nhờ đó, FPN kết hợp cả thông tin chi tiết từ các đặc trưng ở độ phân giải cao và thông tin tổng quát từ độ phân giải thấp.



Hình 2.10 Kiến trúc FPN

Kết quả là một tập hợp các đặc trưng ở nhiều cấp độ với độ chính xác cao, mỗi cấp độ phù hợp để xử lý các đối tượng ở các kích cỡ khác nhau.

## 2.6 YOLO11

### 2.6.1 YOLO11 Variants

YOLO11 được sử dụng như một mô hình học sâu đã được huấn luyện trước đó trên một tập dữ liệu lớn như COCO và sau đó được tinh chỉnh (fine-tuned) cho

các tác vụ cụ thể. Dành cho nhận diện và phát hiện đối tượng trong hình ảnh. YOLO11 có nhiều biến thể (variants), mỗi biến thể được thiết kế với kích thước và cấu hình khác nhau để phù hợp với từng loại ứng dụng và tài nguyên xử lý khác nhau. Các biến thể chính của YOLO11 bao gồm:

- **YOLO11n (Nano):** Đây là biến thể nhỏ nhất trong dòng YOLO11, với số lượng tham số ít nhất và tốc độ xử lý cao nhất. YOLO11 rất phù hợp với các thiết bị có tài nguyên hạn chế, như điện thoại di động hoặc thiết bị nhúng, nơi mà thời gian phản hồi và tối ưu hóa tài nguyên là ưu tiên hàng đầu. Tuy nhiên, do kích thước nhỏ nên khả năng phát hiện và độ chính xác của YOLO11 có thể thấp hơn so với các biến thể lớn hơn.
- **YOLO11s (Small):** Biến thể này lớn hơn YOLO11n một chút, với thêm một vài lớp xử lý và số lượng tham số nhiều hơn, cung cấp độ chính xác cao hơn nhưng vẫn đảm bảo tốc độ xử lý nhanh. YOLO11s thường được sử dụng cho các ứng dụng thời gian thực trên các thiết bị có GPU tầm trung hoặc trong các trường hợp cần sự cân bằng giữa độ chính xác và tốc độ.
- **YOLO11m (Medium):** Đây là phiên bản tầm trung của YOLO11, có số lượng tham số và độ phức tạp cao hơn YOLO11s. YOLO11m phù hợp với các ứng dụng yêu cầu độ chính xác cao hơn, thường được sử dụng trong các dự án nghiên cứu hoặc ứng dụng đòi hỏi nhận diện phức tạp hơn, chẳng hạn như phân loại hoặc phát hiện nhiều đối tượng khác nhau trong cùng một bối cảnh.
- **YOLO11 (Large):** YOLO11 có kích thước lớn hơn YOLO11m, với số lượng tham số và độ phức tạp cao hơn, do đó đạt độ chính xác rất cao. Tuy nhiên, yêu cầu về tài nguyên xử lý của YOLO11l cũng tăng lên, vì vậy nó phù hợp hơn khi chạy trên các GPU mạnh hoặc trong môi trường xử lý ảnh phức tạp, nơi độ chính xác là ưu tiên hàng đầu.
- **YOLO11 (Extra Large):** Đây là biến thể lớn nhất và mạnh mẽ nhất trong dòng YOLO11, với số lượng tham số nhiều nhất và khả năng phát hiện đối tượng chính xác nhất. YOLO11x thích hợp với các ứng dụng quy mô lớn, chẳng hạn như phân tích video trong thời gian thực, hệ thống giám sát, hoặc

các dự án cần độ chính xác gần tuyệt đối. Tuy nhiên, do kích thước và độ phức tạp, YOLO11x yêu cầu hệ thống phần cứng mạnh mẽ và không phù hợp cho các thiết bị tài nguyên hạn chế.

Model Variant	d	w	mc
n	0.33	0.25	1024
s	0.33	0.5	1024
m	0.67	0.75	768
l	1	1	512
x	1	1.25	512

Bảng 2.1 Model Variant

**depth\_multiple (d):** Tham số này xác định số lượng khối Bottleneck (Bottleneck Blocks) được sử dụng trong khối C3K2, qua đó điều chỉnh độ sâu (depth) của mô hình. Khi giá trị của depth\_multiple nhỏ hơn 1, số lớp của mô hình sẽ giảm, giúp mô hình nhẹ hơn và tốc độ xử lý nhanh hơn, tuy nhiên độ chính xác có thể sẽ bị ảnh hưởng. Ngược lại, khi giá trị này lớn hơn 1, số lớp sẽ tăng lên, làm cho mô hình lớn hơn và có thể đạt độ chính xác cao hơn, nhưng tốc độ xử lý sẽ chậm hơn.

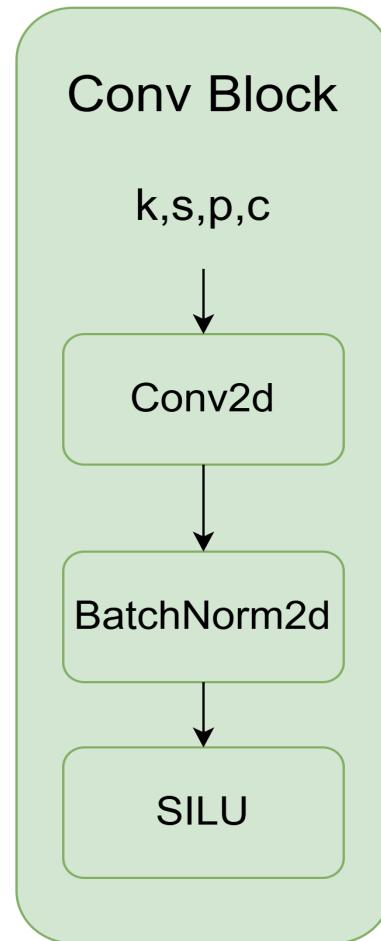
**width\_multiple (w):** Tham số này điều chỉnh số lượng kênh (channels) trong các lớp tích chập (convolutional layers), qua đó điều chỉnh độ rộng của mô hình. Khi width\_multiple nhỏ hơn 1, số lượng kênh giảm, giúp mô hình nhỏ gọn và nhanh hơn, nhưng có thể hy sinh một phần độ chính xác. Ngược lại, khi width\_multiple lớn hơn 1, số lượng kênh tăng, mô hình trở nên rộng hơn, có thể giúp tăng độ chính xác nhưng đòi hỏi nhiều tài nguyên xử lý hơn.

**max\_channels (mc):** Đây là giới hạn trên cho số lượng kênh trong mô hình, giúp kiểm soát tối đa số kênh cho phép trong mạng lưới. Tham số này ngăn không cho mô hình trở nên quá rộng (quá nhiều kênh), đặc biệt khi width\_multiple được

đặt ở mức cao. Điều này giúp kiểm soát kích thước mô hình và ngăn chặn hiện tượng overfitting (quá khớp).

## 2.6.2 Các thành phần trong kiến trúc YOLO11

### 2.6.2.1 Convolutional Block (Conv Block)



Hình 2.11 Conv Block

Đây là khối cơ bản nhất trong kiến trúc mô hình, bao gồm ba lớp: lớp tích chập Conv2d, lớp BatchNorm2d, và hàm kích hoạt SiLU.

#### Lớp Conv2d

Tích chập (Convolution) đã được giải thích ở **2.3.1.1. "2D"** trong Conv2D biểu thị rằng phép tích chập được áp dụng trên hai chiều không gian, thường là chiều cao và chiều rộng.

- **k**: Số lượng filter hoặc kernel. Đây là độ sâu của đầu ra và mỗi filter sẽ chịu trách nhiệm phát hiện các đặc trưng khác nhau trong đầu vào.
- **s**: Stride, là khoảng cách di chuyển của filter/kernel trên đầu vào. Stride càng lớn, kích thước không gian của đầu ra càng nhỏ.
- **p**: Padding, là viền thêm các số 0 vào các cạnh của đầu vào, giúp bảo toàn thông tin không gian và điều chỉnh kích thước không gian của đầu ra.
- **c**: Số lượng kênh đầu vào. Ví dụ, trong ảnh RGB, c sẽ là 3, tương ứng với ba kênh màu: đỏ, xanh lá cây, và xanh dương.

### **Lớp BatchNorm2d**

Batch Normalization (BatchNorm2d) là một kỹ thuật được sử dụng trong mạng nơ-ron sâu nhằm cải thiện độ ổn định khi huấn luyện và tăng tốc độ hội tụ. Trong mạng nơ-ron tích chập (CNN), BatchNorm2d áp dụng chuẩn hóa trên các đầu ra 2D, thường là kết quả của các lớp tích chập. BatchNorm giúp điều chỉnh giá trị trong mạng để không bị quá lớn hoặc quá nhỏ.

### **Hàm kích hoạt SiLU**

SiLU (Sigmoid Linear Unit) là một hàm kích hoạt được sử dụng trong mạng nơ-ron, còn được gọi là hàm kích hoạt Swish. Hàm SiLU được định nghĩa như sau:

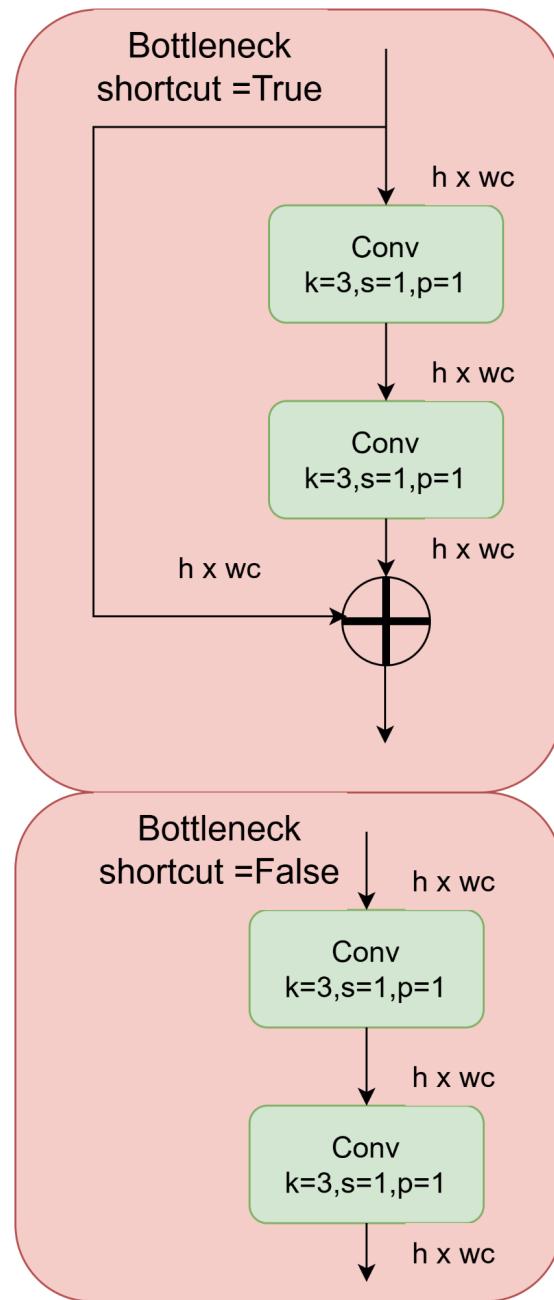
$$SiLU(x) = x \cdot \sigma(x)$$

Trong đó,  $\sigma(x)$  là hàm sigmoid, được định nghĩa bởi công thức:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Điểm đặc trưng của SiLU là nó tạo ra các gradient mượt, giúp ích cho quá trình huấn luyện mạng nơ-ron sâu. Gradient mượt giúp tránh được các vấn đề như vanishing gradients, vốn có thể cản trở quá trình học trong mạng nơ-ron sâu.

### 2.6.2.2 Bottleneck Block



Hình 2.12 Bottleneck Block

Bottleneck Block được thiết kế để giảm số lượng tham số cần thiết, qua đó cải thiện khả năng tính toán và làm cho mô hình nhẹ hơn mà không làm giảm đáng kể độ chính xác. Bottleneck Block có cấu trúc điển hình bao gồm hai lớp tích chập (convolution) và một lớp kết nối ngắn (Shortcut Connection). Đầu tiên, dữ liệu đầu

vào đi qua một lớp tích chập giảm số lượng kênh, giúp giảm số lượng tham số. Sau đó, một lớp tích chập khác mở rộng lại số lượng kênh để phù hợp với đầu ra. Shortcut Connection sẽ giúp giữ lại thông tin ban đầu của dữ liệu và kết hợp với đầu ra của khối để tăng khả năng khôi phục thông tin.

### Kết nối tắt (Shortcut Connection)

Kết nối tắt, hay còn gọi là kết nối bỏ qua hoặc kết nối dư, là một đường nối trực tiếp bỏ qua một hoặc nhiều lớp trong mạng. Kết nối này giúp các giá trị gradient dễ dàng truyền qua mạng trong quá trình huấn luyện, từ đó giải quyết vấn đề ‘gradient bị suy yếu’ và giúp mô hình học hiệu quả hơn.

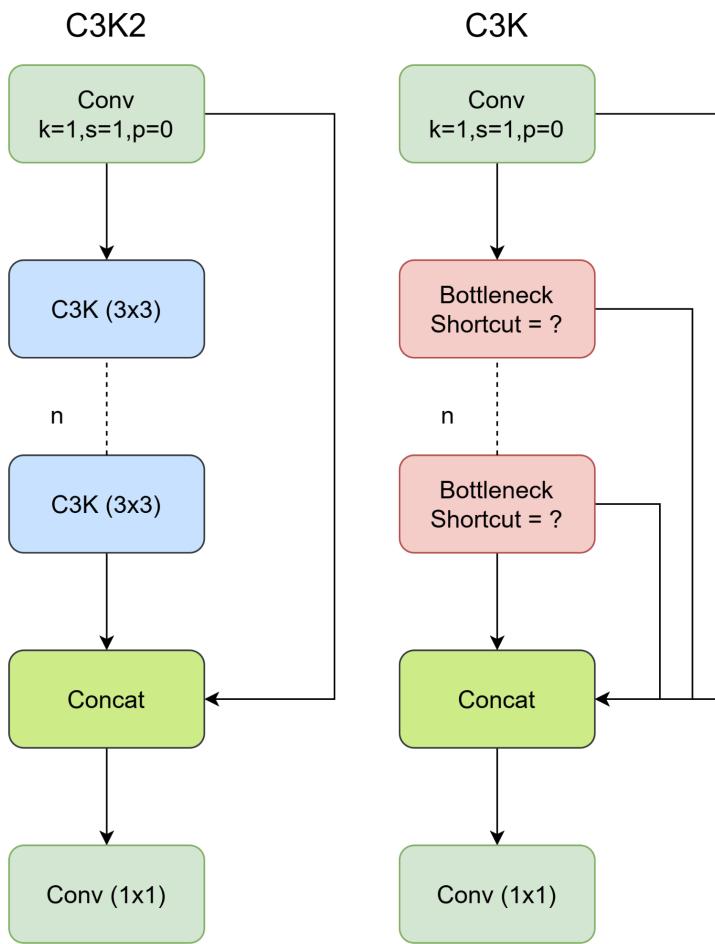
Trong khối bottleneck, kết nối tắt cho phép mô hình bỏ qua các Conv Block khi cần. Nhờ đó, mô hình có thể giữ lại thông tin từ đầu vào mà không qua xử lý, giúp dễ dàng học được các thông tin cơ bản khi cần thiết. Sử dụng kết nối tắt giúp mô hình học được các đặc điểm phức tạp hơn và cải thiện quá trình huấn luyện.

### Vấn đề gradient bị suy yếu là gì?

Vấn đề gradient bị suy yếu xảy ra khi huấn luyện các mạng nơ-ron sâu, đặc biệt là với các mạng có nhiều lớp. Khi đó, gradient của hàm mất mát đối với trọng số của mạng có thể bị giảm quá nhỏ khi truyền ngược từ lớp đầu ra về lớp đầu vào, làm cho các lớp gần đầu vào khó được cập nhật, gây khó khăn cho quá trình học của mô hình.

Tóm lại, Khối bottleneck bao gồm một Conv Block có thêm kết nối tắt (shortcut connection). Nếu `shortcut=true`, kết nối tắt sẽ được sử dụng trong khối bottleneck; nếu không, đầu vào sẽ đi qua hai Conv Block liên tiếp.

### 2.6.2.3 C3K2 và C3K Block



Hình 2.13 C3K2 Block và C3K Block

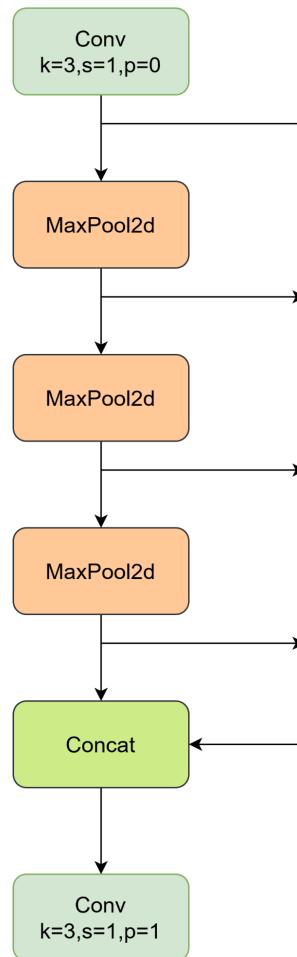
Trọng tâm của backbone YOLO11 là khối C3K2 , đây là sự phát triển của CSP (Cross Stage Partial) được giới thiệu trong các phiên bản trước. Khối C3K2 tối ưu hóa luồng thông tin qua mạng bằng cách chia tách feature map và áp dụng một loạt các phép tích chập với kernel nhỏ hơn (3x3) , nhanh hơn và ít chi phí tính toán so với các phép tích chập hạt nhân lớn hơn.

Khối C3K có cấu trúc tương tự như khối C2F nhưng không có sự phân tách nào được thực hiện ở đây, đầu vào được truyền qua khối Conv sau là một loạt các lớp 'n' Bottle Neck có nối tiếp và kết thúc bằng Khối Conv cuối cùng.

C3K2 sử dụng khối C3K để xử lý thông tin. Nó có 2 khối Conv ở đầu và cuối, tiếp theo là một loạt khối C3K và cuối cùng là nối đầu ra của khối Conv và

đầu ra của khối C3K cuối cùng và kết thúc bằng một khối Conv cuối cùng. Khối này tập trung vào việc duy trì sự cân bằng giữa tốc độ và độ chính xác, tận dụng cấu trúc CSP.

#### 2.6.2.4 Spatial Pyramid Pooling Fast (SPPF) Block:



Hình 2.14 SPPF Block

Khối SPPF bao gồm một lớp tích chập, sau đó là ba lớp MaxPool2d. Mỗi feature map từ các lớp MaxPool2d được ghép lại với nhau và đưa vào một lớp tích chập tiếp theo.

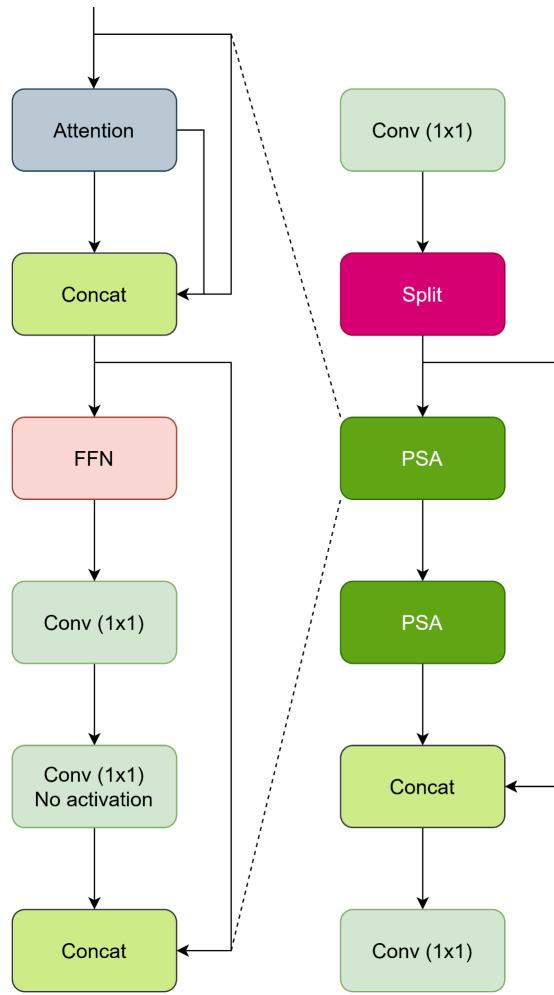
Ý tưởng chính của Spatial Pyramid Pooling (SPP) là chia hình ảnh đầu vào thành các ô nhỏ và lấy đặc trưng riêng từ từng ô này. Điều này cho phép mạng xử lý linh hoạt hình ảnh có kích thước khác nhau.

Với SPP, mạng nơ-ron có thể làm việc hiệu quả hơn với hình ảnh ở nhiều độ phân giải, nhờ khả năng nắm bắt thông tin đa tỉ lệ thông qua quá trình pooling ở các mức chi tiết khác nhau. Điều này đặc biệt hữu ích trong các ứng dụng như nhận dạng đối tượng, khi đối tượng có kích thước khác nhau trong mỗi bức ảnh.

Tuy nhiên, phương pháp SPP truyền thống có thể đòi hỏi nhiều tài nguyên tính toán. Để giải quyết vấn đề này, SPP-Fast sử dụng cách pooling đơn giản hơn, chỉ cần một kernel cố định thay vì nhiều lớp với kích thước kernel khác nhau. Điều này giúp giảm bớt lượng tính toán, mang lại sự cân bằng giữa tốc độ và độ chính xác.

**Lớp MaxPool2d:** được sử dụng để giảm kích thước của đầu vào, từ đó giảm độ phức tạp tính toán và giữ lại các đặc trưng nổi bật nhất. Max pooling là một phương pháp chỉ lấy giá trị lớn nhất trong mỗi vùng của dữ liệu đầu vào.

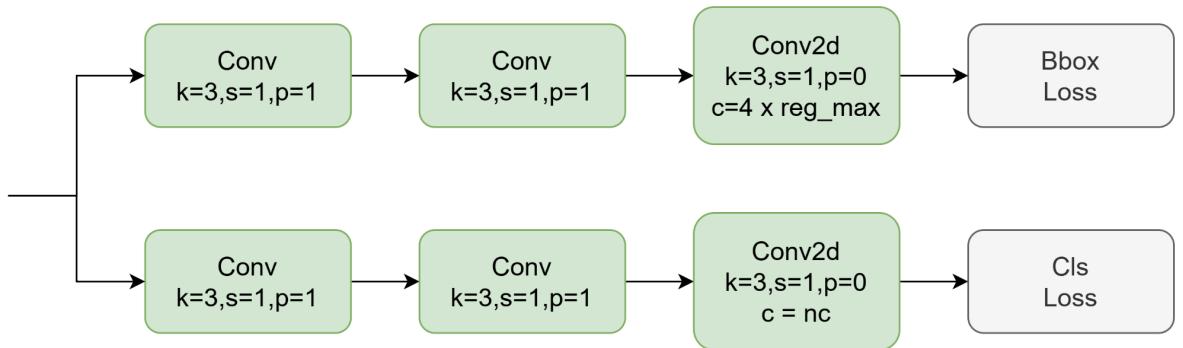
### 2.6.2.5 C2PSA



Hình 2.15 C2PSA Block

Khối C2PSA sử dụng hai mô-đun PSA (Partial Spatial Attention) hoạt động trên các nhánh riêng biệt của feature map và sau đó được nối lại, tương tự như cấu trúc khối C2F ở phiên bản trước. Cấu trúc này đảm bảo mô hình tập trung vào thông tin không gian trong khi vẫn duy trì sự cân bằng giữa chi phí tính toán và độ chính xác phát hiện. Khối C2PSA tinh chỉnh khả năng tập trung có chọn lọc vào các vùng quan tâm của mô hình bằng cách áp dụng Spatial Attention vào các đặc điểm đã trích xuất. Điều này cho phép YOLOv11 hoạt động tốt hơn các phiên bản trước như YOLOv11 trong các tình huống cần có chi tiết để phát hiện chính xác.

### 2.6.2.6 Detect Block

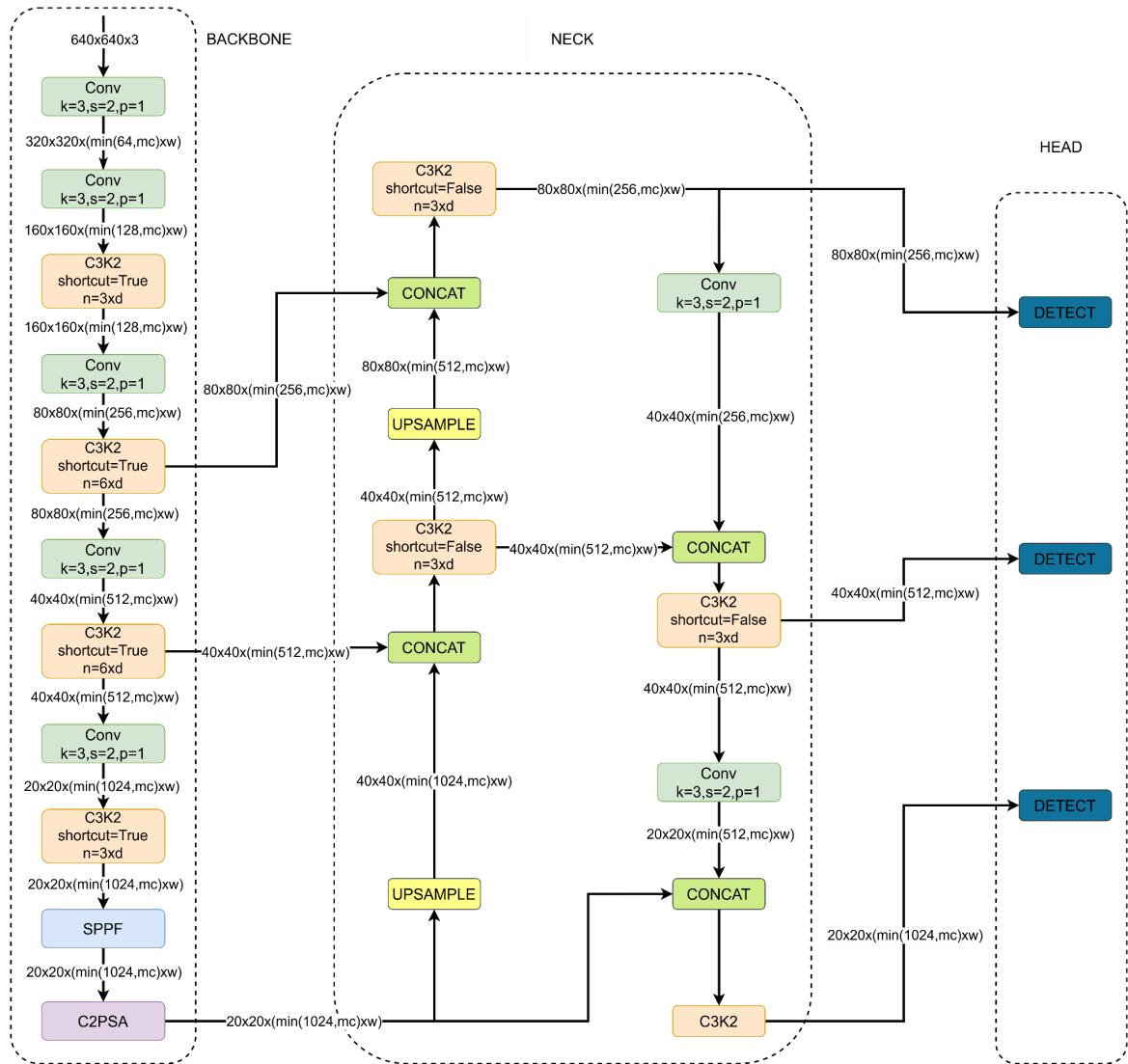


Hình 2.16 Detect Block

Detect Block có nhiệm vụ phát hiện các đối tượng. Khác với các phiên bản YOLO trước, YOLO11 là một mô hình không sử dụng anchor, nghĩa là nó dự đoán trực tiếp vị trí trung tâm của đối tượng thay vì tính toán độ lệch từ các hộp anchor cố định. Việc loại bỏ anchor giúp giảm số lượng dự đoán bounding box, từ đó tăng tốc quá trình xử lý phức tạp sau khi suy luận, khi phải chọn lọc các dự đoán tiềm năng.

Detect Block có hai đường xử lý. Đường đầu tiên dùng để dự đoán hộp bao (bounding box) và đường thứ hai dùng để dự đoán lớp đối tượng (class). Mỗi đường đều bao gồm hai khối tích chập (convolutional blocks), sau đó là một lớp Conv2d duy nhất để đưa ra Bounding Box loss và Class Loss tương ứng.

### 2.6.3 Giải thích Kiến trúc YOLO11



Hình 2.17 Sơ đồ YOLO11

Kiến trúc của YOLO11 bao gồm ba phần chính: Backbone, Neck và Head.

**Backbone** là phần kiến trúc học sâu, có nhiệm vụ trích xuất các đặc trưng từ hình ảnh đầu vào.

Trong Block 0, quá trình bắt đầu với hình ảnh đầu vào có kích thước 640 x 640 x 3, được đưa vào khái Conv với kích thước kernel là 3, stride là 2 và padding là 1. Kích thước không gian sẽ giảm khi stride=2. Khái Conv này tạo ra feature map kích thước 320 x 320 vì kernel di chuyển với bước nhảy 2 pixel.

Block 2 là một khối C3K2 chứa hai tham số: shortcut và n. Tham số shortcut là một giá trị boolean cho biết liệu khối Bottleneck có sử dụng đường tắt (shortcut) hay không. Nếu shortcut=true, khối Bottleneck bên trong khối C3K2 sẽ sử dụng shortcut; ngược lại thì không. Tham số n xác định số lượng khối Bottleneck trong khối C3K2. Trong Block 2, n được xác định như sau:  $n = 3 * d$  với d là depth\_multiple. Trong khối C3K2, độ phân giải của feature map và số chanel đầu ra không thay đổi.

Trong Block 9, khối SPPF được sử dụng sau lớp tích chập cuối cùng của khối C3K2 trong Backbone. Chức năng chính của khối SPPF là tạo ra biểu diễn feature cố định của đối tượng ở các kích thước khác nhau trong hình ảnh mà không cần thay đổi kích thước hình ảnh hay làm mất thông tin không gian.

**Neck** là nơi phóng to feature map và kết hợp các feature map lấy từ nhiều lớp khác nhau của Backbone.

**Head** chịu trách nhiệm dự đoán các class và bounding box của đối tượng, tạo ra kết quả cuối cùng của mô hình nhận diện đối tượng.

Lớp upsample trong phần Neck tăng gấp đôi kích thước của feature map mà không làm thay đổi số lượng kênh đầu ra.

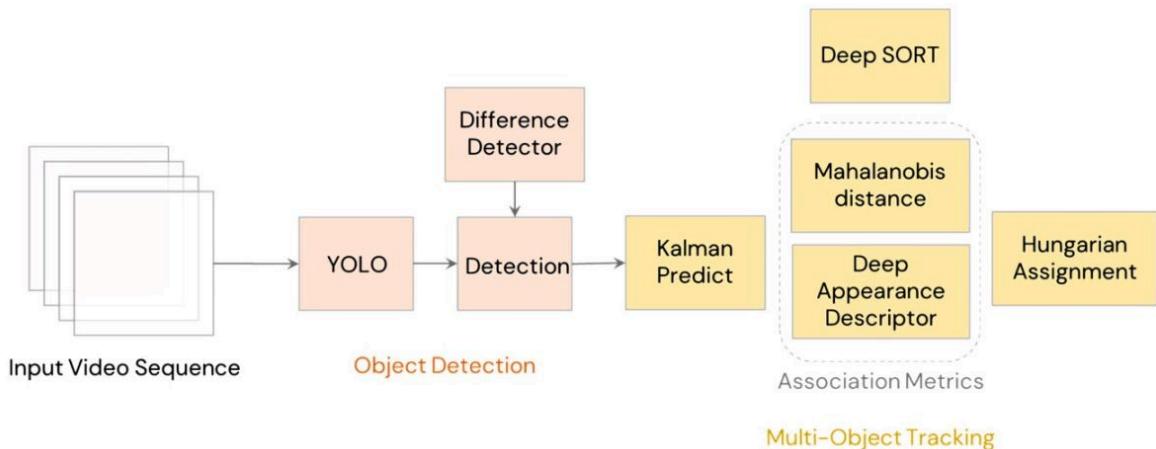
Khối Concat tổng hợp các kênh đầu ra của các khối được ghép nối mà không làm thay đổi độ phân giải.

Khối Detect đầu tiên trong phần Head chuyên phát hiện các đối tượng nhỏ từ khối C3K2 trong Block 15.

Khối Detect thứ hai trong phần Head chuyên phát hiện các đối tượng kích thước trung bình từ khối C3K2 trong Block 18.

Khối Detect thứ ba trong phần Head chuyên phát hiện các đối tượng nhỏ từ khối C3K2 trong Block 21.

## 2.7 SORT (SIMPLE ONLINE AND REALTIME TRACKING)



*Hình 2.18 Multi-Object Tracking*

SORT (Simple Online and Realtime Tracking) là một phương pháp tiếp cận thực dụng cho bài toán theo dõi đa đối tượng, tập trung vào các thuật toán đơn giản và hiệu quả. Trong lĩnh vực phát hiện và theo dõi vật thể dưới nước, việc lựa chọn phương pháp SORT mang lại nhiều lợi ích đáng kể về hiệu quả và ứng dụng thực tế. SORT là một thuật toán theo dõi trực tuyến và thời gian thực, được thiết kế để cân bằng giữa độ chính xác và tốc độ, đặc biệt phù hợp cho các hệ thống có tài nguyên tính toán hạn chế. Đối với môi trường dưới nước, nơi các yếu tố như ánh sáng yếu, độ mờ cao, và nhiễu động môi trường thường gây khó khăn cho việc theo dõi, SORT có khả năng xử lý dữ liệu nhanh chóng, đồng thời giảm thiểu sự phức tạp trong việc tính toán.

Một trong những ưu điểm nổi bật của SORT là cơ chế sử dụng Kalman Filter kết hợp với thuật toán phân cụm Hungarian để tối ưu hóa việc gán các khung hình. Trong môi trường dưới nước, các vật thể thường chuyển động không đều hoặc bị che khuất một phần bởi bùn đất hoặc tảo biển. SORT cho phép duy trì theo dõi liên tục bằng cách dự đoán vị trí vật thể trong khung hình tiếp theo, ngay cả khi tạm thời mất tín hiệu trực tiếp. Khả năng này đặc biệt quan trọng khi cần giám sát lâu dài hoặc theo dõi các vật thể di chuyển ở tốc độ khác nhau, chẳng hạn như sinh vật biển hoặc khi phải xử lý thông tin từ các hình ảnh hoặc video có chất lượng thấp.

### 2.7.1 Giải thuật Hungary

Giải pháp thuật Hungary, một trong những thuật toán tối ưu hóa nổi tiếng nhất trong lý thuyết đồ thị, được thiết kế để giải bài toán phân công tối ưu trên đồ thị hai phía (đồ thị lưỡng cực). Đây là hiệu quả hoạt động thuật toán cho công việc giải quyết bài toán phân công (bài toán phân công), trong đó mục tiêu là phân bổ chi phí tối ưu giữa các nút tập hợp. Trong lĩnh vực **object tracking**, Thuật toán Hungary đóng vai trò quan trọng trong công việc phân bổ các đối tượng được phát hiện từ khung hình này sang khung hình tiếp theo, theo dõi chính xác và liên tục các đối tượng. Giải thuật giúp tìm ra phương án phân bổ, điều này rất quan trọng khi cần theo dõi chính xác nhiều đối tượng trong các khung hình liên tiếp. Ví dụ, trong môi trường phức tạp giám sát dưới nước, nơi có nhiều đối tượng có thể xuất hiện, biến mất hoặc giao nhau, giải thuật Hungary giúp hệ thống theo dõi duy trì tính ổn định và chính xác cao.

Trong bối cảnh theo dõi đối tượng, bài toán có thể được hiểu là xác định mối quan hệ giữa các đối tượng đã được phát hiện (track) ở khung hình trước và các phát hiện (detections) mới trong khung hình hiện tại. Thực hiện thay đổi vị trí, kích thước hoặc hiển thị các đối tượng trong video, việc phân công công việc này không đơn giản. Giải thuật Hungary giúp tìm ra sự hợp lý tốt nhất bằng cách giảm thiểu chi phí, đảm bảo rằng quá trình phân bổ là tối ưu. Ta có thể mô hình bài toán:

$$\sum_{i=1}^{|N|} \sum_{j=1}^{|M|} c_{ij} x_{ij} \rightarrow \min \quad (1)$$

- Tập N: Các đối tượng (track) đang theo dõi từ khung hình trước (t - 1).
- Tập M: Các phát hiện (detections) mới trong khung hình hiện tại (t).
- Ma trận chi phí  $C = [c_{ij}]$  trong đó  $c_{ij}$  là chi phí gán đối tượng  $i \in N$  cho phát hiện  $j \in M$

Mỗi đối tượng chỉ có thể được gán cho tối đa một phát hiện:

$$\sum_{j=1}^{|M|} x_{ij} \leq 1, \forall i \in N \quad (2)$$

Mỗi phát hiện chỉ có thể được gán cho tối đa một đối tượng:

$$\sum_{i=1}^{|N|} x_{ij} \leq 1, \forall j \in M \quad (3)$$

Các giá trị  $x_{ij}$  là nhị phân:

$$x_{ij} = \{0, 1\}, \forall i, j \quad (4)$$

Thuật toán Hungarian được sử dụng để tìm ra phép gán tối ưu, gán các phát hiện cho các đối tượng theo dõi hiện có dựa trên khoảng cách IOU. Ma trận chi phí được xây dựng, trong đó mỗi phần tử đại diện cho khoảng cách IOU giữa một detection và một track. Với  $b_i, b_j$  là bounding boxes của track i và detection j, ta có:

$$c_{ij} = 1 - IoU(b_i, b_j) \quad (5)$$

Giá trị IOU (Intersection over Union) là một chỉ số đo lường mức độ trùng khớp giữa vùng dự đoán (Predicted Bounding Box) và vùng thực tế (Ground Truth Bounding Box). Giá trị IOU dao động từ **0** đến **1**. IOU được tính bằng:

$$IOU = \frac{\text{Diện tích giao nhau giữa hai bounding box (Intersection)}}{\text{Tổng diện tích của hai bounding box trừ đi phần giao nhau (Union)}}$$

### 2.7.2 Kalman Filter

Kalman Filter là một thuật toán tối ưu được thiết kế để ước lượng trạng thái của một hệ thống động trong thời gian thực bằng cách kết hợp thông tin từ các phép đo không hoàn hảo (do nhiễu) với mô hình dự đoán trạng thái. Trong bối cảnh của SORT, Kalman Filter đóng vai trò cốt lõi trong việc dự đoán và điều chỉnh vị trí của các vật thể qua từng khung hình, đặc biệt trong các môi trường phức tạp như theo dõi vật thể dưới nước, nơi dữ liệu có thể bị nhiễu hoặc không liên tục.

Kalman Filter trong SORT hoạt động dựa trên hai bước chính: **dự đoán (prediction)** và **cập nhật (update)**. Trong bước dự đoán, SORT sử dụng mô hình vận tốc không đổi tuyến tính để dự đoán chuyển động của đối tượng giữa các khung hình. Mô hình này giả định rằng đối tượng di chuyển với vận tốc không đổi, độc lập với các đối tượng khác và chuyển động của camera. Mô hình ước lượng trạng thái tiếp theo của vật thể, chẳng hạn như vị trí và vận tốc, dựa trên trạng thái hiện tại và dữ liệu lịch sử. Điều này rất quan trọng trong các tình huống mà vật thể bị che khuất tạm thời hoặc khi tín hiệu từ cảm biến không rõ ràng, một vấn đề phổ biến trong môi trường dưới nước do ảnh hưởng của nhiễu động ánh sáng và chất lượng hình ảnh kém. Bước cập nhật cho phép Kalman Filter điều chỉnh dự đoán ban đầu bằng cách kết hợp với các phép đo thực tế từ mô hình detection. Trạng thái của mỗi đối tượng (State Vector) được biểu diễn bằng một vectơ trạng thái bao gồm:

$$x = [u, v, s, r, u', v', s']^T \quad (1)$$

- $u, v$ : tọa độ tâm của đối tượng (center coordinates).
- $s$ : diện tích của bounding box.
- $r$ : tỉ lệ khung hình (aspect ratio) của bounding box.
- $u', v', s'$ : tốc độ thay đổi tương ứng của các tham số trên.

$$\begin{bmatrix} x_k \\ y_k \\ s_k \\ r_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{s}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ s_{k-1} \\ r_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \dot{s}_{k-1} \end{bmatrix} + u_{k-1}$$

Quan sát được cung cấp từ detector (Measurement Vector):

$$z = [u, v, s, t]^T \quad (2)$$

Mô hình chuyển trạng thái (State Transition Model) dự đoán trạng thái tiếp theo dựa trên trạng thái hiện tại:

$$x_{k+1} = Ax_k + w_k \quad (3)$$

- A: Ma trận chuyển trạng thái
- $w_k$ : nhiễu hệ thống (process noise).

Ma trận A thường được thiết lập để mô phỏng chuyển động tuyến tính.

Quan hệ giữa trạng thái và quan sát (Measurement Model):

$$z_k = Hx_k + v_k \quad (4)$$

- H: Ma trận đo đạc (thường là ma trận đơn vị để trích xuất  $u, v, s, t$ ).
- $v_k$ : nhiễu đo đạc (measurement noise).

**Prediction (Dự đoán)** Kalman Filter sử dụng trạng thái hiện tại để dự đoán trạng thái tiếp theo

$$\hat{x}_{k,k-1} = A\hat{x}_{k-1,k-1} \quad (5)$$

$$P_{k,k-1} = AP_{k-1,k-1}A^T + Q \quad (6)$$

- $\hat{x}_{k,k-1}$ : trạng thái dự đoán.
- $P_{k,k-1}$ : hiệp phương sai dự đoán.
- $Q$ : ma trận nhiễu.

**Update (Cập nhật)** khi có quan sát từ detector, Kalman Filter cập nhật trạng thái dự đoán.

$$K_k = P_{k,k-1}H^T(HP_{k,k-1}H^T + R)^{-1} \quad (7)$$

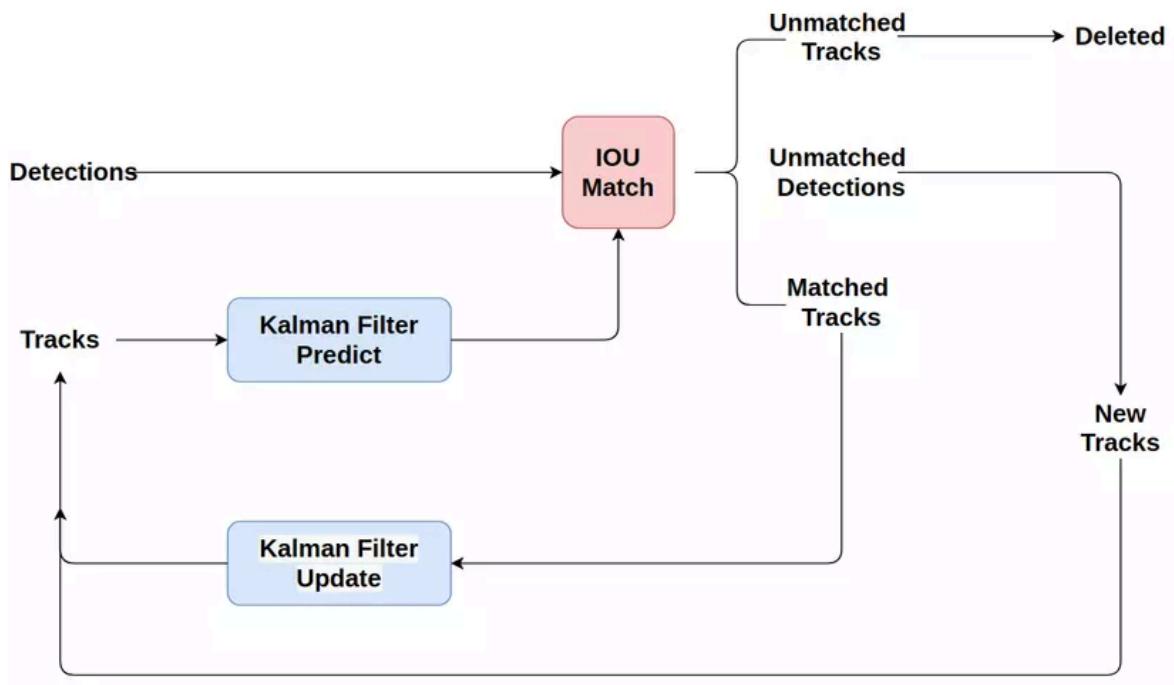
$$\hat{x}_{k,k} = \hat{x}_{k,k-1} + K_k(z_k - H\hat{x}_{k,k-1}) \quad (8)$$

$$P_{k,k} = (I - K_k H)P_{k,k-1} \quad (9)$$

- $K_k$ : Kalman Gain.
- $z_k$ : quan sát mới từ detector.

Liên quan đến khả năng theo dõi thời gian thực của SORT, Kalman Filter đóng vai trò đảm bảo rằng các tính toán dự đoán và cập nhật được thực hiện với độ trễ thấp, tối ưu hóa hiệu suất hệ thống.

### 2.7.3 Mô tả SORT



Hình 2.19 Mô tả SORT

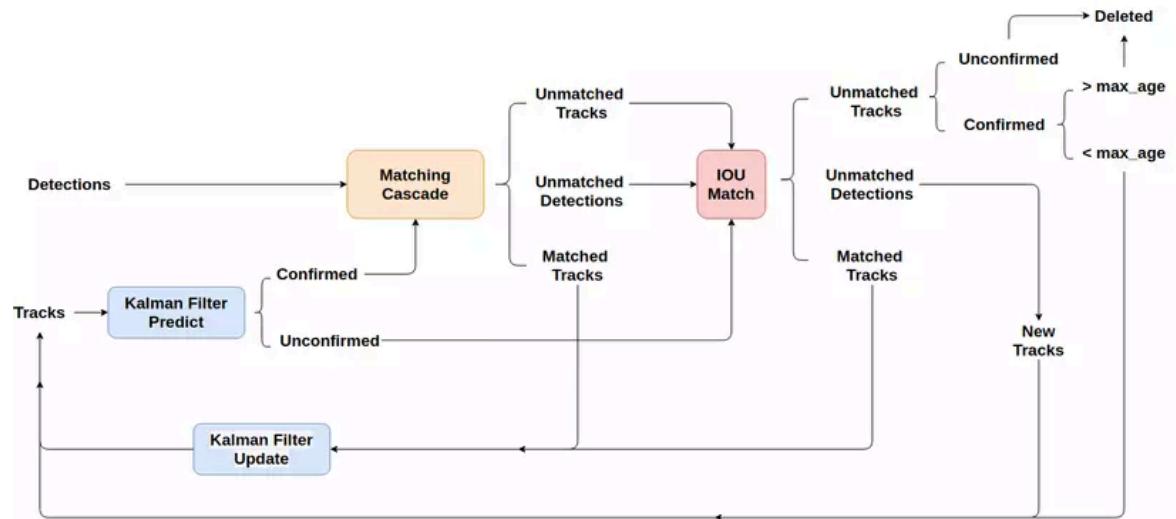
Bước 1: Khi một detection mới xuất hiện trong khung hình bất kỳ track nào có độ trùng lặp nhỏ hơn IOUmin với tất cả các đối tượng hiện có đều được coi là một đối tượng chưa được track (Unmatched Detections)

Bước 2: Track mới (New Tracks) khởi tạo dựa trên thông tin bounding box do Unmatched Detections cung cấp, với vận tốc được đặt bằng 0 và hiệp phương sai của thành phần vận tốc được khởi tạo với giá trị lớn để phản ánh sự không chắc chắn. Track mới đi qua bộ lọc Kalman để dự đoán (Kalman Filter Predict) bounding box

Bước 3: Xử lí, phân loại các detection (IOU Match). Xây dựng ma trận chi phí. Thuật toán Hungarian được sử dụng để tìm ra phép gán tối ưu, gán các phát hiện cho các đối tượng theo dõi hiện có dựa trên khoảng cách IOU.

Bước 4: Sử dụng Kalman filter update để update những detection đã được liên kết với track. Nếu không có detection nào được liên kết với track, trạng thái của nó chỉ được dự đoán mà không cần hiệu chỉnh bằng cách sử dụng mô hình vận tốc tuyến tính. Những detection không được liên kết với track thì Delete Track.

#### 2.7.4 Deep SORT



Hình 2.20 Mô tả Deep SORT

Deep SORT, do Nicolai Wojke và Alex Bewley phát triển, được xây dựng trên nền tảng SORT nhằm khắc phục các hạn chế liên quan đến số lượng lớn ID switches. Phương pháp này áp dụng công nghệ deep learning để trích xuất đặc trưng đối tượng, từ đó cải thiện độ chính xác trong quá trình gán dữ liệu. Bên cạnh đó, Deep SORT còn giới thiệu chiến lược liên kết Matching Cascade, giúp nâng cao hiệu quả trong việc nhận diện lại các đối tượng đã biến mất sau một khoảng thời gian. Trong SORT đã giải quyết vấn đề data association dựa trên thuật toán Hungarian. Tuy nhiên, việc liên kết không chỉ dựa trên IOU mà còn quan tâm đến các yếu tố khác: khoảng cách của detection và track (xét tính tương quan trong không gian vector) và khoảng cách cosine giữa 2 vector đặc trưng được trích xuất từ detection và track.

#### 2.7.4.1 Kiến trúc trích xuất đặc trưng

Để lấy các vector đặc trưng, DEEP SORT áp dụng một mạng nơ-ron để thực hiện việc trích xuất. Các mạng nơ-ron truyền thống thường được phát triển theo xu hướng gia tăng độ sâu để nâng cao hiệu suất, nhằm tạo ra một bộ trích xuất đặc trưng hiệu quả. Tuy nhiên, chiến lược này có nhược điểm là thời gian suy luận (inference) sẽ tăng lên đáng kể. Điều này hoàn toàn trái ngược với mục tiêu của DEEP SORT, vốn ưu tiên tốc độ và hiệu quả trong xử lý thời gian thực. Do đó, DEEP SORT sử dụng mạng nơ-ron nông (Shallow Neural Network), đặc biệt là Wide Residual Network (WRN). Kiến trúc này chỉ bao gồm một số lớp rất ít (16 lớp), giúp giảm thiểu thời gian suy luận.

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
$\ell_2$ normalization		128

Hình 2.21 Mạng WRN trong Deep SORT

#### 2.7.4.2 Khoảng cách Mahalanobis và Khoảng cách cosine

Để cải thiện quá trình liên kết dữ liệu, DEEP SORT có thêm 2 bộ đo mới  
**Khoảng cách Mahalanobis:**

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \quad (1)$$

- $(y_j, S_i)$  là giá trị kì vọng và ma trận hiệp phương sai của biến ngẫu nhiên track thứ  $i$
- $d_i$  là giá trị của detection thứ  $j$

Từ trên ta có thể xây dựng được ma trận chi phí riêng cho khoảng cách Mahalanobis giữa track và detection. Để loại trừ các liên kết không chắc chắn bằng cách lập ngưỡng khoảng cách Mahalanobis ở khoảng tin cậy 95% được tính từ phân phối  $\chi^2$

$$b_{ij}^{(1)} = 1[d^{(1)}(i,j) \leq t^{(1)}] \quad (2)$$

Với  $t^{(1)} = 9.4877$

#### Khoảng cách cosine:

Với mỗi detection, đặc trưng  $r_j$  được trích xuất với  $\|r_j\| = 1$ . Với mỗi track, một danh sách với độ dài khoảng 100 được sử dụng để lưu trữ đặc trưng của 100 track gần nhất:  $R_k = \{r_k^{(i)}\}_{k=1}^{L_k}$ ,  $L_k = 100$ . Khi đó, độ đo mới giữa track và detection được tính bằng khoảng cách cosine:

$$d^{(2)}(i,j) = \min(1 - r_j^T r_k^{(i)} | r_k^{(i)} \in R_i) \quad (3)$$

Và

$$b_{ij}^{(2)} = 1[d^{(2)}(i,j) \leq t^{(2)}] \quad (4)$$

Với  $t^{(2)}$  được lựa chọn tùy vào tập dữ liệu sử dụng.

**Tóm lại**, khoảng cách Mahalanobis cung cấp thông tin về vị trí của đối tượng dựa trên chuyển động hiện tại, giúp ích cho các dự đoán ngắn hạn. Trong khi đó, khoảng cách cosine lại tập trung vào đặc trưng của đối tượng, đặc biệt hữu ích cho các dự đoán dài hạn hoặc khi đối tượng khó phân biệt. Bằng cách kết hợp hai độ đo này với trọng số phù hợp, DEEP SORT tạo ra một độ đo mới:

$$c_{ij} = \lambda d^{(1)}(i,j) + (1 - \lambda)d^{(2)}(i,j) \quad (5)$$

Với:

$$b_{i,j} = \prod_{m=1}^2 b_{i,j}^{(m)} \quad (6)$$

Giá trị  $\lambda$  có thể được điều chỉnh tùy theo trường hợp sử dụng để cân bằng giữa 2 độ đo.

#### 2.7.4.3 Matching Cascade

Matching Cascade được áp dụng để cải thiện độ chính xác trong việc liên kết các đối tượng. Khi đối tượng biến mất trong thời gian dài, dẫn đến sự gia tăng đáng kể độ không chắc chắn trong bộ lọc Kalman. Nếu dự đoán không được cập nhật, phương sai của phân phối chuẩn sẽ tiếp tục mở rộng, làm cho giá trị khoảng cách Mahalanobis giữa các điểm xa giá trị kỳ vọng. **Matching Cascade** thực hiện việc xử lý từng track từ các khung hình trước, xây dựng ma trận chi phí và giải bài toán phân công theo từng tầng. Chi tiết thuật toán được minh họa cụ thể thông qua mã giả dưới đây:

---

Listing 1 Matching Cascade

---

```

Input: Track indices  $\mathcal{T} = \{1, \dots, N\}$ , Detection indices  $\mathcal{D} = \{1, \dots, M\}$ , Maximum age  $A_{\max}$ 
1: Compute cost matrix  $C = [c_{i,j}]$ 
2: Compute gate matrix  $B = [b_{i,j}]$ 
3: Initialize set of matches  $\mathcal{M} \leftarrow \emptyset$ 
4: Initialize set of unmatched detections  $\mathcal{U} \leftarrow \mathcal{D}$ 
5: for  $n \in \{1, \dots, A_{\max}\}$  do
6:   Select tracks by age  $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$ 
7:    $[x_{i,j}] \leftarrow \text{min cost matching } (C, \mathcal{T}_n, \mathcal{U})$ 
8:    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j) \mid b_{i,j} \cdot x_{i,j} > 0\}$ 
9:    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$ 
10: end for
11: return  $\mathcal{M}, \mathcal{U}$ 

```

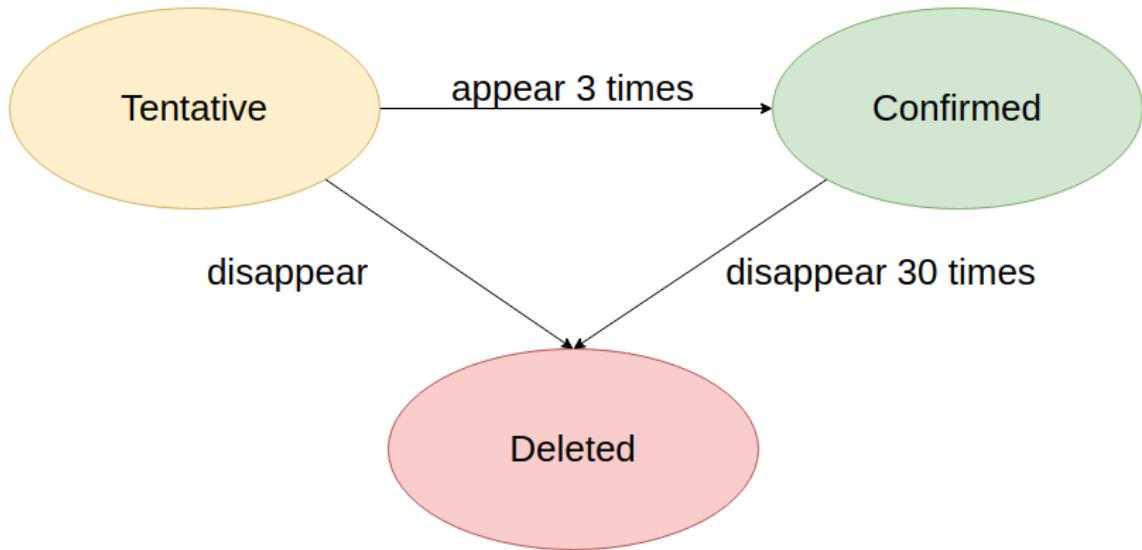
---

Hình 2.22 Mã giả Matching Cascade

#### 2.7.4.4 Track Lifecycle

Deep SORT quản lí vòng đợi của 1 track dựa trên 1 biến trạng thái với 3 giá trị (tentative, confirmed, deleted)

- Các trạng thái khi mới khởi tạo sẽ được gán giá trị thăm dò (tentative).
- Nếu giá trị này vẫn được duy trì trong 3 frame tiếp theo, trạng thái sẽ chuyển từ thăm dò sang xác nhận (confirmed).
- Các track có trạng thái confirmed sẽ được tiếp tục theo dõi, và ngay cả khi bị mất dấu, DEEP SORT vẫn duy trì theo dõi trong tối đa 30 frame tiếp theo (max\_age).
- Ngược lại, nếu mất dấu trước khi đủ 3 frame, trạng thái của track sẽ bị xóa khỏi hệ thống theo dõi (deleted).



Hình 2.23 Track lifecycle

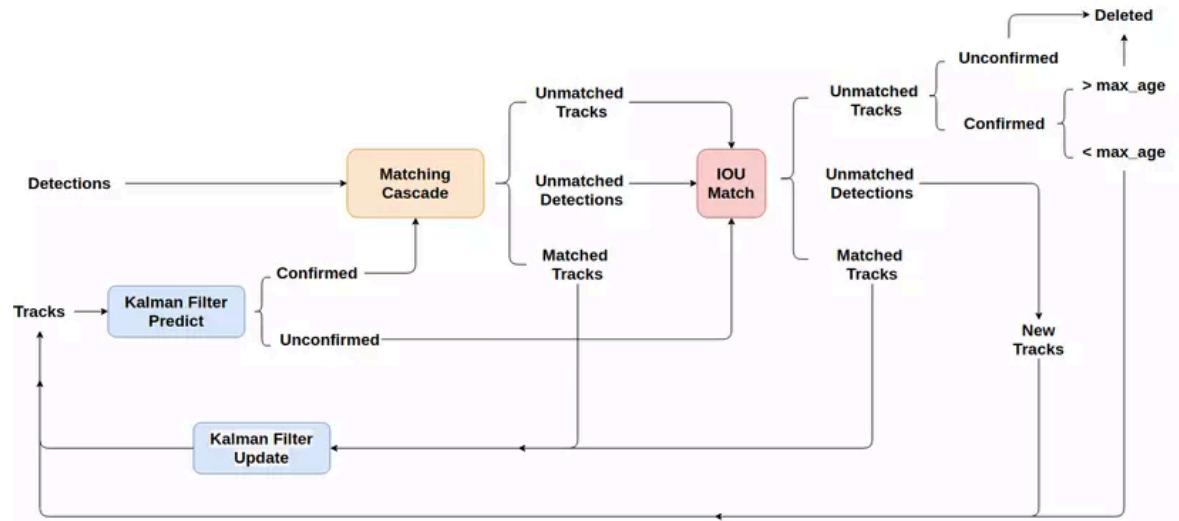
### 2.7.5 Mô tả DEEP SORT

Do SORT đã mô hình hóa bài toán khá ổn, bên DEEP SORT, các thành phần trong trạng thái track không thay đổi quá nhiều. Mỗi track bao gồm 8 thành phần:

$$x = [u, v, \gamma, h, u', v', \gamma', h']^T \quad (1)$$

- $u, v$ : tọa độ tâm của đối tượng (center coordinates).
- $\gamma$ : tỉ lệ khung hình
- $h$ : là chiều cao của bounding box
- $u', v', \gamma', h'$ : tốc độ thay đổi tương ứng của các tham số trên.

Các mô hình trạng thái và mô hình quan sát của deep SORT hầu như giống với SORT



Hình 2.24 Mô tả Deep SORT

Luồng xử lí của Deep SORT:

Bước 1: Detection mới xuất hiện

Bước 2: Deep SORT sử dụng Kalman Filter để dự đoán các trạng thái track mới dựa trên các track trong quá khứ.

Bước 3: Sử dụng những track đã được xác nhận, tiến hành đưa vào matching cascade nhằm liên kết với các detection phát hiện được dựa trên độ đo về khoảng cách và đặc trưng.

Bước 4: Các track và các detection chưa được liên kết sẽ được đưa đến 1 lớp lọc tiếp theo. Sử dụng giải thuật Hungarian giải bài toán phân công với ma trận chi phí IOU để liên kết lần 2

Bước 5: Xử lí, phân loại các detection và các track

Bước 6: Sử dụng Kalman filter để hiệu chỉnh lại giá trị của track từ những detection đã được liên kết với track và khởi tạo các track mới.

## 2.8 Kết luận chương 2

Trong nghiên cứu này, chúng tôi đã lựa chọn sử dụng công nghệ deep learning để xác định vật thể dưới nước, dựa trên các ưu điểm nổi bật của nó trong việc xử lý các bài toán phức tạp và phi tuyến tính. Deep learning vượt trội so với các phương pháp truyền thống nhờ khả năng tự động học đặc trưng từ dữ liệu, không yêu cầu các bước tiền xử lý thủ công phức tạp. Trong môi trường dưới nước, nơi các yếu tố như ánh sáng, độ mờ và nhiễu ảnh hưởng lớn đến việc nhận dạng, deep learning chứng minh hiệu quả vượt trội với khả năng trích xuất đặc trưng mạnh mẽ và phù hợp trong điều kiện thay đổi đa dạng.

Mô hình YOLO được chọn vì đây là phiên bản cải tiến mới nhất của dòng YOLO (You Only Look Once), mang lại sự cân bằng tối ưu giữa tốc độ và độ chính xác. YOLO được thiết kế để cải thiện hiệu suất phát hiện vật thể với cấu trúc nhẹ, tốc độ xử lý nhanh và khả năng phát hiện vật thể nhỏ, phù hợp với đặc thù của môi trường dưới nước, nơi các vật thể thường có kích thước nhỏ và khó quan sát.

DEEPSORT được tích hợp để giải quyết bài toán theo dõi vật thể (object tracking) sau khi phát hiện. Công cụ này sử dụng kết hợp giữa bộ lọc Kalman và metric khoảng cách Mahalanobis, giúp theo dõi các vật thể một cách mượt mà và chính xác ngay cả khi chúng di chuyển hoặc biến mất trong thời gian ngắn. Với đặc điểm của môi trường dưới nước, nơi vật thể dễ bị mất dấu do các yếu tố như dòng nước hoặc che khuất, DEEPSORT đảm bảo khả năng duy trì liên tục việc theo dõi, đồng thời tối ưu hóa hiệu quả tính toán.

Tóm lại, sự kết hợp giữa deep learning, mô hình YOLO11 và DEEPSORT tạo nên một hệ thống toàn diện, mạnh mẽ và hiệu quả để giải quyết bài toán nhận dạng và theo dõi vật thể dưới nước, đáp ứng được các yêu cầu về độ chính xác, tốc độ và tính ổn định trong môi trường khắc nghiệt.

## CHƯƠNG 3: THIẾT KẾ VÀ THỰC HIỆN

### 3.1 Yêu cầu thiết kế

- Hệ thống có thời gian inference (suy luận) <200ms, tức là một giây phải xử lý tối thiểu được 5 frame. Mô hình sau khi train phải có độ chính xác (precision) lớn hơn 0.8 và khi hoạt động thực tế có thể hoàn thành tác vụ với ngưỡng tin cậy (confidence threshold) là 0.7
- Nhận diện được các vật thể dưới nước với các class: cá (fish), tôm (shrimp), cua(crab).
- Có khả năng stream không dây phục vụ các tác vụ detect dưới nước ví dụ như Phuong tiện dưới nước không người lái UUV (*Unmanned Underwater Vehicles*)
- Có khả năng tracking và đánh số id cho object khi detect.

### 3.2 Thực hiện

#### 3.2.1. Chuẩn bị dataset

Để chuẩn bị dataset cho việc huấn luyện mô hình, bước đầu tiên là thu thập dữ liệu từ các nguồn phù hợp. Ở đây chủ yếu là các video và hình ảnh dưới nước. Ta có thể tìm kiếm với các từ khóa *underwater object*, *underwater footage*. Ở đồ án này ta dùng video tự quay và sử dụng một vài video có trên internet.

Sau khi thu thập, dữ liệu cần được gắn nhãn (label), xử lý và phân loại sao cho phù hợp. Các bước xử lý dữ liệu có thể bao gồm loại bỏ dữ liệu lỗi, xử lý missing values, chuẩn hóa hoặc chuẩn hóa giá trị, và phân chia dữ liệu thành các tập huấn luyện, kiểm tra và xác nhận.. Tôi dùng **Roboflow**, một công cụ Data Labeling và có thể đơn giản hóa các bước này.

Video tự quay	Video có trên internet
	

Bảng 3.1 So sánh giữa video tự quay và có sẵn

Tổng số hình ảnh đã được đánh nhãn là 1319 với thông tin:

Class	Count	Màu
fish	3.898	Đỏ
crab	302	Xanh lá
shrimp	948	Đỏ

Bảng 3.2 Thông tin về các class

Để cải thiện chất lượng mô hình, dữ liệu cũng có thể được tăng cường thông qua các kỹ thuật như xoay, thay đổi kích thước, hoặc thêm nhiều đối với hình ảnh. Ở đây Roboflow giúp ta thực hiện các bước đó mà không cần thông qua viết chương trình với các thông số sau:

- Tiền xử lý

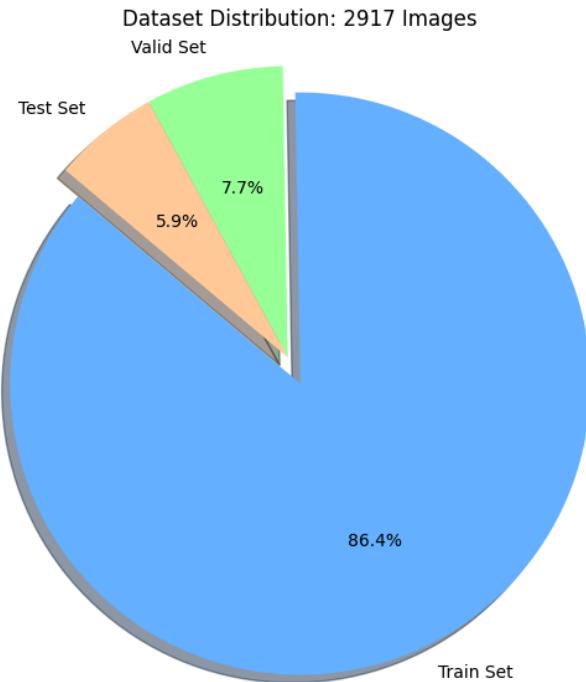
Cắt ảnh: 25-75% Ngang, 25-75% Dọc

- Tăng cường

Lật ảnh: Ngang

Làm mờ: Up to 3px

Sau bước tăng cường dữ liệu thì ta có được bộ dataset với train set 2520 hình ảnh (86.4%), valid set 224 hình ảnh (7.7%), test set 173 hình ảnh (5.9%). Tổng 2917 hình ảnh.

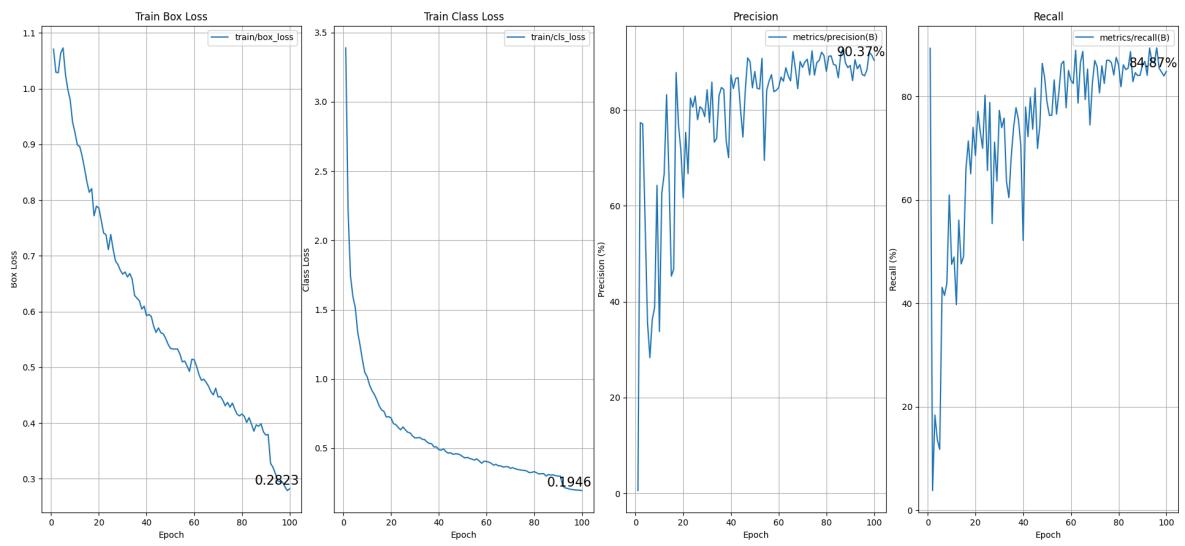


Hình 3.1 Tỷ lệ train, valid, test trong dataset

### 3.2.2 Train mô hình

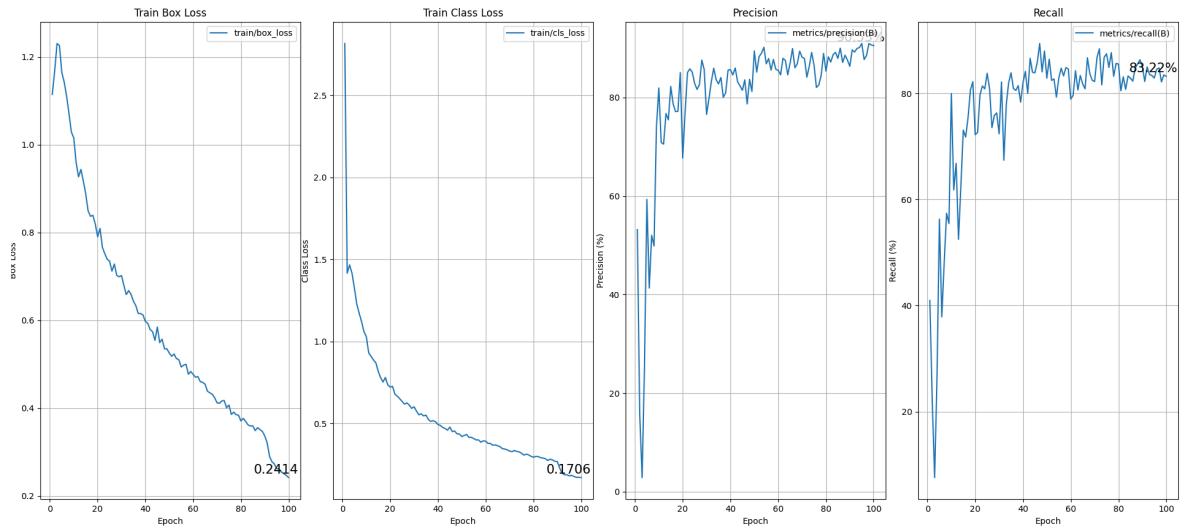
Ta thực hiện train nhiều variants để tìm ra variants có thời gian inference phù hợp với yêu cầu thiết kế. Vì thời gian 200ms là khá ngắn nên ta chọn variants nhỏ với ít parameter để phù hợp: n, s, m. Đây là kết quả train với 100 epoch:

**Nano** với kết quả cuối cùng là:



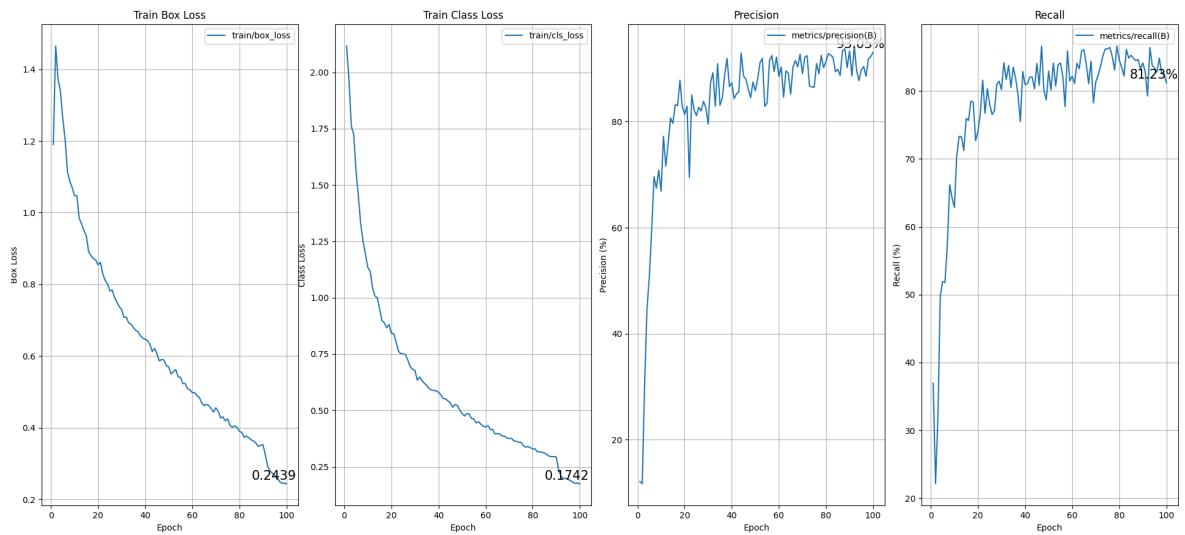
Hình 3.2 Kết quả train với variant nano

**Small** với kết quả cuối cùng là:



Hình 3.3 Kết quả train với variant small

**Medium** với kết quả cuối cùng là:



Hình 3.4 Kết quả train với variant medium

**box\_loss:** box loss được tính bằng cách so sánh sự khác biệt giữa các tọa độ dự đoán và thực tế của các bounding box.

**cls\_loss:** Đây là sự khác biệt giữa các predict class và class thực tế của đối tượng.

**dfl\_loss:** đo lường sự khác biệt giữa các đặc trưng dự đoán và các đặc trưng thực tế.

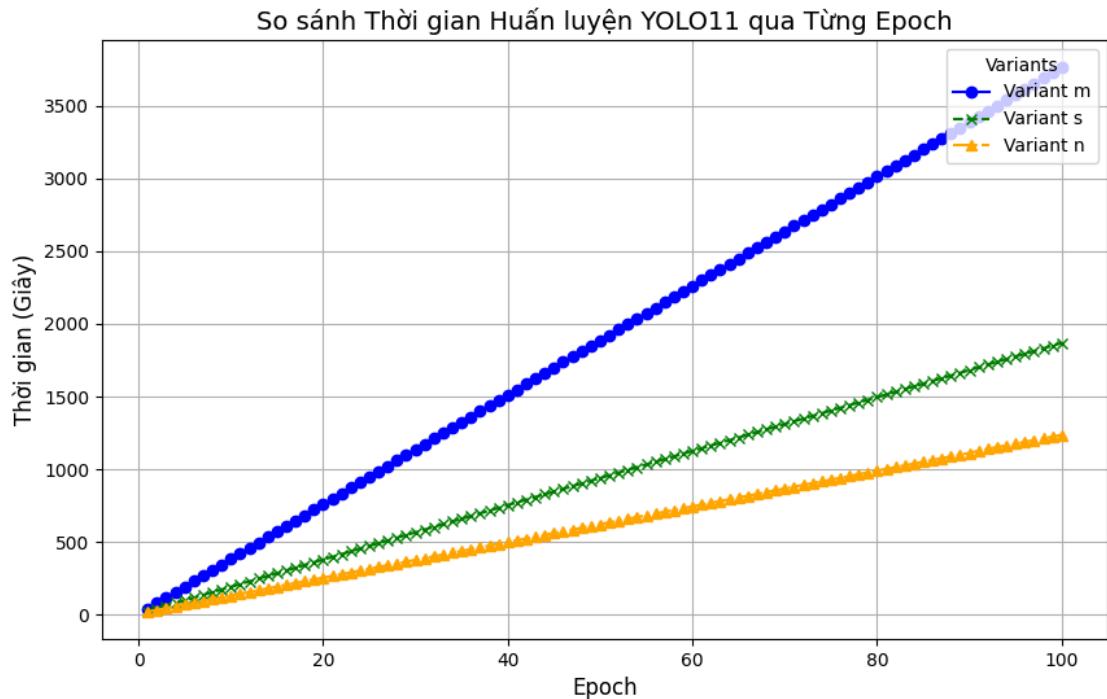
**metrics/precision(B):** Độ chính xác (precision) của mô hình trong việc phát hiện đối tượng, tính bằng tỷ lệ số lượng đối tượng được mô hình dự đoán chính xác so với tổng số đối tượng được mô hình dự đoán.

**metrics/recall(B):** Độ nhạy (recall) của mô hình, tính bằng tỷ lệ số lượng đối tượng được mô hình dự đoán chính xác so với tổng số đối tượng thực sự có mặt trong ảnh. Recall đo lường khả năng phát hiện của mô hình.

**metrics/mAP50(B):** Độ chính xác trung bình tại IOU (Intersection over Union) = 50% (mAP50) của mô hình. Nó tính trung bình độ chính xác của mô hình trên tất cả các lớp với ngưỡng IOU là 0.5.

**metrics/mAP50-95(B):** Độ chính xác trung bình tại IOU = 50% đến IOU = 95% (mAP50-95) của mô hình, tính toán độ chính xác trung bình trên nhiều ngưỡng IOU khác nhau (từ 0.5 đến 0.95).

Đồ thị **recall** cho ta thấy mô hình với càng nhiều parameter thì sẽ cho khả năng hội tụ càng cao nhưng đồng thời cũng có sự đánh đổi cho thời gian huấn luyện.



Hình 3.5 So sánh thời gian train

Sau khi train mô hình ta chạy predict để xem time inference và chọn:

	Time inference	Result
Nano	51.0 - 66.0 ms	

Small	122.0 - 153.0 ms	
Medium	255.0 - 269.0 ms	

Bảng 3.3 Time inference

Ở đây ta chọn Nano và Small phù hợp với yêu cầu thiết kế.

### 3.2.3 Triển khai DEEP SORT

Ở phần triển khai DEEP SORT để kết hợp với mô hình đã train được ta cần chú ý các thông số sau:

**max\_iou\_distance:** Tham số này xác định khoảng cách IOU (đề cập ở 2.7.1) tối đa giữa hai bounding box để chúng được coi là "cùng một đối tượng". Nếu IOU giữa hai đối tượng lớn hơn giá trị này, chúng sẽ được coi là một đối tượng duy nhất. Nếu chỉ số này lớn quá thì khi cá bơi nhanh IOU sẽ thấp hơn ngưỡng, nhưng tóm lại di chuyển chậm IOU sẽ cao. Nên ta chọn giá trị phù hợp ở đây là 0.7.

**max\_age:** Tham số này quy định số khung hình tối đa mà một đối tượng có thể không được phát hiện (hoặc bị mất) trước khi nó bị loại khỏi quá trình theo dõi. Nếu đối tượng không xuất hiện trong số khung hình ngưỡng liên tiếp, nó sẽ bị loại. Vì time inference lớn nhất ở variant Small là khoảng 150ms nên số lần (số frame) detect được trong một giây là:  $\frac{1}{0.15} = 6.66$ .

Số frame trống không được detect với một camera 30 fps là:  $\frac{30}{6.66} \approx 5$ .

Ta chọn giá trị phù hợp ở đây là 10 hoặc 15 (gấp 2 hoặc 3) là phù hợp.

**n\_init**: Đây là số khung hình tối thiểu mà một đối tượng phải được phát hiện trước khi DeepSort bắt đầu theo dõi nó. Nếu một đối tượng được phát hiện ít hơn số lần tối thiểu liên tiếp, thuật toán sẽ không bắt đầu theo dõi nó. Tham số này ta chọn phụ thuộc vào môi trường dưới nước có nhiều nhiễu không, ở demo là 3.

**nms\_max\_overlap**: Tham số này liên quan đến Non-Maximum Suppression (NMS), một kỹ thuật được sử dụng để loại bỏ các bounding box chồng lấp nhau. Tham số này xác định tỷ lệ chồng lấp tối đa cho phép giữa các bounding box trước khi NMS loại bỏ chúng. Nếu hai hộp có IOU lớn hơn, chỉ một trong chúng sẽ được giữ lại. Giá trị sử dụng ở đây là 0.5.

```
tracker = DeepSort(  
    max_iou_distance= 0.7,  
    max_age=10,  
    n_init=3,  
    nms_max_overlap=0.5)
```

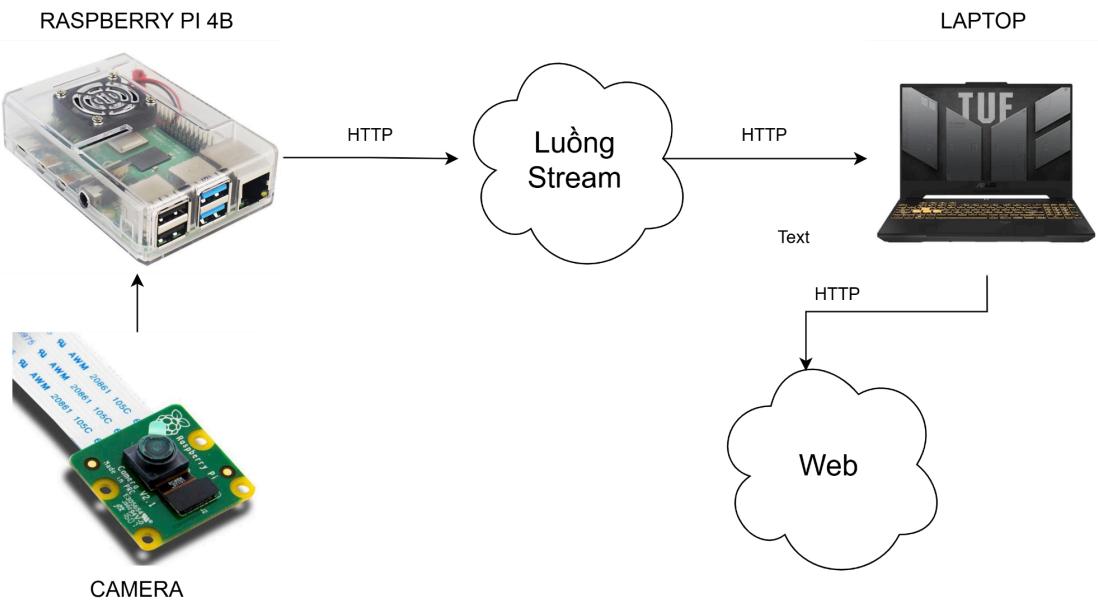
### 3.3 Kết luận chương 3

Chương này đã trình bày chi tiết quá trình thiết kế và triển khai hệ thống nhận diện và theo dõi đối tượng dưới nước. Mô hình được phát triển nhằm đáp ứng yêu cầu thời gian inference dưới 200ms và có khả năng nhận diện các đối tượng như cá, tôm, và cua, đồng thời hỗ trợ việc tracking và gán ID cho các đối tượng khi phát hiện.

Quá trình chuẩn bị dataset, bao gồm thu thập dữ liệu từ các video dưới nước và tăng cường dữ liệu thông qua các kỹ thuật như xoay, thay đổi kích thước và thêm nhiễu, đã giúp tạo ra một bộ dữ liệu huấn luyện chất lượng. Sau đó, mô hình được huấn luyện với các biến thể nhỏ (Nano, Small, Medium), với kết quả tối ưu đạt được ở các biến thể Nano và Small, phù hợp với yêu cầu về thời gian inference. Việc triển khai DeepSORT giúp cải thiện khả năng tracking đối tượng, đảm bảo theo dõi chính xác trong môi trường động.

Cuối cùng, hệ thống đã được triển khai trên Raspberry Pi, sử dụng Flask để stream video và hiển thị kết quả nhận diện trực tiếp trên web. Hệ thống có thể thực hiện nhiệm vụ nhận diện đối tượng trong môi trường dưới nước với độ chính xác cao, đáp ứng các yêu cầu về hiệu suất và khả năng hoạt động không dây.

## CHƯƠNG 4: KẾT QUẢ THỰC HIỆN



Hình 3.6 Tổng quan hệ thống

Mô tả hoạt động của sơ đồ hệ thống như sau:

### Camera thu thập hình ảnh:

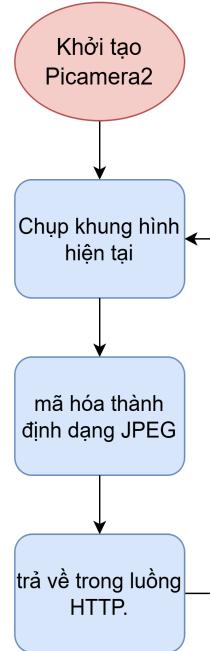
- Model: Camera Module 77 Degree FoV Adjustable Focus OV5647 5MP for Raspberry Pi
- Độ phân giải: 5MP
- Angle of View (diagonal): 65 degree
- Kích thước: 25x24mm

Camera sẽ thu thập hình ảnh hoặc video từ môi trường xung quanh. Hình ảnh này sẽ được xử lý để trích xuất các khung hình với độ phân giải đã chỉ định (1296x972).

### Raspberry Pi phát luồng stream:

- Model: Raspberry Pi 4B 4GB
- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 4GB LPDDR4-2400 SDRAM (depending on model)

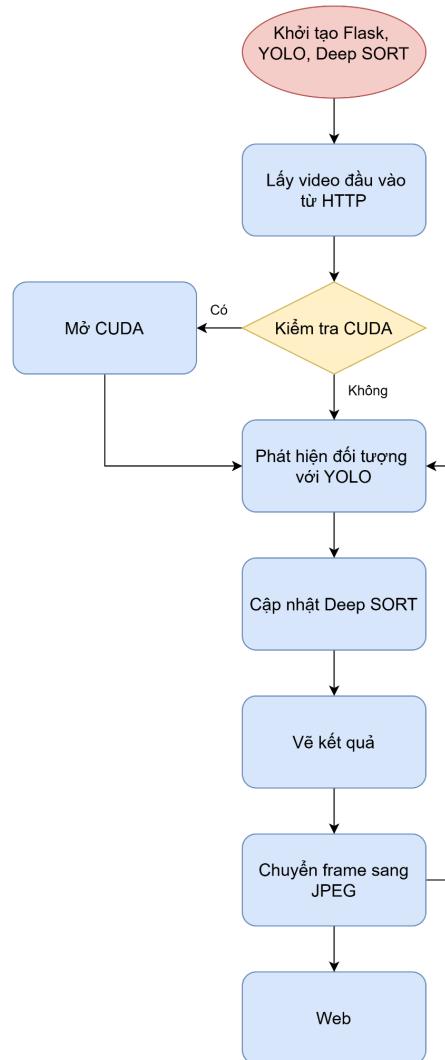
Raspberry Pi 4 sẽ nhận các khung hình từ camera và chuyển tiếp chúng dưới dạng luồng video (stream) tới một máy chủ web thông qua Flask app. Flask sẽ chịu trách nhiệm stream video dưới dạng JPEG frames đến các endpoint ([/video\\_feed](#)) để trình duyệt web có thể nhận và hiển thị.



Hình 3.7 Lưu đồ giải thuật trên raspberry

#### Laptop thực hiện detect:

Laptop nhận các khung hình video từ luồng stream và sử dụng hệ thống phát hiện đối tượng (object detection model) để phân tích các khung hình. Hệ thống này sẽ nhận diện các đối tượng như cá, tôm, hoặc cua trong video.



Hình 3.8 Lưu đồ giải thuật ở sever host

### Gửi kết quả lên web:

Sau khi laptop thực hiện việc nhận diện các đối tượng, kết quả (bao gồm tên đối tượng và tọa độ trong khung hình) sẽ được gửi đến một trang web. Route `/video_feed` sử dụng `Response(generate_frames())` để trả về video dưới dạng một luồng (`multipart/x-mixed-replace`). Video này được phát liên tục tới người dùng qua trình duyệt, nơi kết quả có thể được hiển thị trực tiếp cho người dùng hoặc lưu trữ để phân tích thêm.



*Hình 3.9 Web stream*

## CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Kết quả đạt được

Sau khi tìm hiểu, nghiên cứu, phân tích, thực hiện và kiểm nghiệm trên thực tế, đã

tài đã được những yêu cầu ở mục tiêu đã đề ra:

- Xây dựng được hệ thống nhận diện vật thể dưới nước với các chức năng:
- Cho phép người dùng theo dõi trực tiếp tình hình hiện tại ở dưới nước trên website.
- Bộ phát hiện đối tượng có thể phát hiện ở các điều kiện thời tiết, ánh sáng với tốc độ phù hợp, có thể chạy ổn định chuyển đổi linh hoạt giữa CPU và GPU. Website cũng thực hiện tốt trên hầu hết các trình duyệt web như: Chrome, Edge.

### Nhược điểm

- Do dataset chỉ được gắn label bởi cá nhân nên số lượng chưa đủ nhiều để có thể detect trong nhiều môi trường nước khác nhau. Đặc biệt là môi trường nước có nhiều phù sa. Cần thu thập khá nhiều dữ liệu ảnh sau đó huấn luyện để đạt được sự khai quát hoá cao
- Website còn ít chức năng, giao diện GUI đơn giản.

### Hướng phát triển

- Khắc phục những nhược điểm trên
- Trang website có thể thêm chức năng IoT từ xa như bật tắt đèn, khởi động động cơ trong các tác vụ của uuv, thay đổi độ phân giải của camera để tăng tốc độ tín hiệu.
- Khi khắc phục được các nhược điểm có thể cho thuê model để auto-label, giảm bớt thời gian tác vụ gắn nhãn, train được model tốt hơn
- Tăng số lượng class mà mô hình có thể nhận diện được ví dụ như rùa, sứa, lươn ...

## CHƯƠNG 6: TÀI LIỆU THAM KHẢO

- [1] Vũ Hữu Tiệp's Blog, "Perceptron Learning Algorithm", "Machine Learning cơ bản", 21-Jan-2017. [Online].  
<https://machinelearningcoban.com/2017/01/21/perceptron/>.  
[Truy cập: 28-Nov-2024].
- [2] Vũ Hữu Tiệp's Blog, "Multi-layer Perceptron và Backpropagation", "Machine Learning cơ bản", Feb 24, 2017. [Online].  
<https://machinelearningcoban.com/2017/02/24/mlp/> [Truy cập: 28-Nov-2024].
- [3] N. Tuan, "Bài 6: Convolutional Neural Network," "NTTuan8", [Online].  
<https://nttuan8.com/bai-6-convolutional-neural-network/> [Truy cập: 28-Nov-2024].
- [4] P. D. Khanh, "Transfer Learning," \*Phạm Đình Khanh's Blog\*, 15-Apr-2020.  
[Online]. <https://phamdinhkhanh.github.io/2020/04/15/TransferLearning.html>  
[Truy cập: 28-Nov-2024].
- [5] T. Nguyen, "Paper Explain: RCNet - Kiến trúc FPN Định Cao cho Object Detection," \*Viblo\*, 19-Sep-2022 [Online].  
<https://viblo.asia/p/paper-explain-rcnet-kien-truc-fpn-dinh-cao-cho-object-detection-GAWVp0ZPV05> [Truy cập: 28-Nov-2024].
- [6] "YOLOv11 Model," \*Ultralytics Documentation\*, [Online].  
<https://docs.ultralytics.com/vi/models/yolo11/> [Truy cập: 28-Nov-2024].
- [7] N. Rao, "YOLOv11 Explained: Next-level Object Detection with Enhanced Speed and Accuracy," \*Medium\*, 20-Oct-2020. [Online].  
<https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71> [Truy cập: 28-Nov-2024].
- [8] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft, "Simple Online and Realtime Tracking," \*arXiv\*, 7-Jul-2017. [Online].  
<https://arxiv.org/pdf/1602.00763.pdf> [Truy cập: 28-Nov-2024].
- [9] "Kalman Filter," \*Kalmanfilter.net\*, [Online].  
<https://www.kalmanfilter.net/default.aspx> [Truy cập: 28-Nov-2024].
- [10] "Hungarian Matching," \*Brilliant.org\*. [Online].  
<https://brilliant.org/wiki/hungarian-matching/> [Truy cập: 28-Nov-2024].

- [11] Nicolai Wojke, Alex Bewley, Dietrich Paulus, “Simple Online and Realtime Tracking with a Deep Association Metric,” \*arXiv\*, 21-Mar-2017. [Online]. <https://arxiv.org/abs/1703.07402> [Truy cập: 28-Nov-2024].
- [12] Nicolai Wojke, Alex Bewley, “Deep Cosine Metric Learning for Person Re-Identification,” \*arXiv\*, 2-Dec-2018. [Online]. <https://arxiv.org/abs/1812.00442> [Truy cập: 28-Nov-2024].
- [13] "Mahalanobis Distance," \*Wikipedia\*, [Online]. [https://en.wikipedia.org/wiki/Mahalanobis\\_distance](https://en.wikipedia.org/wiki/Mahalanobis_distance) [Truy cập: 28-Nov-2024].

## **PHỤ LỤC**

[https://github.com/nhutnam116/DoAnTotNghiep\\_1914214](https://github.com/nhutnam116/DoAnTotNghiep_1914214)