

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

—oo—



HỌC SÂU VÀ ỨNG DỤNG TRONG  
BÀI TOÁN ĐÊM CÂY

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

*Chuyên ngành: TOÁN TIN*

Giảng viên hướng dẫn: TS. LÊ HẢI HÀ

Họ và tên sinh viên: LAI ĐỨC THẮNG

Số hiệu sinh viên: 20163830

Lớp: Toán Tin K61

HÀ NỘI - 2020

# Mục lục

Danh mục từ viết tắt	3
Danh sách hình vẽ	3
Lời cảm ơn	6
Lời mở đầu	7
<b>1 Lý thuyết học sâu và bài toán nhận diện vật thể</b>	<b>8</b>
1.1 Neural Network - Mạng Neural . . . . .	10
1.2 Mạng Neural tích chập (Convolutional Neural Network - CNN) . . . . .	12
1.3 Bài toán nhận diện vật thể (Object Detection) . . . . .	16
1.3.1 Tổng quan bài toán . . . . .	16
1.3.2 Sơ lược về Transfer Learning . . . . .	17
1.3.3 Một số phương pháp đánh giá mô hình nhận diện vật thể . . . . .	18
1.3.3.1 Intersection over Union (IoU) . . . . .	18
1.3.3.2 Average Precision . . . . .	20
1.4 Mô hình RetinaNet . . . . .	22
1.4.1 Mạng kim tự tháp đặc trưng Feature Pyramid Network (FPN) .	23
1.4.2 Mạng con phân loại . . . . .	26
1.4.3 Mạng con hồi quy . . . . .	26
1.4.4 Hàm mất mát . . . . .	27
<b>2 Ứng dụng học sâu vào bài toán đếm cây trên ảnh viễn thám</b>	<b>32</b>
2.1 Giới thiệu bài toán . . . . .	32

2.2	Mô hình hoá bài toán và thiết kế dữ liệu luyện . . . . .	33
2.3	Huấn luyện mạng neural . . . . .	38
2.4	Xây dựng chương trình . . . . .	39
2.5	Kết quả huấn luyện . . . . .	40
<b>3</b>	<b>Cài đặt chương trình và đánh giá kết quả</b>	<b>41</b>
3.1	Môi trường cài đặt chương trình và các yêu cầu liên quan . . . . .	41
3.1.1	Môi trường cài đặt chương trình . . . . .	41
3.1.2	Các yêu cầu liên quan . . . . .	41
3.2	Dữ liệu đầu vào . . . . .	42
3.3	Kết quả mô hình . . . . .	42
3.4	Đánh giá kết quả . . . . .	43
3.5	Định hướng phát triển trong tương lai . . . . .	45
	Tài liệu tham khảo . . . . .	46

# **Danh mục từ viết tắt**

# Danh sách hình vẽ

1.1	Mối quan hệ giữa AI, Machine Learning và Deep Learning. (Nguồn: <i>What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?</i> ) . . . . .	9
1.2	Ví dụ một mạng Neural có $k$ tầng ẩn (Nguồn: <i>CS229</i> ) . . . . .	10
1.3	Một số hàm kích hoạt thường dùng (Nguồn: <i>CS229</i> ) . . . . .	11
1.4	Mối quan hệ giữa network, layers, loss function và optimizer (Nguồn: Deep Learing with Python - Francois Chollet) . . . . .	12
1.5	Ví dụ về một CNN (Nguồn: <i>CS230</i> ) . . . . .	13
1.6	Mô tả hoạt động của CONV (Nguồn: <i>CS230</i> ) . . . . .	13
1.7	Hai kiểu pooling phổ biến (Nguồn: <i>CS230</i> ) . . . . .	14
1.8	Fully Connected layer (Nguồn: <i>CS230</i> ) . . . . .	15
1.9	ReLU và biến thể (Nguồn: <i>CS230</i> ) . . . . .	15
1.10	Ví dụ về đầu ra của bài toán nhận diện vật thể . . . . .	16
1.11	Một ví dụ về phát hiện biển báo Stop từ hình ảnh. (Nguồn: <i>pyimagesearch.com</i> )	19
1.12	Tính toán Intersection over Union. (Nguồn: <i>pyimagesearch.com</i> ) . . . .	19
1.13	Một ví dụ về tính toán IoU cho những bounding box khác nhau. (Nguồn: <i>pyimagesearch.com</i> ) . . . . .	20
1.14	Cách tính Precision và Recall. (Nguồn: <i>Machine Learning cơ bản</i> ) . . .	20
1.15	Precision-recall curve (Nguồn: <i>Medium</i> ) . . . . .	21
1.16	Một số metric được sử dụng để đánh giá kết quả trên bộ dữ liệu COCO (Nguồn: <i>cocodataset.org</i> ) . . . . .	22
1.17	Kiến trúc mô hình RetinaNet. . . . .	23
1.18	Khối thường (trái) và khối residual (phải) . . . . .	24

1.19	Feature Pyramid Network (FPN) . . . . .	24
1.20	Trích xuất đặc trưng trong mô hình FPN . . . . .	25
1.21	Top-down . . . . .	25
1.22	Bottom-up và top-down. . . . .	26
1.23	Đồ thị hàm focal loss với các giá trị $\gamma$ khác nhau . . . . .	29
1.24	Minh họa các lớp positive và negative trong object detection . . . . .	31
2.2	Dữ liệu sử dụng trong bài toán . . . . .	34
2.3	Minh họa ảnh viễn thám với phần mềm QGIS . . . . .	35
2.4	Toạ độ các bounding box được gán nhãn trên ảnh, biểu thị bởi các hình tròn màu vàng . . . . .	35
2.5	Đọc file shapefile với thư viện geopandas . . . . .	36
2.6	Toạ độ 4 đỉnh của một polygon . . . . .	37
2.7	Optional caption for list of figures . . . . .	38
2.8	Cấu trúc chương trình . . . . .	39
3.1	Cấu hình máy tính sử dụng cho bài toán . . . . .	42
3.2	Dữ liệu đầu vào để phát hiện vật thể . . . . .	43
3.3	Kết quả mô hình: thư mục chứa các shapfile . . . . .	43
3.4	Kết quả mô hình: các ô vuông màu xanh là kết quả phát hiện của mô hình; chấm màu cam là các cây được gán nhãn sẵn. . . . .	44
3.5	Dánh giá kết quả . . . . .	45

# Lời cảm ơn

# **Lời mở đầu**

Việc sản xuất và phân phối loại cây trồng có giá trị kinh tế cao đòi hỏi phải có sự giám sát thường xuyên đối với theo chu kỳ và theo mùa vụ. Đây là một nhiệm vụ quan trọng đối với việc quản lý tài nguyên, rừng. Để đáp ứng nhu cầu ngày càng tăng của thế giới và sản lượng nông nghiệp tiêu dùng, năng suất cây trồng phải được ước tính. Việc phát hiện, thu thập thủ công dữ liệu cây trồng để lưu trữ hồ sơ trên một diện tích đất lớn là không khả thi đối với con người, gây tốn kém tiền bạc và dễ bị ảnh hưởng bởi lỗi của con người. Việc phát hiện cây trồng trên một diện tích lớn sẽ giúp doanh nghiệp giảm bớt sự phụ thuộc vào con người và chi phí, dễ dàng quản lý và lên kế hoạch kịp thời cho sản xuất. Trong vài năm trở lại đây, lĩnh vực học sâu và xử lý ảnh đang nhận được nhiều sự quan tâm, áp dụng trong nhiều lĩnh vực khoa học khác nhau. Hơn nữa, cùng với sự phát triển của công nghệ, các bức ảnh viễn thám chụp bề mặt trái đất có kích thước ngày càng lớn, có độ nét cao, mang lại cơ hội cho việc viễn thám các cảnh quan sinh học có quy mô từ các sinh vật riêng lẻ đến các hệ thống toàn cầu. Trong báo cáo đồ án này, em xin giới thiệu một giải pháp phát hiện cây trong các bức ảnh viễn thám dựa trên mô hình học sâu. Nội dung của đồ án gồm có phần mở đầu, 4 chương, phần kết luận, tài liệu tham khảo:

• **Chương 1:**

• **Chương 2:**

• **Chương 3:**

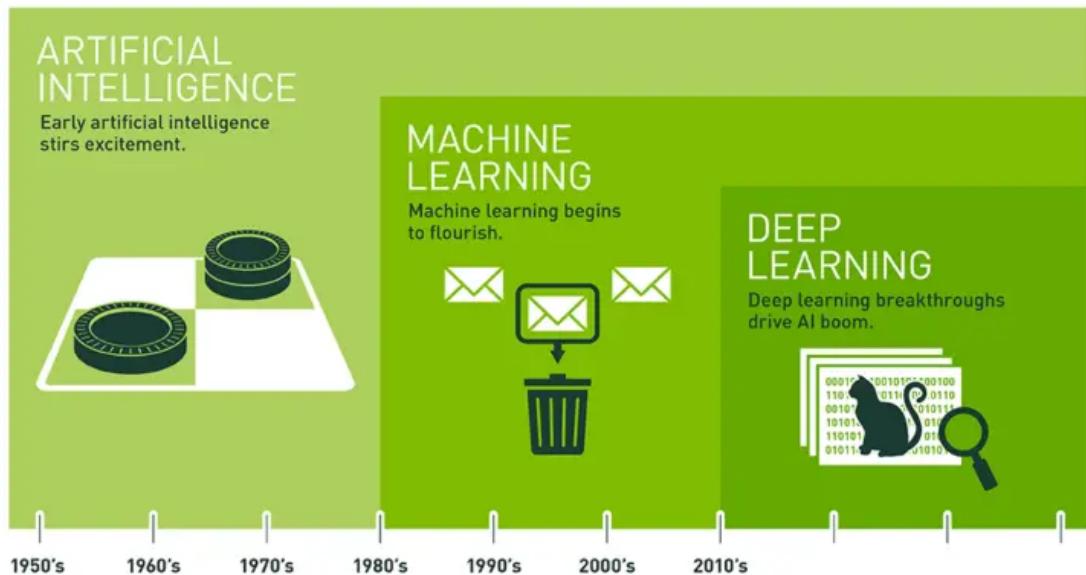
# Chương 1

## Lý thuyết học sâu và bài toán nhận diện vật thể

Deep Learning là một kỹ thuật Machine Learning mà ở đó huấn luyện máy tính giống như cách thức tự nhiên của con người: Học qua các ví dụ. Những năm gần đây, Deep learning đã mang đến nhiều bất ngờ trên quy mô toàn cầu và dẫn đường cho những tiến triển nhanh chóng trong nhiều lĩnh vực khác nhau như thị giác máy tính, xử lý ngôn ngữ tự nhiên (natural language processing), nhận dạng giọng nói tự động (automatic speech recognition), học tăng cường (reinforcement learning), và mô hình hóa thống kê (statistical modeling). Với những tiến bộ này, chúng ta bây giờ có thể xây dựng xe tự lái với mức độ tự động ngày càng cao (nhưng chưa nhiều tới mức như vài công ty đang tuyên bố), xây dựng các hệ thống giúp trả lời thư tự động khi con người ngập trong núi email, hay lập trình phần mềm chơi cờ vây có thể thắng cả nhà vô địch thế giới, một kỷ tích từng được xem là không thể đạt được trong nhiều thập kỷ tới. Những công cụ này đã và đang gây ảnh hưởng rộng rãi tới các ngành công nghiệp và đời sống xã hội, thay đổi cách tạo ra các bộ phim, cách chẩn đoán bệnh và đóng một vai trò ngày càng tăng trong các ngành khoa học cơ bản – từ vật lý thiên văn tới sinh học.

Với Deep Learning, một mô hình máy tính học cách thực hiện một công việc phân loại (classification) trực tiếp từ các hình ảnh, chữ viết (text) hoặc âm thanh. Các mô hình (models) Deep Learning có thể đạt được độ chính xác cao, đôi khi còn hơn cả con người. Các mô hình được huấn luyện bởi việc sử dụng một tập bao gồm bộ dữ liệu

được gán nhãn và các kiến trúc mạng neural gồm nhiều lớp (layer).



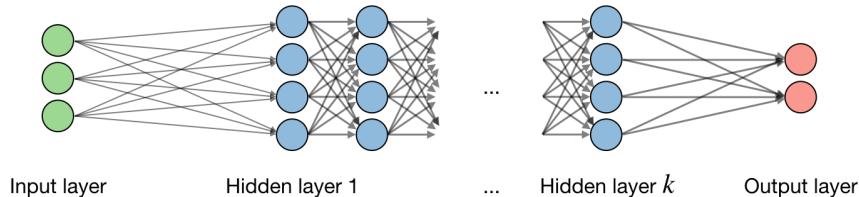
Hình 1.1: Mối quan hệ giữa AI, Machine Learning và Deep Learning. (Nguồn: *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?*)

**Vậy điều gì mang đến sự thành công của deep learning?** Rất nhiều những ý tưởng cơ bản của deep learning được đặt nền móng từ những năm 80-90 của thế kỷ trước, tuy nhiên deep learning chỉ đột phá trong khoảng từ năm 2012. Vì sao? Có thể kể đến một vài nhân tố dẫn đến sự bùng nổ này:

- Sự ra đời của các bộ dữ liệu lớn được gán nhãn.
- Khả năng tính toán song song tốc độ cao của GPU.
- Sự cải tiến của các kiến trúc: GoogLeNet, VGG, ResNet, ... và các kỹ thuật transfer learning, fine tuning.
- Nhiều thư viện mới hỗ trợ việc huấn luyện deep network với GPU: Theano, Caffe, TensorFlow, PyTorch, Keras,...

## 1.1 Neural Network - Mạng Neural

Tổng quan kiến trúc một mạng Neural như sau



Hình 1.2: Ví dụ một mạng Neural có  $k$  tầng ẩn (Nguồn: CS229)

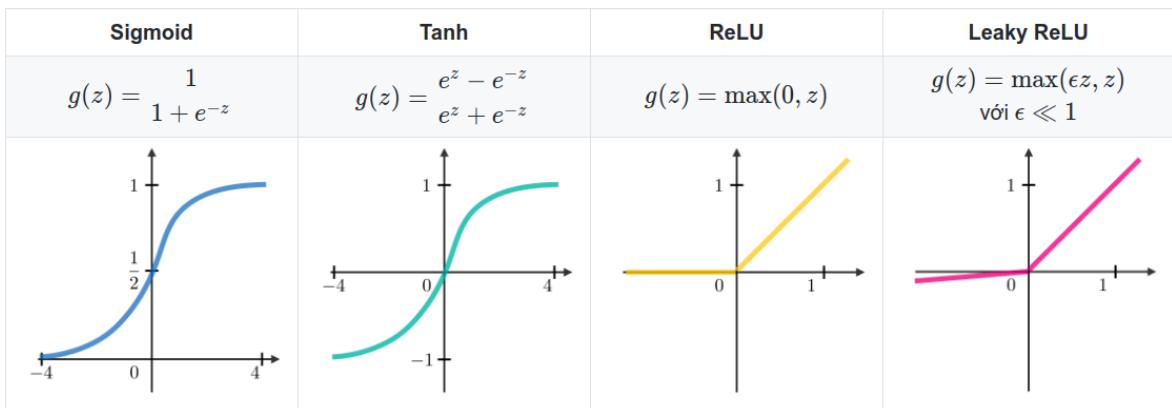
Với  $i$  là lớp thứ  $i$  của mạng,  $j$  là đơn vị ẩn thứ  $j$  của lớp, ta có:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

trong đó:  $w$  là weight,  $b$  là bias,  $z$  là đầu ra.

**Hàm kích hoạt (Activation function):** Bản chất của công thức trên là một tổ hợp tuyến tính giữa các giá trị input  $x$  và bộ trọng số  $w$ , do đó, khi áp dụng chúng với các dữ liệu mà có dạng tuyến tính, tức là dữ liệu mà ta có thể kẻ một đường thẳng để phân cách giữa chúng thì công thức tổ hợp trên đã đủ để giúp cho mô hình máy học có thể hoạt động tốt, chúng ta không cần tới hàm kích hoạt (activation function). Nhưng với các dữ liệu không có dạng tuyến tính, ta không thể kẻ một đường thẳng tuyến tính mà phân tách 2 dữ liệu ra được, và câu hỏi đặt ra là làm thế nào với một công thức tổ hợp tuyến tính như ban đầu mà dùng để phân lớp dữ liệu phi tuyến tính được. **Hàm kích hoạt** được tạo ra để làm điều này, hàm kích hoạt đóng vai trò như một người trung gian có nhiệm vụ chuyển đổi, nén hoặc chế biến output  $z$  từ tuyến tính trở thành phi tuyến tính.

**Hàm mất mát (Loss function):** Hàm mất mát trả về một số thực không âm thể hiện sự chênh lệch giữa hai đại lượng:  $y_{pred}$  là giá trị được dự đoán và  $y_{true}$  là giá trị thực. Trong trường hợp lý tưởng,  $y_{pred} = y_{true}$ , hàm mất mát sẽ có giá trị bằng 0. Hàm loss được sử dụng phổ biến trong các mô hình Deep learning hiện nay là Cross-entropy cùng các biến thể cải tiến của nó (Weighted cross entropy, Focal loss...). Cross-entropy



Hình 1.3: Một số hàm kích hoạt thường dùng (Nguồn: CS229)

loss  $L(z, y)$  được định nghĩa như sau:

$$L(z, y) = -[y \log z + (1 - y) \log(1 - z)]$$

**Optimizer và Learning rate:** Sau khi tính giá trị hàm loss, việc cần làm là tối ưu (cực tiểu hóa) hàm loss và update bộ trọng số  $\{w\}$  mới. Learning rate, thường được ký hiệu là  $\alpha$  hoặc  $\eta$ , thể hiện cho tốc độ học hay tốc độ update trọng số. Learning rate có thể là cố định hoặc được thay đổi tùy biến trong quá trình học.

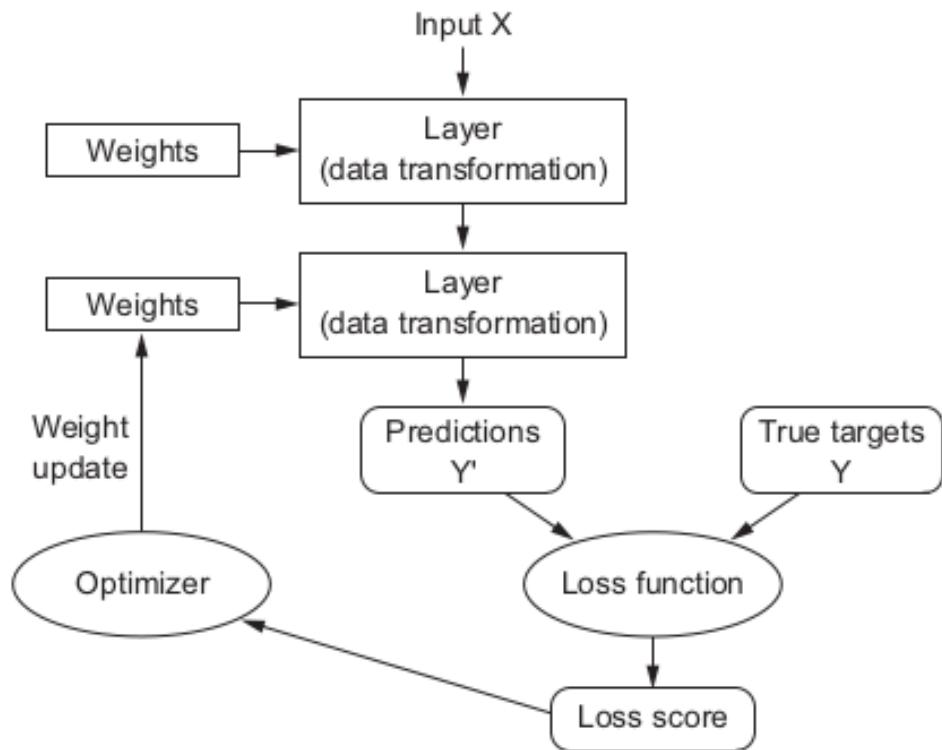
**Lan truyền ngược (Backpropagation):** Lan truyền ngược là phương thức dùng để cập nhật trọng số trong mạng neural bằng cách tính toán đầu ra thực sự và đầu ra mong muốn. Đạo hàm theo trọng số  $w$  được tính bằng cách sử dụng quy tắc chuỗi (chain rule) dưới đây:

$$\frac{\partial L(z, y)}{\partial w} = \frac{\partial L(z, y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

Như kết quả, trọng số được cập nhật như sau:

$$w := w - \eta \frac{\partial L(z, y)}{\partial w}$$

Tổng kết lại, ta có sơ đồ quá trình học của một mạng neural cơ bản như sau



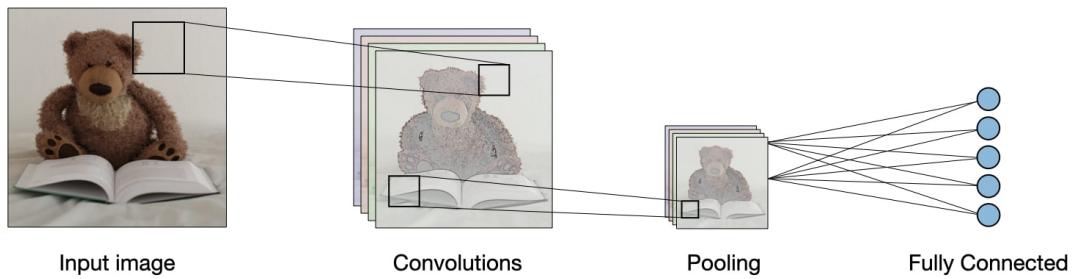
Hình 1.4: Mối quan hệ giữa network, layers, loss function và optimizer (Nguồn: Deep Learning with Python - Francois Chollet)

## 1.2 Mạng Neural tích chập (Convolutional Neural Network - CNN)

Mạng Neural tích chập là một trong những mô hình Deep learning phổ biến nhất và có ảnh hưởng nhiều nhất trong lĩnh vực Computer Vision. CNNs được dùng trong nhiều bài toán như nhận dạng ảnh, phân tích video, ảnh MRI, hoặc có thể cho cả các bài của lĩnh vực xử lý ngôn ngữ tự nhiên, và hầu hết đều giải quyết tốt các bài toán này.

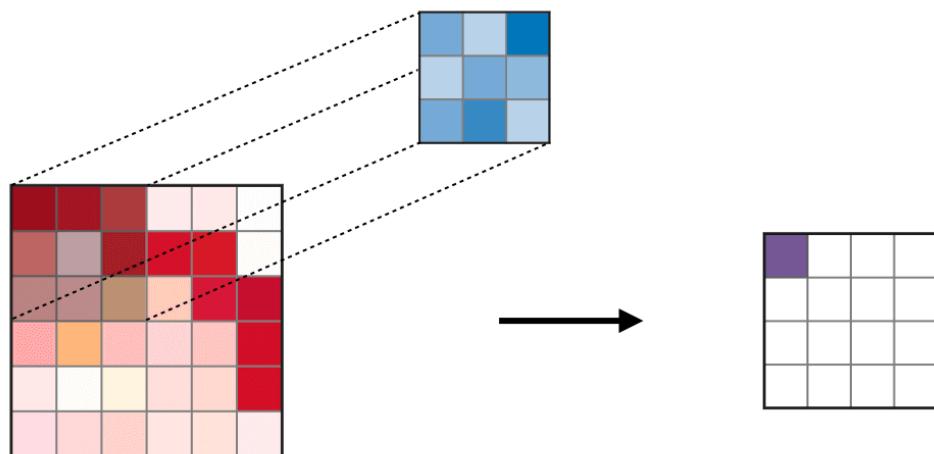
Mạng neural tích chập, còn được biết đến với tên CNNs, là một dạng mạng neural được cấu thành bởi các layer sau:

**Lớp tích chập (Convolution layer):** Tầng tích chập (CONV) sử dụng các bộ lọc (filters) để thực hiện phép tích chập khi đưa chúng đi qua input  $I$  theo các chiều của nó. Các *hyperparameters* của filter bao gồm kích thước  $F$ , độ trượt (stride)  $S$ . Kết quả



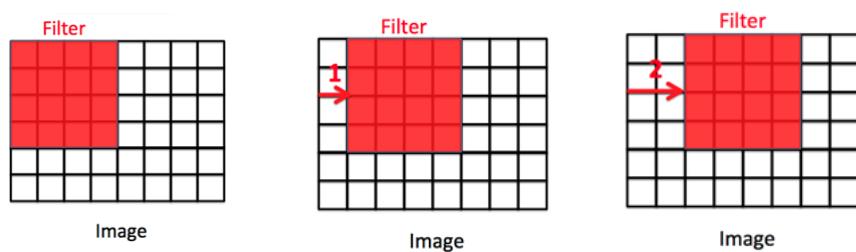
Hình 1.5: Ví dụ về một CNN (Nguồn: CS230)

đầu ra của lớp này được gọi là feature map.



Hình 1.6: Mô tả hoạt động của CONV (Nguồn: CS230)

**Stride** là số lượng pixel dịch chuyển trên ma trận đầu vào hay Stride dùng để dịch chuyển filter theo mỗi bước xác định.



Ví dụ về stride = 1 và stride bằng 2

**Padding:** Khi áp dụng phép CONV thì ma trận đầu vào sẽ có nhỏ dần đi, do đó số layer của mô hình CNN sẽ bị giới hạn, và không thể xây dựng mô hình mong muốn.

Để giải quyết tình trạng này, ta cần "bọc" bên ngoài ma trận đầu vào để đảm bảo kích thước đầu ra sau mỗi tầng convolution là không đổi. Do đó có thể xây dựng được mô hình với số tầng convolution lớn tùy ý. Một cách đơn giản và phổ biến nhất để padding là sử dụng hàng số 0, ngoài ra có một số phương pháp khác như reflection padding hay là symmetric padding.

**Lớp Pooling:** Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

**Spatial pooling** được gọi là lấy mẫu con làm giảm chiều của mỗi map nhưng vẫn giữ được thông tin quan trọng. Spatial pooling có thể có nhiều loại khác nhau như Max Pooling và Average Pooling.

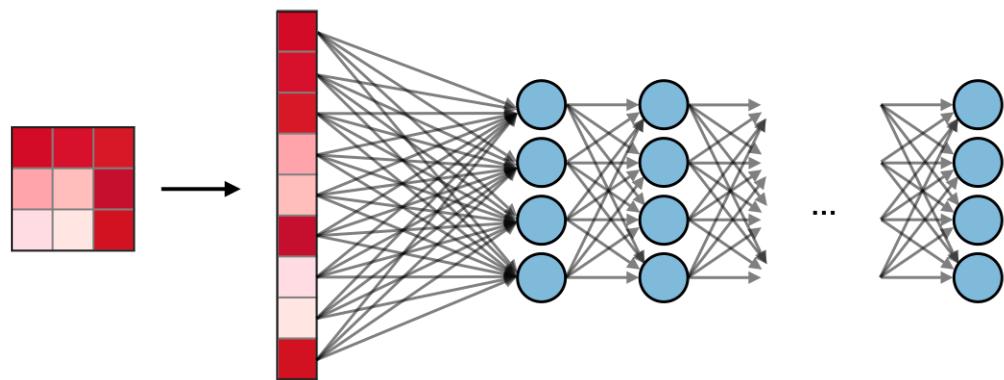
Kiểu	Max pooling	Average pooling
<b>Chức năng</b>	Từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng	Từng phép pooling tính trung bình các giá trị trong khu vực mà nó đang được áp dụng
<b>Mình họa</b>		
<b>Nhận xét</b>	<ul style="list-style-type: none"> <li>Bảo toàn các đặc trưng đã phát hiện</li> <li>Được sử dụng thường xuyên</li> </ul>	<ul style="list-style-type: none"> <li>Giảm kích thước feature map</li> <li>Được sử dụng trong mạng LeNet</li> </ul>

Hình 1.7: Hai kiểu pooling phổ biến (Nguồn: CS230)

**Fully Connected (FC):** Lớp Fully Connected nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các lớp FC thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.

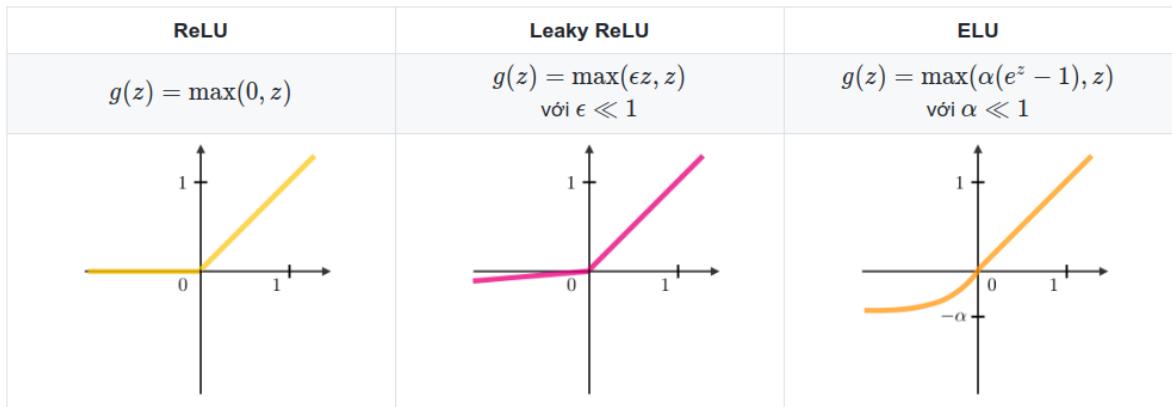
**Các hàm kích hoạt thường gặp:**

- **Rectified Linear Unit (ReLU):** ReLU là một hàm kích hoạt  $g$  được sử dụng trên tất cả các thành phần. Mục đích của nó là tăng tính phi tuyến tính cho



Hình 1.8: Fully Connected layer (Nguồn: CS230)

mạng. Những biến thể khác của ReLU được tổng hợp ở bảng dưới



Hình 1.9: ReLU và biến thể (Nguồn: CS230)

- **Softmax:** Bước softmax có thể được coi là một hàm logistic tổng quát lấy đầu vào là một vector chứa các giá trị  $x \in \mathbb{R}^n$  và cho ra là một vector gồm các xác suất  $p \in \mathbb{R}^n$  thông qua một hàm softmax ở layer cuối.

$$p = (p_1, \dots, p_n)^T \text{ trong đó } p = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

## 1.3 Bài toán nhận diện vật thể (Object Detection)

### 1.3.1 Tổng quan bài toán

Các hình ảnh trong cuộc sống bình thường thì không chỉ chứa 1 đối tượng mà thường bao gồm rất nhiều các đối tượng. Ta quan tâm đến vị trí của từng đối tượng trong ảnh. Bài toán như vậy được gọi là: object detection.



Hình 1.10: Ví dụ về đầu ra của bài toán nhận diện vật thể

Bài toán object detection có input là ảnh màu và output là vị trí của các đối tượng trong ảnh. Ta thấy nó bao gồm 2 bài toán nhỏ:

- Xác định các bounding box (hình chữ nhật) quanh đối tượng.
- Với mỗi bounding box thì cần phân loại xem đây là đối tượng gì (người, mèo, ô tô,...) với bao nhiêu phần trăm chắc chắn.

Object Detection là 1 bài toán đã đạt được rất nhiều các thành tựu trong những năm gần đây, cả phần ứng dụng và mô hình thuật toán. Diễn hình là các phương pháp Object Detection sử dụng Deep Learning đã đạt được các bước cải thiện vượt trội so với các phương pháp xử lý ảnh thông thường khác.

Có hai loại bài toán Object detection: two-stage object detection và one-stage object detection.

**Two-stage object detection:** Diễn hình họ các thuật toán R-CNN. Việc gọi là two-stage là do cách model xử lý để lấy ra được các vùng có khả năng chứa vật thể từ bức ảnh. Ví dụ, với Faster-RCNN thì trong stage-1, ảnh sẽ được đưa ra 1 sub-network gọi là RPN (Region Proposal Network) với nhiệm vụ extract các vùng trên ảnh có khả năng chứa đối tượng dựa vào các anchor. Sau khi đã thu được các vùng đặc trưng từ RPN, model Faster-RCNN sẽ thực hiện tiếp việc phân loại đối tượng và xác định vị trí nhờ vào việc chia làm 2 nhánh tại phần cuối của mô hình (Object classification & Bounding box regression).

**One-stage Object Detection:** Các thuật toán diễn hình như: SSD, YOLO, RetinaNet. Gọi là one-stage vì trong việc thiết kế model hoàn toàn không có phần trích chọn các vùng đặc trưng (các vùng có khả năng chứa đối tượng) như RPN của Faster-RCNN. Các mô hình one-stage object detection coi phần việc phát hiện đối tượng (object localization) như một bài toán regression (với 4 tọa độ offset, ví dụ x, y, w, h) và cũng dựa trên các box được định nghĩa sẵn gọi là anchor để làm việc đó. Các mô hình dạng này thường nhanh hơn tuy nhiên "độ chính xác" của model thường kém hơn so với two-stage object detection. Tuy nhiên, một số mô hình one-stage vẫn tỏ ra vượt trội hơn một chút so với two-stage như Retina-Net với việc việc thiết kế mạng theo FPN (Feature Pyramid Network) và Focal Loss.

### 1.3.2 Sơ lược về Transfer Learning

Những năm gần đây, Deep Learning phát triển cực nhanh dựa trên lượng dữ liệu training khổng lồ và khả năng tính toán ngày càng được cải tiến của các máy tính. Các kết quả cho bài toán phân loại ảnh ngày càng được nâng cao. Bộ cơ sở dữ liệu thường được dùng nhất là ImageNet với 1.2M ảnh cho 1000 classes khác nhau. Rất nhiều các mô hình Deep Learning đã giành chiến thắng trong các cuộc thi ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Có thể kể ra một vài: AlexNet, ZFNet, GoogLeNet, ResNet, VGG.

Nhìn chung, các mô hình này đều bao gồm rất nhiều layers. Các layers phía trước

thường là các Convolutional layers kết hợp với các nonlinear activation functions và pooling layers (và được gọi chung là ConvNet). Layer cuối cùng là một Fully Connected Layer và thường là một Softmax Regression (Xem Hình 1). Số lượng units ở layer cuối cùng bằng với số lượng classes (với ImageNet là 1000). Vì vậy output ở layer gần cuối cùng (second to last layer) có thể được coi là feature vectors và Softmax Regression chính là Classifier được sử dụng.

Chính nhờ việc features và classifier được trained cùng nhau qua deep networks khiến cho các mô hình này đạt kết quả tốt. Tuy nhiên, những mô hình này đều là các Deep Networks với rất nhiều layers. Việc training dựa trên 1.2M bức ảnh của ImageNet cũng tốn rất nhiều thời gian (2-3 tuần).

Với các bài toán dựa trên tập dữ liệu khác, rất ít khi người ta xây dựng và train lại toàn bộ Network từ đầu, bởi vì có rất ít các cơ sở dữ liệu có kích thước lớn. Thay vào đó, phương pháp thường được dùng là sử dụng các mô hình (nếu phía trên) đã được trained từ trước, và sử dụng một vài kỹ thuật khác để giải quyết bài toán. Phương pháp sử dụng các mô hình có sẵn như thế này được gọi là Transfer Learning.

Có 2 loại transfer learning:

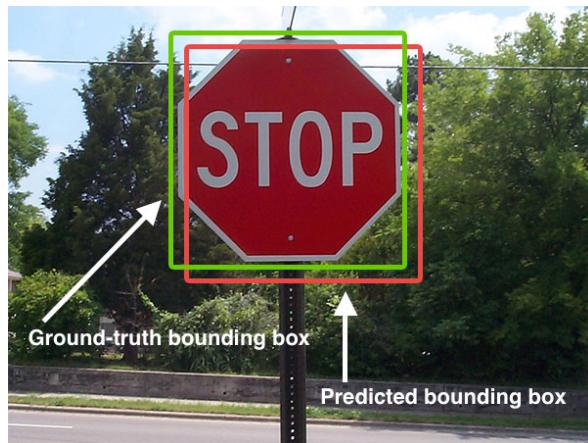
- **Feature extractor:** Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ dùng linear classifier (linear SVM, softmax classifier,...) để phân loại ảnh.
- **Fine tuning:** Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ coi đây là input của 1 CNN mới bằng cách thêm các ConvNet và Fully Connected layer.

### 1.3.3 Một số phương pháp đánh giá mô hình nhận diện vật thể

#### 1.3.3.1 Intersection over Union (IoU)

Intersection over Union là chỉ số đánh giá được sử dụng để đo độ chính xác của Object detector trên tập dữ liệu cụ thể. IoU đơn giản chỉ là một chỉ số đánh giá. Mọi thuật

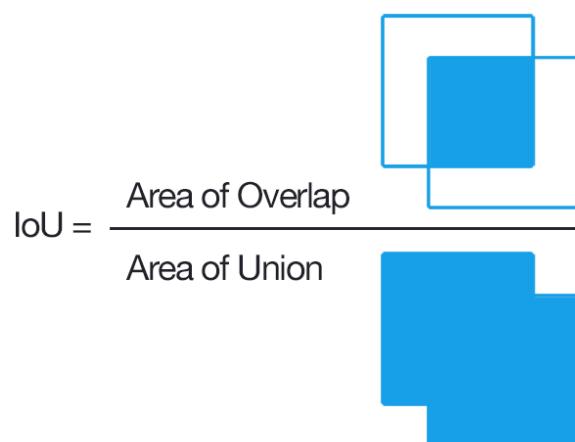
toán có khả năng predict ra các bounding box làm output đều có thể được đánh giá thông qua IoU.



Hình 1.11: Một ví dụ về phát hiện biển báo Stop từ hình ảnh. (Nguồn: [pyimagesearch.com](http://pyimagesearch.com))

Để áp dụng được IoU để đánh giá một mô hình nhận diện vật thể bất kì ta cần:

- Những ground-truth bounding box (bounding box đúng của đối tượng, ví dụ như bounding box của đối tượng được khoanh vùng và gán nhãn bằng tay sử dụng trong tập test.)
- Những bounding box dự đoán được model sinh ra.



Hình 1.12: Tính toán Intersection over Union. (Nguồn: [pyimagesearch.com](http://pyimagesearch.com))



Hình 1.13: Một ví dụ về tính toán IoU cho những bounding box khác nhau. (Nguồn: [pyimagesearch.com](http://pyimagesearch.com))

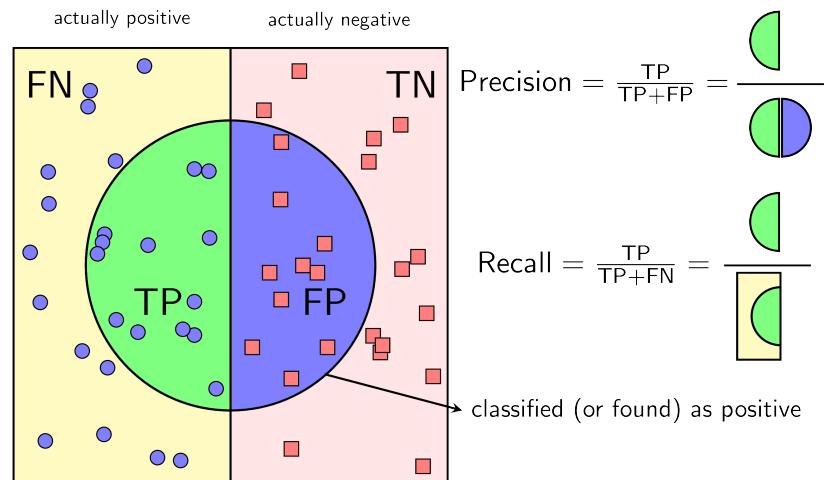
### 1.3.3.2 Average Precision

AP (Average Precision) là một metric phổ biến trong việc đánh giá độ chính xác của các mô hình nhận diện vật thể như Faster R-CNN, SSD,...

#### Precision – Recall

Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đo hiệu quả thường được sử dụng là Precision-Recall.

Trước hết xét bài toán phân loại nhị phân. Ta cũng coi một trong hai lớp là positive, lớp còn lại là negative.



Hình 1.14: Cách tính Precision và Recall. (Nguồn: *Machine Learning cơ bản*)

**Precision** cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. **Recall** cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự

positive là thấp.

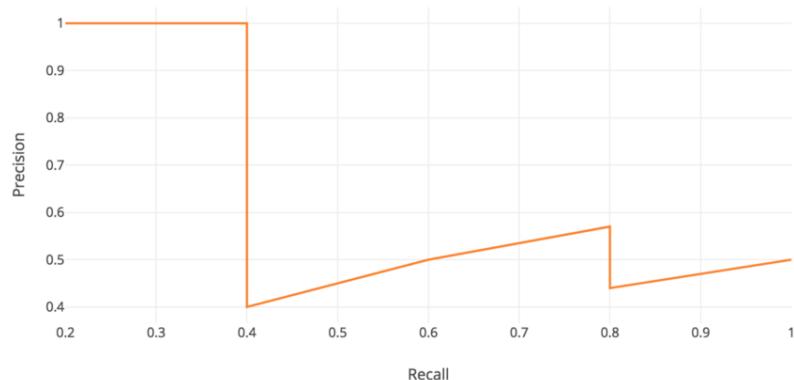
### Precision-Recall curve và Average precision

Ta có thể đánh giá mô hình dựa trên việc thay đổi một ngưỡng và quan sát giá trị của Precision và Recall.

**Average Precision:** Ta xét một ví dụ, với bộ dữ liệu có 5 quả táo, mô hình lần lượt đưa ra 10 dự đoán, ta chọn ngưỡng cho dự đoán đúng là  $\text{IoU} > 0.5$ , dự đoán được xếp giảm dần theo sự 'tự tin' của dự đoán. Số liệu được thể hiện trong bảng sau:

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Biểu diễn các điểm precision-recall ta được một đường zig-zag sau

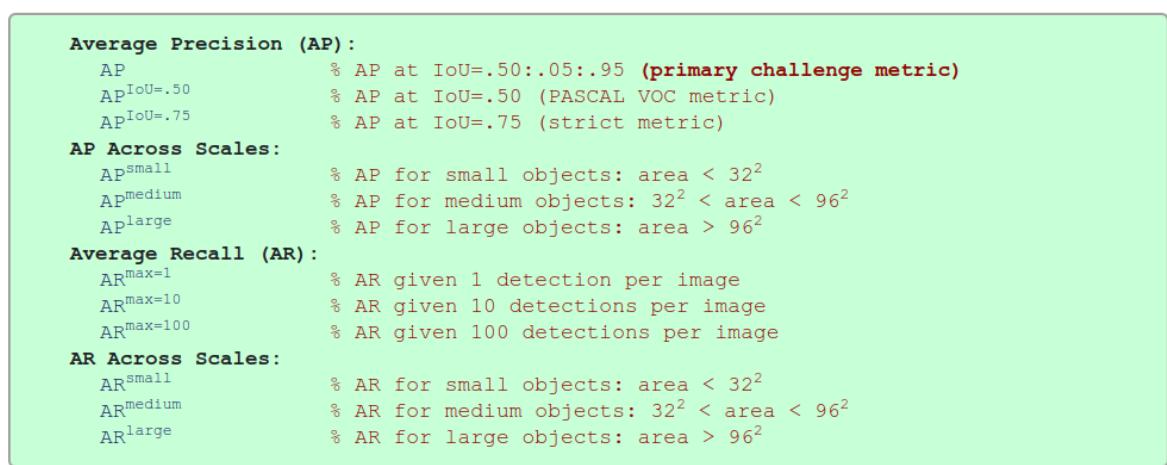


Hình 1.15: Precision-recall curve (Nguồn: Medium)

Dịnh nghĩa tổng quát cho Average Precision (AP) là diện tích phía dưới precision-recall curve

$$AP = \int_0^1 p(r)dr$$

Precision và recall luôn nằm trong đoạn [0; 1] do đó AP cũng nằm trong đoạn [0; 1]. Mean average precision (mAP) là trung bình của AP. Trong một số trường hợp, ta tính AP cho mỗi class và lấy trung bình của chúng, một số khác thì lại giống nhau. Ví dụ theo COCO, không có sự khác biệt giữa AP và mAP.

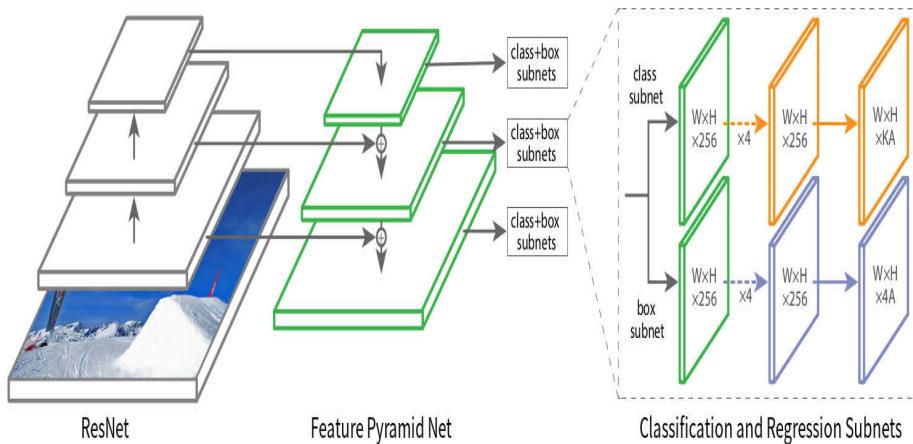


Hình 1.16: Một số metric được sử dụng để đánh giá kết quả trên bộ dữ liệu COCO (Nguồn: [cocodataset.org](http://cocodataset.org))

## 1.4 Mô hình RetinaNet

RetinaNet là một mạng tổng hợp bao gồm:

- Mạng kim tự tháp đặc trưng-Feature Pyramid Networks (FPN) được xây dựng dựa trên mạng ResNet; chịu trách nhiệm tính toán, trích xuất các đặc trưng trong bức ảnh
- Mạng con chịu trách nhiệm thực hiện phân loại đối tượng bằng cách sử dụng đầu ra của mạng kim tự tháp đặc trưng FPN
- Mạng con chịu trách nhiệm điều chỉnh các kích thước phát hiện vật thể từ đầu ra của mạng kim tự tháp đặc trưng FPN



Hình 1.17: Kiến trúc mô hình RetinaNet.

### 1.4.1 Mạng kim tự tháp đặc trưng Feature Pyramid Network (FPN)

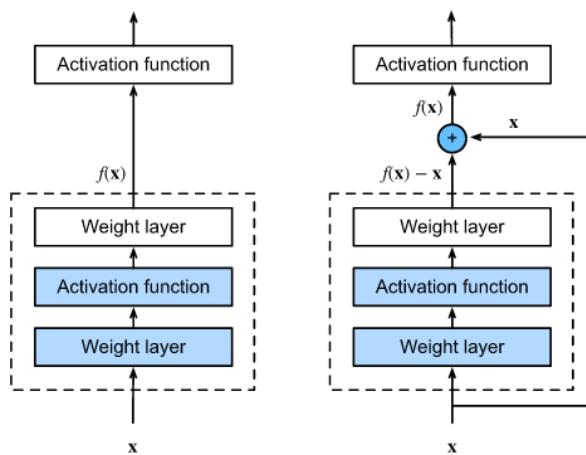
**Residual Network (ResNet):** Một vấn đề phổ biến của Deep learning là Vanishing Gradients, tức là theo công thức tính đạo hàm bằng chain rule (trong lan truyền ngược, back propagation), nếu gradient của các lớp sau nhỏ thì gradient ở các lớp đầu sẽ gần bằng 0. Vì thế mà parameter của các lớp trước sẽ không được cập nhật nên các parameter này không đóng góp được gì trong việc đưa ra output. Vậy nên dù ta dùng nhiều lớp, thực chất số lớp hữu ích chỉ là một vài lớp cuối, điều đó làm giảm sự hiệu quả của mạng neural. ResNet ra đời để giải quyết vấn đề đó, giải pháp mà ResNet đưa ra là sử dụng kết nối "tắt" đồng nhất để xuyên qua một hay nhiều lớp. Một khối như vậy được gọi là một Residual Block.

ResNet-50 là một mạng bao gồm 50 lớp residual.

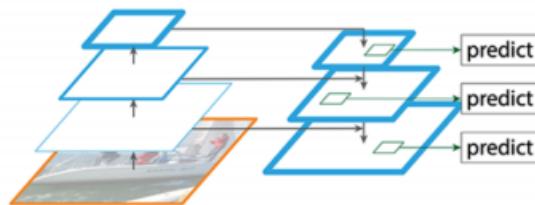
#### Mạng kim tự tháp đặc trưng-Feature Pyramid Networks (FPN)

Phát hiện các đối tượng có kích thước nhỏ là một vấn đề vô cùng quan trọng để nâng cao độ chính xác. Và FPN là mô hình mạng được thiết kế ra dựa trên khái niệm kim tự tháp (pyramid) để giải quyết vấn đề này.

Mô hình FPN giúp nhận diện được cả những đối tượng có kích thước to và nhỏ nhờ sự kết hợp của tất cả các lớp kim tự tháp. Mô hình được xây dựng bằng cách thực



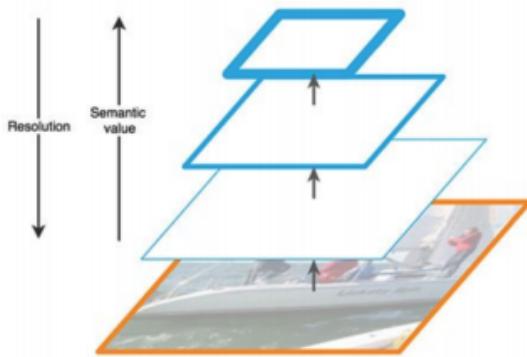
Hình 1.18: Khối thường (trái) và khối residual (phải)



Hình 1.19: Feature Pyramid Network (FPN)

hiện quá trình bottom-up kết hợp với top-down để dò tìm đối tượng (trong khi đó, các thuật toán khác chỉ thường sử dụng bottom-up). Trong đó bottom-up sử dụng mạng tích chập thông thường dùng để trích xuất các đặc trưng, khi chúng ta ở bottom và đi lên (up) độ phân giải sẽ giảm, nhưng giá trị cấu trúc cấp cao (high-level semantic) sẽ tăng lên.

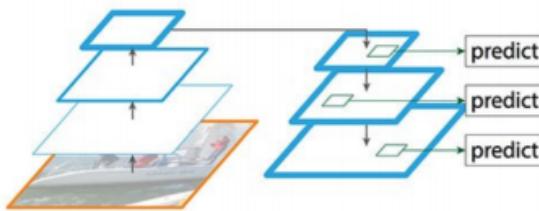
Một vài các mô hình khác trong phát hiện đối tượng (ví dụ: SSD,...) quyết định dựa vào nhiều feature map. Nhưng tầng ở dưới không được sử dụng để nhận dạng đối tượng. Vì những tầng này có độ phân giải cao nhưng giá trị cấu trúc cấp cao của chúng lại không đủ cao nên những nhà nghiên cứu bỏ chúng đi để tăng tốc độ xử lý. Các nhà nghiên cứu biện minh rằng các tầng ở dưới chưa đủ mức ý nghĩa cần thiết để nâng cao độ chính xác, thêm các tầng đó vào sẽ không nâng độ chính xác cao thêm bao nhiêu và họ bỏ chúng đi để có tốc độ tốt hơn. Cho nên, những mô hình phát hiện đối tượng này chỉ sử dụng các tầng ở lớp trên, và do đó sẽ không nhận dạng được các đối tượng có kích thước nhỏ.



Hình 1.20: Trích xuất đặc trưng trong mô hình FPN

Trong khi đó, FPN xây dựng thêm mô hình top-down, nhằm mục đích xây dựng các tầng có độ phân giải cao hơn từ các tầng high-level semantics

Trong quá trình xây dựng lại các tầng top-down, chúng ta sẽ gấp một ván để

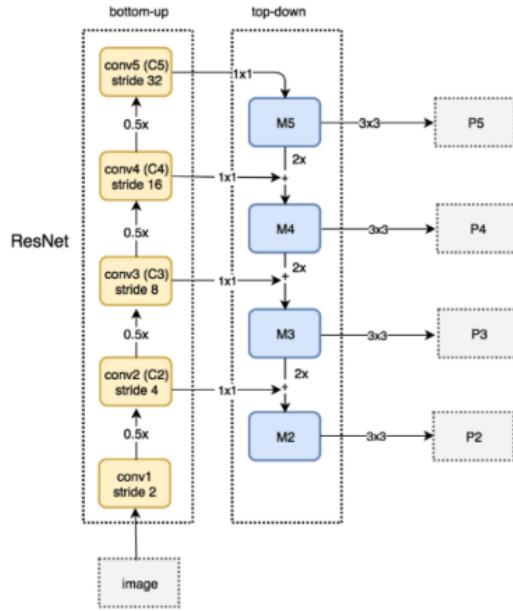


Hình 1.21: Top-down

khá nghiêm trọng là bị mất mát thông tin của các đối tượng. Ví dụ một đối tượng nhỏ khi đi lên (top) sẽ không thấy nó, và từ trên đi ngược lại sẽ không thể tái tạo lại đối tượng nhỏ đó. Để giải quyết vấn đề này, chúng ta sẽ tạo các kết nối (skip connection) giữa các reconstruction layer và các feature maps để giúp quá trình dự đoán các vị trí của đối tượng thực hiện tốt hơn (sao cho hàm mất mát đạt giá trị nhỏ nhất).

Đồ hình bên dưới biểu diễn chi tiết đường đi theo bottom-up và topdown. P2, P3, P4, P5 là các pyramid của các feature map.

FPN không phải là mô hình phát hiện đối tượng. Nó là mô hình phát hiện đặc trưng và được sử dụng trong phát hiện đối tượng. Các feature map từ P2 đến P5 trong Hình 1.22 độc lập với nhau và các đặc trưng được sử dụng để phát hiện đối tượng.



Hình 1.22: Bottom-up và top-down.

### 1.4.2 Mạng con phân loại

Mạng con phân loại là một mạng tích chập đầy đủ (FCN) gắn ở mỗi mức của FPN. Mạng con này chứa 4 lớp tích chập  $3 \times 3$  với 256 bộ lọc, kết hợp với hàm kích hoạt RELU; tiếp đến là lớp tích chập  $3 \times 3$  với số bộ lọc =  $KA$ . Do đó đầu ra của feature map có kích thước  $W * H * KA$ , trong đó  $W, H$  là các kích thước của feature map,  $K, A$  lần lượt là số lượng class và số lượng anchor box. Lý do cho việc sử dụng lớp tích chập cuối cùng của mạng con phân loại có  $K * A$  bộ lọc vì nếu có “A” anchor box được đề xuất cho mỗi vị trí trong feature map thu được từ lớp tích chập cuối thì mỗi anchor box có khả năng được phân loại được  $K$  lớp. Do đó, đầu ra của feature map sẽ có  $K * A$  bộ lọc.

### 1.4.3 Mạng con hồi quy

Song song với mạng con phân loại đối tượng, mô hình đi kèm một mạng tích chập đầy đủ khác vào mỗi cấp của mạng FPN nhằm mục đích điều chỉnh kích thước các anchor box trong việc được phát hiện các đối tượng (nếu có). Cấu trúc của mạng con này

giống với mạng con phân loại ở trên, ngoại trừ lớp tích chập cuối có kích thước  $3 \times 3$  với 4 bộ lọc cho kết quả đầu ra feature map của mạng con này có kích thước  $W * H * 4A$ . Lý do để lớp tích chập cuối có 4 bộ lọc là vì để xác định vị trí đối tượng trong ảnh, mạng con hồi quy tạo ra 4 giá trị cho mỗi anchor box dự đoán độ lệch tương đối (về tọa độ tâm, chiều rộng và chiều cao) giữa các anchor box với box thực tế chứa đối tượng. Do đó, đầu ra của feature map trong mạng con hồi quy sẽ có  $4 * A$  bộ lọc (kênh)

#### 1.4.4 Hàm mất mát

Trong bài toán nhận diện vật thể, nhãn dữ liệu mô tả chính xác vị trí đối tượng trong bức ảnh được gọi là các ground-truth boxes. Thông qua 2 mạng con ở trên, mỗi anchor box được dự đoán sẽ được phân vào một lớp nào đó và vị trí của anchor box đó trong ảnh. Để tính toán hàm mất mát trong training, ta cần phải so sánh các giá trị dự đoán này với nhãn của dữ liệu. Một anchor box được coi là dự đoán đúng với một ground truth box nào đó nếu giá trị IOU giữa 2 đối tượng này  $> 0.5$ ; một anchor box được coi là không khớp với bất kỳ ground truth box nào trong ảnh nếu giá trị IOU giữa chúng  $< 0.4$ . Cuối cùng, nếu giá trị IOU giữa một anchor box với bất kỳ ground truth box nào nằm giữa 0.4 và 0.5 sẽ không đóng góp vào hàm mất mát.

Hàm mất mát của mô hình RetinaNet chứa 2 thành phần: Hàm mất mát hồi quy và hàm mất mát phân loại

#### Hàm mất mát hồi quy

Giả sử các cặp dữ liệu trùng khớp nhau kí hiệu  $(A^i, G^i)_{i=1, \dots, N}$ , trong đó  $A$  là tập các anchor box,  $G$  là tập các ground truth box,  $N$  là số lượng các ground truth box.

Với mỗi anchor, mô hình dự đoán 4 giá trị, kí hiệu lần lượt  $P^i = (P_x^i, P_y^i, P_w^i, P_h^i)$ . Hai giá trị đầu là độ lệch giữa tâm của anchor  $A_i$  và ground truth  $G_i$ , trong đó 2 giá trị cuối là độ lệch giữa chiều cao và chiều rộng của 2 box đó. Tương ứng với mỗi dự đoán,

giá trị mục tiêu  $T_i$  được tính như là độ lệch giữa anchor box và gt box theo công thức:

$$T_x^i = (G_x^i - A_x^i) / A_w^i \quad (1.1)$$

$$T_y^i = (G_y^i - A_y^i) / A_h^i \quad (1.2)$$

$$T_w^i = \log(G_w^i / A_w^i) \quad (1.3)$$

$$T_h^i = \log(G_h^i / A_h^i) \quad (1.4)$$

Với các giá trị trên, hàm mất mát hồi quy được định nghĩa là:

$$L_{loc} = \sum_{j \in \{x,y,w,h\}} smooth_{L1}(P_j^i - T_j^i) \quad (1.5)$$

trong đó hàm  $smooth_{L1}(x)$  có công thức:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & |x| \geq 1 \end{cases} \quad (1.6)$$

## Hàm mất mát phân loại

Hàm mất mát phân loại mà mô hình retinanet sử dụng là Focal Loss. Nói một cách đơn giản, Focal loss (FL) là một phiên bản cải tiến của hàm Cross-Entropy Loss (CE) cố gắng xử lý vấn đề mất cân bằng giữa các lớp foreground và background; một điều thường xảy ra trong các bài toán object detection. Hàm được xây dựng bằng cách gán nhiều trọng số hơn cho các mẫu khó hoặc dễ bị phân loại sai và giảm trọng số cho các mẫu dễ phân loại. Vì vậy, Focal loss làm giảm sự đóng góp tổn thất từ các mẫu dễ phân loại và tăng tầm quan trọng của việc sửa lỗi các mẫu được phân loại sai. Đầu tiên ta sẽ tìm hiểu về hàm Cross-Entropy (CE) loss trong bài toán phân loại nhị phân:

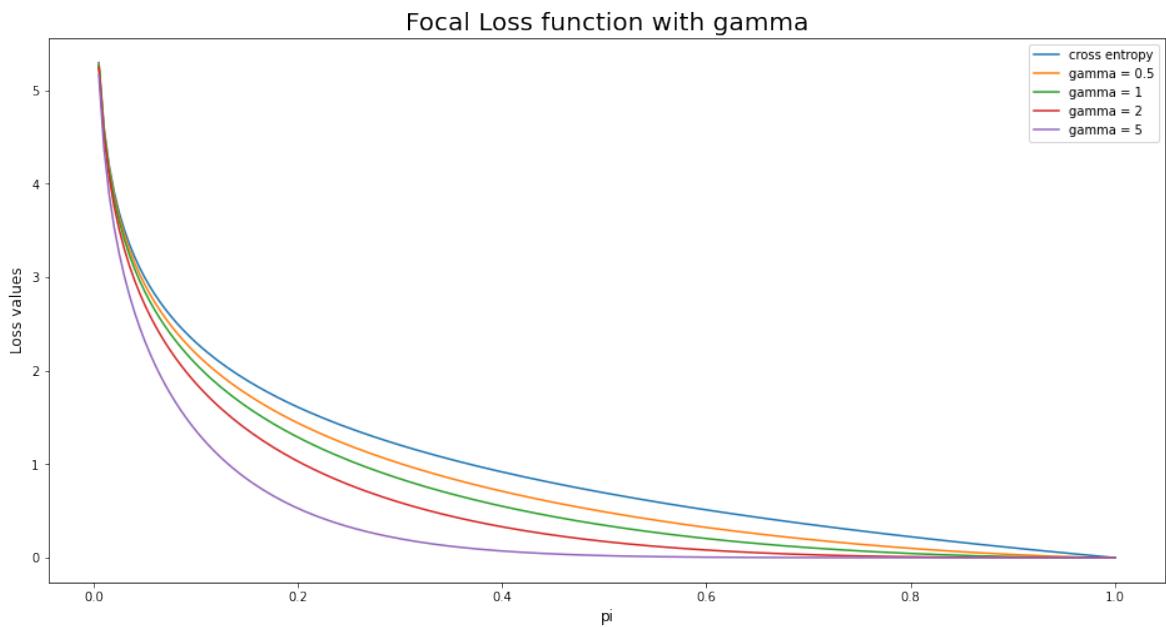
$$CE(p, y) = \begin{cases} -\log(p) \text{ if } y = 1 \\ -\log(1 - p) \text{ otherwise,} \end{cases} \quad (1.7)$$

với  $y$  là nhãn phân loại của dữ liệu,  $p \in [0, 1]$  là xác suất dự đoán cho lớp có nhãn  $y$

của mô hình. Để thuận tiện, ta định nghĩa  $p_t$ :

$$p_t = \begin{cases} p & \text{nếu } y = 1 \\ 1 - p & \text{otherwise,} \end{cases} \quad (1.8)$$

Khi đó  $CE(p, y) = CE(p_t) = -\log(p_t)$



Hình 1.23: Đồ thị hàm focal loss với các giá trị  $\gamma$  khác nhau

Đồ thị của CE loss có thể được xem là đường cong màu xanh trong hình 1.23. Một điểm đáng chú ý có thể dễ dàng nhận ra từ đồ thị trên, đó là ngay cả những mẫu dễ phân loại (có  $p_t \gg 0.5$ ) cũng có giá trị loss khá lớn.

Trong cross entropy ta thấy rằng vai trò đóng góp vào loss function của các class cùng bằng  $-\log(p_i)$ . Khi xảy ra hiện tượng mất cân bằng, chúng ta muốn rằng mô hình sẽ dự báo chuẩn hơn đối với những class thiểu số. Do đó cần một hàm loss function hiệu quả hơn, có thể điều chỉnh được giá trị phạt lớn hơn đối với nhóm thiểu số.

Một kỹ thuật thường dùng để giải quyết việc mất cân bằng giữa các lớp là áp dụng trọng số  $\alpha$  bằng nghịch đảo tần suất nhãn vào hàm cross entropy. Hàm loss function mới được gọi là balanced cross entropy:

$$BCE(\mathbf{p}, \mathbf{q}) = -\alpha_i \log(q_i), \quad \text{với } p_i = 1 \quad (1.9)$$

hàm mất mát này là một mở rộng nhỏ của CE loss mà tác giả của bài báo coi là cơ sở cho đề xuất hàm mất mát mới: Focal loss. Focal loss là hàm loss function lần đầu được giới thiệu trong RetinaNet. Hàm loss function này đã chứng minh được tính hiệu quả trong các bài toán object detection. Đây là lớp bài toán có sự mất cân bằng nghiêm trọng giữa hai class positive (các bounding box có chứa object) và negative (các bounding box không chứa object). Thường thì negative có số lượng lớn hơn positive rất nhiều. Lấy ví dụ như hình 1.24: Chỉ có 4 bounding box thuộc positive (đường viền in đậm), các trường hợp còn lại thuộc nhóm negative.

Do đó nếu áp dụng loss function là hàm cross entropy sẽ giảm độ chính xác khi dự báo các bounding box có chứa object. Trong bài báo Focal Loss for Dense Object Detection tác giả đã giới thiệu hàm Focal Loss có công thức như sau:

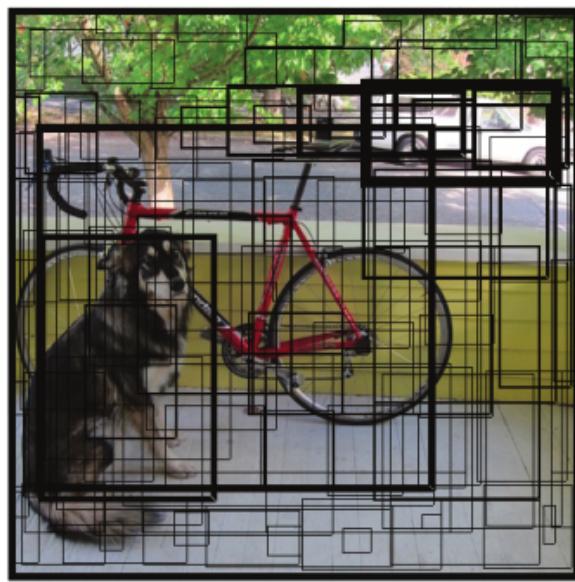
$$FP(\mathbf{p}, \mathbf{q}) = -(1 - q_i)^\gamma \log(q_i), \quad \text{với } p_i = 1 \quad (1.10)$$

Ta thấy hàm focal loss chỉ thêm nhân tử  $(1 - q_i)^\gamma$  so với công thức của balanced cross entropy. Tuy nhiên nhân tử này lại có tác dụng rất lớn trong việc điều chỉnh ảnh hưởng của nhãn lên đồng thời loss function và gradient descent. Thật vậy, xét hai trường hợp dễ dự báo (easy examples) và khó dự báo (hard examples):

- Dễ dự báo: Chúng ta thấy rằng mô hình huấn luyện trên mẫu mất cân bằng thường dự báo chính xác các mẫu đa số. Những trường hợp này được gọi là dễ dự báo. Xác suất  $q_i$  của các trường hợp dễ dự báo có xu hướng cao hơn. Do đó  $(1 - p_t)^\gamma$  có xu hướng rất nhỏ và dường như không tác động lên hàm mất mát đáng kể.
- Khó dự báo: Trường hợp khó dự báo thì  $q_i$  là một giá trị nhỏ hơn. Do đó độ lớn tác động của nó lên loss function là  $(1 - p_t)^\gamma$  sẽ gần bằng 1. Mức độ tác động này lớn hơn rất nhiều lần so với trường hợp dễ dự báo. Cụ thể hơn, nếu trường hợp dễ dự báo có  $p_i = 0.9$  và khó dự báo có  $p_i = 0.1$  thì tỷ lệ chênh lệch của đóng góp vào loss function (giả sử khi  $\gamma = 2$ ) sẽ là:

$$\frac{(1 - 0.1)^2}{(1 - 0.9)^2} = \frac{0.9^2}{0.1^2} = 81$$

Tỷ lệ này sẽ còn lớn hơn nữa nếu tăng  $\gamma$  hoặc giá trị của  $p_i$  đổi với trường hợp dễ dự báo càng gần 1 và khó dự báo càng gần 0.



Hình 1.24: Minh họa các lớp positive và negative trong object detection

## Chương 2

# Ứng dụng học sâu vào bài toán đếm cây trên ảnh viễn thám

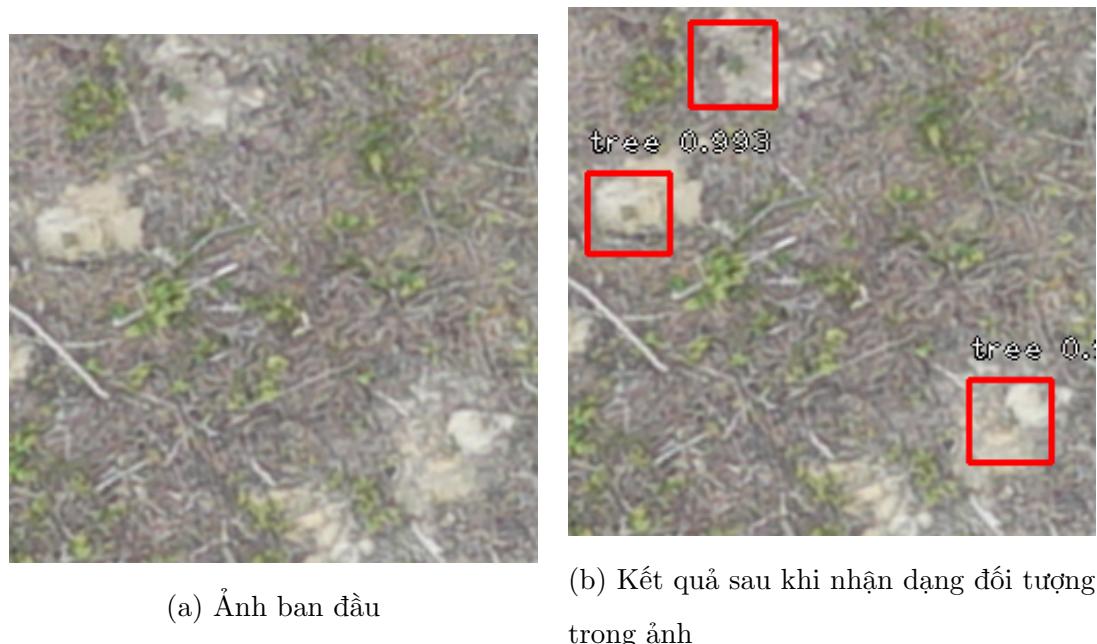
### 2.1 Giới thiệu bài toán

Cây keo là một loại cây dễ trồng, có khả năng thích nghi với các loại đất nghèo chất dinh dưỡng, ở những nơi thời tiết khắc nghiệt. Gỗ keo có rất nhiều tác dụng nên keo rất phù hợp với các dự án lâm nghiệp thương mại: là nguyên liệu sử dụng để sản xuất đồ nội thất xuất khẩu và sản xuất giấy. Cây keo ở Việt Nam ngày càng được trồng với quy mô lớn, nhằm nhanh chóng phủ xanh đồi núi trọc keo được trồng thành rừng, đem lại nguồn thu nhập lớn, tạo công ăn việc làm cho nhiều người dân. Để đáp ứng nhu cầu tiêu thụ gỗ ngày càng tăng ở trong nước và thế giới, sản lượng nông nghiệp tiêu dùng, năng suất cây trồng phải được ước tính. Việc phát hiện, kiểm soát các cây keo trên một diện tích lớn trong khu rừng giúp cho doanh nghiệp và các nhà quản lý biết được tình trạng hiện tại của các cây keo, có thể trồng thêm được cây nào vào những khu đất còn trống. Tuy nhiên việc thu thập thủ công dữ liệu cây trồng để lưu trữ trong hồ sơ trên một diện tích đất lớn là không khả thi đối với con người, gây tốn kém tiền bạc và dễ bị ảnh hưởng bởi lỗi của con người.

Một tiềm năng lớn để giải quyết vấn đề này là sử dụng các hình ảnh vệ tinh kết hợp với các thuật toán thị giác máy tính để phát hiện quản lý những cây trồng này. Sử dụng hình ảnh vệ tinh để khảo sát sẽ giảm bớt một phần công việc của con người

khi phải trực tiếp đến khu vực đó tiến hành theo dõi, phân tích. Các bức ảnh viễn thám được chụp lại với kích thước lớn, có thể bao quát một khu diện tích đất trồng rộng giúp tiết kiệm thời gian điều tra khảo sát, từ đó giúp các nhà quản lý của doanh nghiệp có thể nắm bắt tình hình kịp thời, lập kế hoạch sản xuất phù hợp.

Để việc giải quyết bài toán này trở nên đơn giản và tự động hơn, trong đồ án này, em trình bày về việc sử dụng các phương pháp Deep Learning để tiến hành phân tích và phát hiện các cây trong bức ảnh viễn thám.



## 2.2 Mô hình hóa bài toán và thiết kế dữ liệu luyện

a) Mô hình hóa bài toán

**Input:**

Ảnh viễn thám được lưu ở định dạng .tif, có kích thước lớn (từ vài chục MB đến GB)

**Tiền xử lý:**

Ảnh đầu vào sẽ được chia nhỏ thành các ảnh có kích thước  $256 \times 256$  (bằng với

kích thước trong tập ảnh training), với tỉ lệ *overlap* giữa các tấm ảnh là 0.5

### Output:

Các file định dạng .txt chứa các bounding box mà mô hình đoán đó là vật thể trong bức ảnh đó. Vì mục tiêu của bài toán là nhận dạng và đếm số cây trên một bức ảnh lớn, nên ta sẽ ghép các tấm ảnh nhỏ ở trên lại để đưa về hình ảnh gốc của chúng. Vì trong quá trình tiền xử lý, ta thiết lập overlap = 0.5 nên việc các bounding box của các bức ảnh gần nhau sẽ đè lên nhau, gây ra hiện tượng trùng lặp. Để giải quyết vấn đề này, các bounding box thu được trên toàn bộ các ảnh đầu tiên sẽ được đưa về tọa độ địa lý, sau đó sử dụng thuật toán NMS để loại bỏ các bounding box trùng nhau (nếu như giá trị IOU của hai bounding box này vượt quá một ngưỡng (trong mô hình này em lấy ngưỡng = 0.3)). Kết quả sau khi thực hiện thuật toán NMS sẽ

#### b) Thiết kế dữ liệu luyện mạng

Dữ liệu sử dụng trong bài toán này được cung cấp bởi thầy giáo hướng dẫn TS. Lê Hải Hà. Dữ liệu bao gồm 2 thư mục, mỗi thư mục gồm một vài ảnh ở định dạng GeoTif và các **shapefile** chứa tọa độ các bounding box trong các bức ảnh đó

W03_202003311249_RI_RSK_RSKA014702_RGB	21:13, 12-11-2 Reiwa	95,7 MB	Thư mục
W03_202003311249_RI....RSKA014702_RGB_0.tif	15:39, 04-11-2 Reiwa	23,7 MB	Ảnh TIFF
W03_202003311249_RI....K_RSKA014702_RGB_1.tif	15:39, 04-11-2 Reiwa	23,7 MB	Ảnh TIFF
W03_202003311249_RI....RSKA014702_RGB_2.tif	15:39, 04-11-2 Reiwa	23,7 MB	Ảnh TIFF
W03_202003311249_RI....RSKA014702_RGB_3.tif	15:39, 04-11-2 Reiwa	23,7 MB	Ảnh TIFF
W03_202003311249_RI_RSK_RSKA014702_RGB.dbf	09:24, 11-08-2 Reiwa	9 KB	Tài liệu
W03_202003311249_RI_RSK_RSKA014702_RGB.prj	09:24, 11-08-2 Reiwa	389 byte	Tài liệu
W03_202003311249_RI_RSK_RSKA014702_RGB.shp	09:24, 11-08-2 Reiwa	857 KB	ESRI Sh...cument
W03_202003311249_RI_RSK_RSKA014702_RGB.shx	09:24, 11-08-2 Reiwa	6 KB	Tài liệu
W05_202003281250_RI_RSK_RSKA003603_RGB	00:32, Hôm nay	292,2 MB	Thư mục
W05_202003281250_RI....RSKA003603_RGB_0.tif	15:39, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI....RSKA003603_RGB_1.tif	15:38, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI....RSKA003603_RGB_2.tif	15:39, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI....RSKA003603_RGB_3.tif	15:39, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI....RSKA003603_RGB_4.tif	15:39, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI....RSKA003603_RGB_5.tif	15:39, 04-11-2 Reiwa	48,1 MB	Ảnh TIFF
W05_202003281250_RI_RSK_RSKA003603_RGB.dbf	09:25, 11-08-2 Reiwa	42 KB	Tài liệu
W05_202003281250_RI_RSK_RSKA003603_RGB.prj	09:25, 11-08-2 Reiwa	389 byte	Tài liệu
W05_202003281250_RI....RSKA003603_RGB.shp	09:25, 11-08-2 Reiwa	3,8 MB	ESRI Sh...cument
W05_202003281250_RI_RSK_RSKA003603_RGB.shx	09:25, 11-08-2 Reiwa	28 KB	Tài liệu

Hình 2.2: Dữ liệu sử dụng trong bài toán



Hình 2.3: Minh họa ảnh viễn thám với phần mềm QGIS



Hình 2.4: Toạ độ các bounding box được gán nhãn trên ảnh, biểu thị bởi các hình tròn màu vàng

Các ảnh trong bộ dữ liệu ở trên có 2 loại kích thước :  $3141 \times 1885$  và  $3874 \times 3100$ .

Để thuận tiện cho việc training mô hình, ta sẽ chia nhỏ mỗi ảnh thành nhiều ảnh con có kích thước  $256 \times 256$  với giá trị  $overlap = 0.5$ . Để lấy bounding box trong từng bức ảnh được cắt ra, đầu tiên ta tiến hành đọc file *shapefile* của ảnh đó bằng thư viện *geopandas* (hình 2.5):

	<b>FID</b>	<b>geometry</b>
0	0	POLYGON ((782749.090 87317.692, 782749.088 873...
1	1	POLYGON ((782748.127 87316.369, 782748.126 873...
2	2	POLYGON ((782736.356 87317.697, 782736.355 873...
3	3	POLYGON ((782741.586 87317.399, 782741.584 873...
4	4	POLYGON ((782747.997 87314.795, 782747.995 873...
...	...	...
3462	3462	POLYGON ((782858.704 87068.587, 782858.702 870...
3463	3463	POLYGON ((782858.220 87067.431, 782858.218 870...
3464	3464	POLYGON ((782909.154 87222.547, 782909.152 872...
3465	3465	POLYGON ((782913.085 87225.018, 782913.083 872...
3466	3466	POLYGON ((782958.935 87267.731, 782958.933 872...
3467	rows × 2 columns	

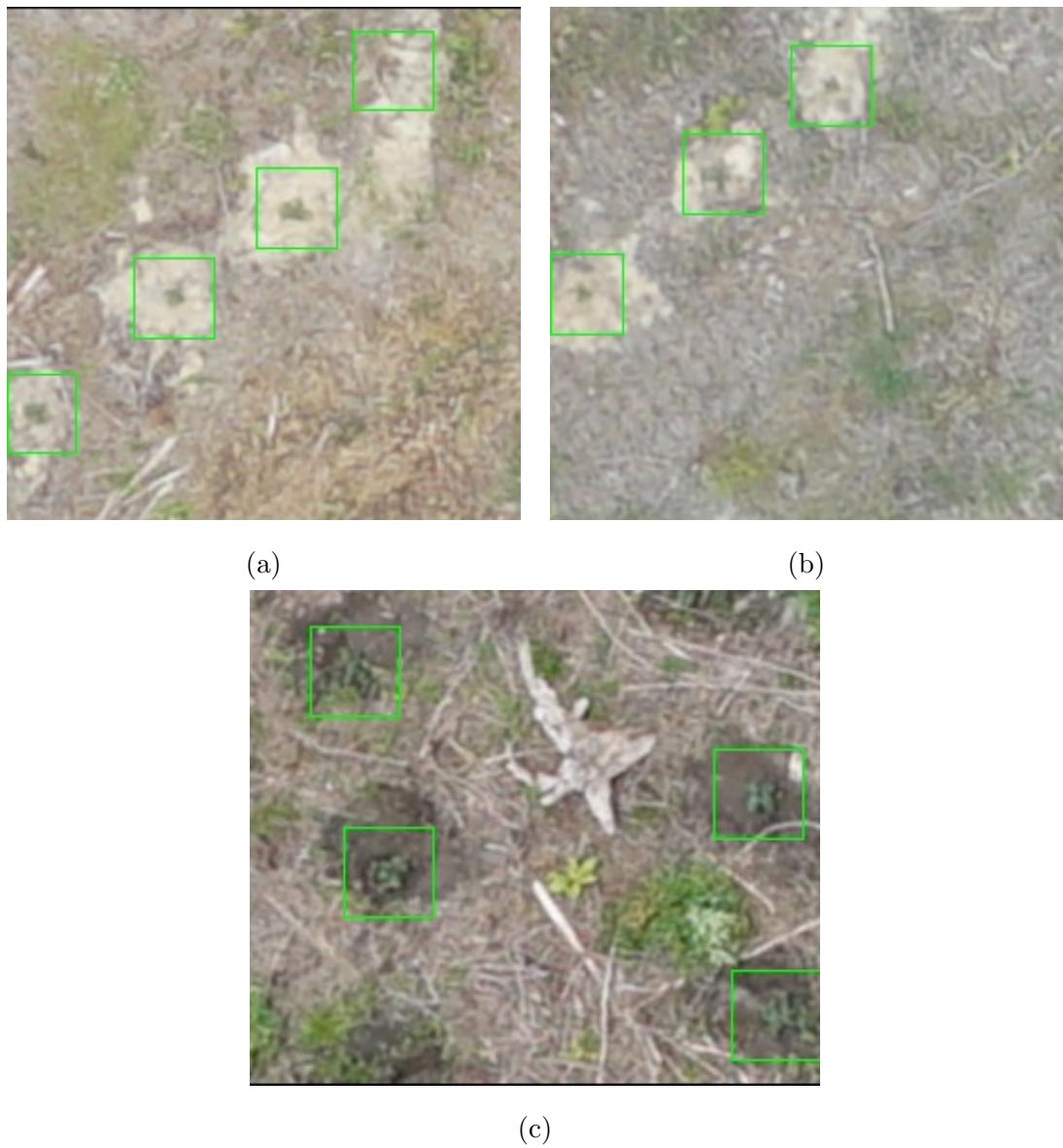
Hình 2.5: Đọc file shapefile với thư viện geopandas

Vì tọa độ các câu được đánh dấu theo hình tròn nên ta cần chuyển định dạng hình tròn về bounding box (hình 2.6)

polygons.bounds				
	minx	miny	maxx	maxy
0	782748.289193	87317.294211	782749.089590	87318.089250
1	782747.327019	87315.971222	782748.127415	87316.766261
2	782735.556070	87317.299441	782736.356466	87318.094480
3	782740.785279	87317.001376	782741.585675	87317.796415
4	782747.196289	87314.397230	782747.996685	87315.192269
...	...	...	...	...
3462	782857.903169	87068.189629	782858.703566	87068.984669
3463	782857.419266	87067.032845	782858.219664	87067.827885
3464	782908.353211	87222.149652	782909.153608	87222.944692
3465	782912.284263	87224.619968	782913.084660	87225.415008
3466	782958.134333	87267.332957	782958.934730	87268.127997
3467 rows × 4 columns				

Hình 2.6: Toạ độ 4 đỉnh của một polygon

Tuy nhiên trong hình trên, mỗi điểm là một toạ độ trên một mặt phẳng địa lý, ta phải chuyển các toạ độ  $x, y$  về toạ độ cột, hàng tương ứng. Một vài kết quả thu được sau khi tiến hành cắt ảnh và lấy bounding box tương ứng cho mỗi bức ảnh đó:



Hình 2.7: Một vài ảnh trong bộ dữ liệu được tạo ra

Kết quả sau khi cắt ảnh ra ta thu được bộ dữ liệu hơn 5000 ảnh kích thước  $256 \times 256$  cùng với tọa độ các bounding box chứa đối tượng trong từng ảnh đó

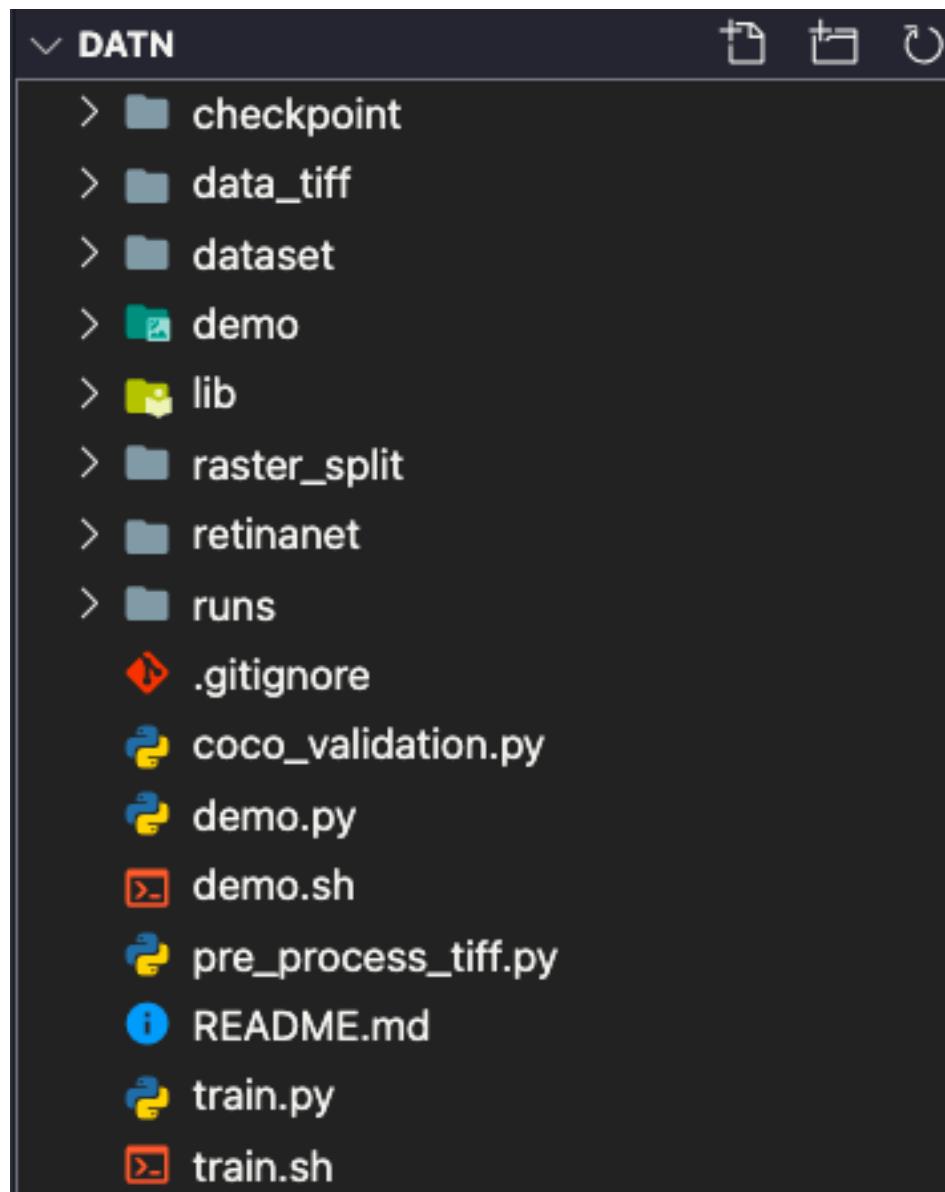
## 2.3 Huấn luyện mạng neural

Chúng ta sẽ tiến hành huấn luyện mô hình dựa trên bộ dữ liệu được tạo ra ở trên. Mô hình mà em sử dụng trong bài toán này là RetinaNet; trong đó backbone sử dụng ở

đây là ResNet50 được pretrained trên bộ dữ liệu nổi tiếng ImageNet phục vụ cho việc trích xuất các đặc trưng trong ảnh

## 2.4 Xây dựng chương trình

Chương trình được xây dựng trên nền tảng ngôn ngữ Python (phiên bản 3.6.9).. Cấu trúc thư mục của chương trình như hình 4.2 dưới đây: Dưới đây ta sẽ tìm hiểu kỹ chức



Hình 2.8: Cấu trúc chương trình

năng các thư mục và file trên:

- /data\_tiff: Thư mục này chứa các ảnh viễn thám định dạng GeoTiff cùng với các file ".shp" chứa các thông tin về đặc dạng hình học của đối tượng có trong ảnh
- /retinanet: Thư mục xây dựng mô hình RetinaNet: khởi tạo model, hàm loss, DataLoader để đưa dữ liệu vào mô hình
- /checkpoint: Thư mục này lưu lại các trọng số trong mạng sau mỗi lần duyệt qua toàn bộ dataset, phục vụ cho việc đánh giá và dự đoán model. Các file trong thư mục này đều có định dạng: \*.pt
- /raster\_split: Thư mục này dùng để chuyển các ảnh vệ tinh định dạng GeoTiff về định dạng .png đồng thời tạo một bộ dataset gồm các ảnh png và file .txt chứa vị trí các đối tượng trong ảnh.
- /lib: Thư mục này xây dựng các metrics trong việc đánh giá một mô hình object detection: Precision, Recall, Average Precision, F1-score.
- demo.py: Kết hợp với các file trong thư mục /demo thực hiện nhiệm vụ testing: Đầu tiên sẽ load file weight trong thư mục checkpoint, đọc dữ liệu ảnh GeoTiff đầu vào và đưa vào trong model. Kết quả sau đó được lưu lại và xử lý để xuất ra shapefile
- train.py: File này có nhiệm vụ chạy mô hình RetinaNet và lưu kết quả vào thư mục /checkpoint sau mỗi lần lặp trên bộ dataset. Thao tác này chỉ cần thực hiện một lần duy nhất.

## 2.5 Kết quả huấn luyện

# **Chương 3**

## **Cài đặt chương trình và đánh giá kết quả**

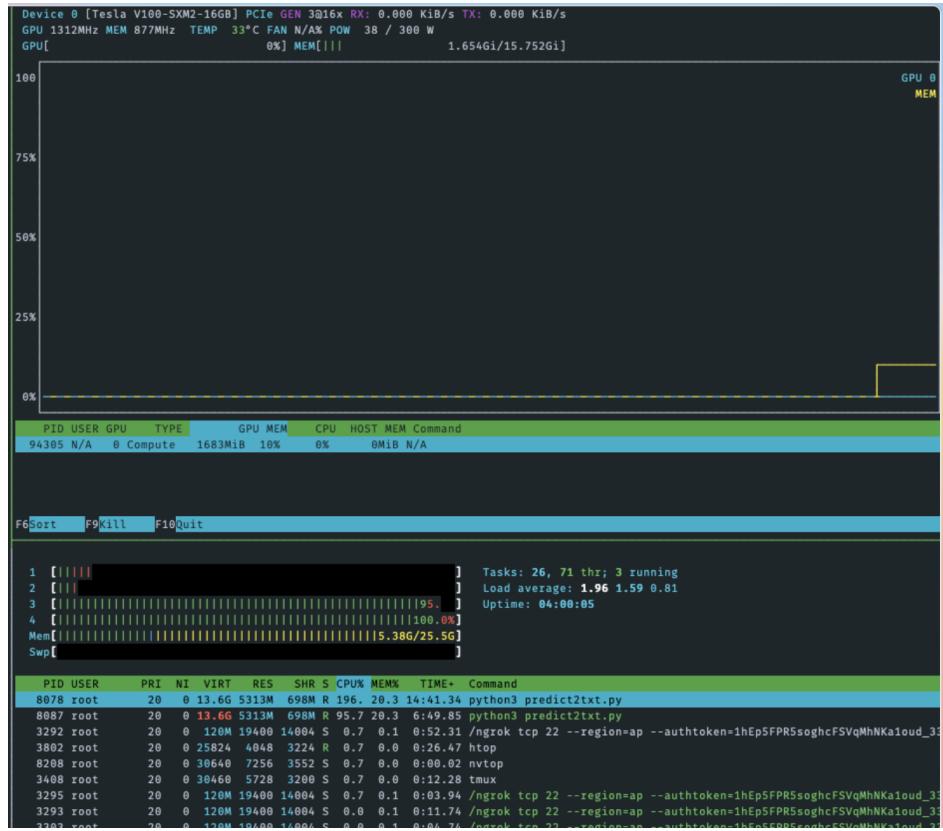
### **3.1 Môi trường cài đặt chương trình và các yêu cầu liên quan**

#### **3.1.1 Môi trường cài đặt chương trình**

Chương trình được cài đặt trên ngôn ngữ Python (phiên bản 3.6.9) và được thử nghiệm trên hệ điều hành máy ảo của google colab sử dụng chip 4 cores , bộ nhớ RAM 24GB, card đồ họa NVIDIA Tesla V100 16GB (hình 3.1)

#### **3.1.2 Các yêu cầu liên quan**

- Ngôn ngữ sử dụng: Python phiên bản 3.6.9
- IDE sử dụng: Visual Studio Code
- Các thư viện Python sử dụng trong chương trình được viết trong file "requirements.txt"



Hình 3.1: Cấu hình máy tính sử dụng cho bài toán

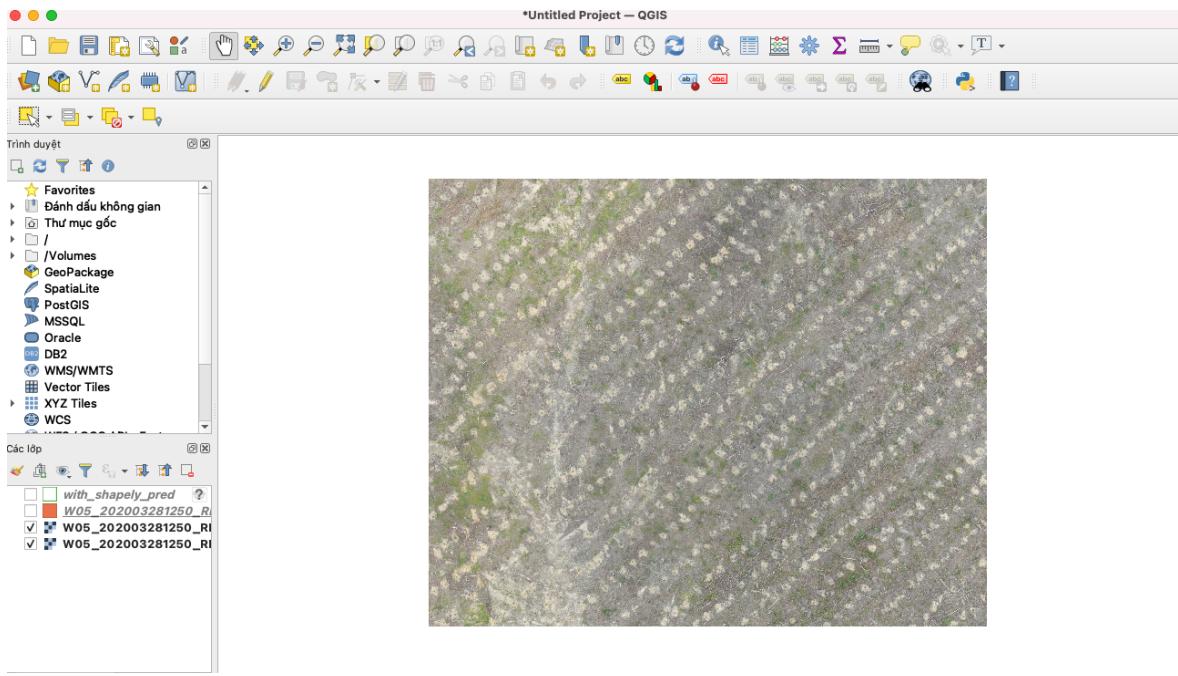
## 3.2 Dữ liệu đầu vào

Vì dữ liệu em được cung cấp bị hạn chế, nên trong bộ dữ liệu ban đầu em lấy ra 1 ảnh chưa được xử lý trong tập train và validation (hình 3.2) để làm dữ liệu đầu vào cho mô hình nhận diện vật thể đã xây dựng ở chương trước. Lý do của việc lấy một ảnh kích thước lớn để dự đoán là vì theo yêu cầu của khách hàng, họ cần đánh giá kết quả mô hình trên toàn bộ diện tích đất của họ chứ không phải các tấm ảnh kích thước  $256 \times 256$  được cắt nhỏ từ tấm ảnh ban đầu.

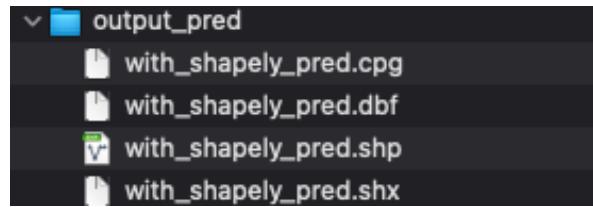
## 3.3 Kết quả mô hình

Thời gian chạy chương trình cho ảnh trên: 43 giây

Hình ảnh kết quả được lưu lại ở định dạng shapefile: (hình 3.3)



Hình 3.2: Dữ liệu đầu vào để phát hiện vật thể

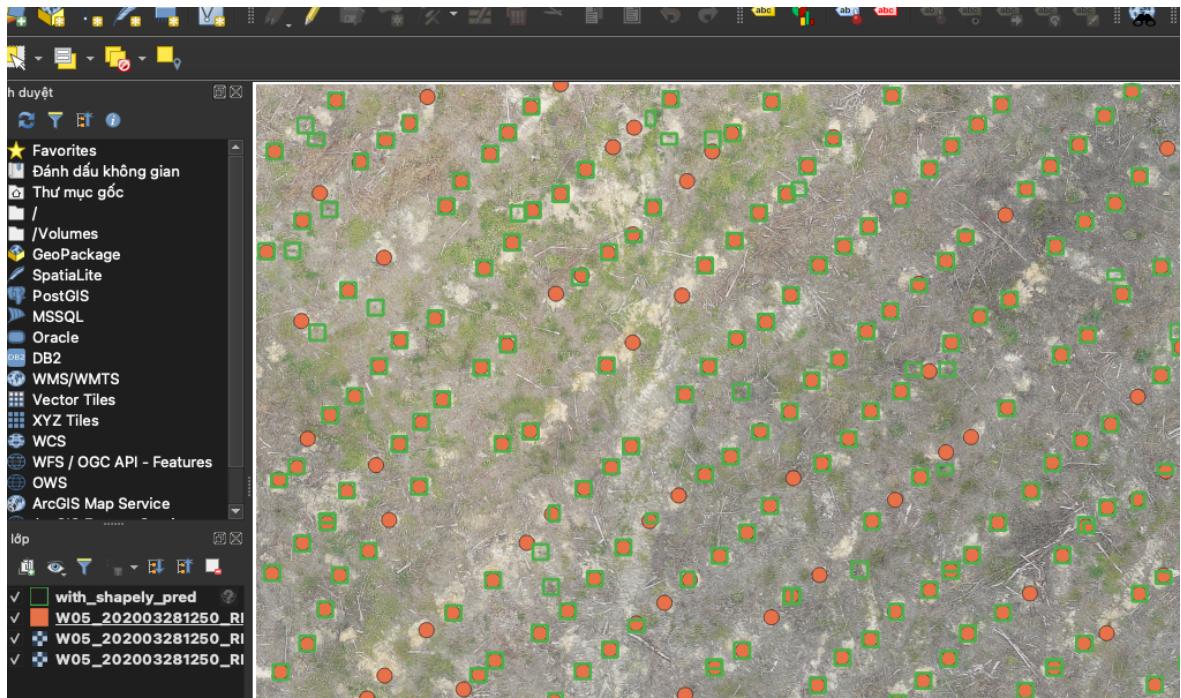


Hình 3.3: Kết quả mô hình: thư mục chứa các shapfile

Ta sử dụng file shapefile ở trên để mô phỏng kết quả trên phần mềm QGIS (hình 3.4):

### 3.4 Đánh giá kết quả

Hình 3.4 chỉ cho ta một cách khái quát về kết quả mô hình khi so sánh với các bounding box ban đầu. Để đánh giá kết quả của mô hình, trước tiên ta định nghĩa: Hai bounding box cùng phát hiện một đối tượng nếu giá trị IOU của chúng  $\geq 0.3$ . Khi đó, bounding box được dự đoán từ mô hình sẽ được coi là một *True positive*, ngược lại sẽ là *False positive*. Trong trường hợp có nhiều hơn 1 bounding box cùng dự đoán một đối tượng, ta sẽ lấy bounding box có confidence (độ tin cậy) cao nhất làm TP, các bounding box



Hình 3.4: Kết quả mô hình: các ô vuông màu xanh là kết quả phát hiện của mô hình; chấm màu cam là các cây được gán nhãn sẵn.

còn lại sẽ là FP. Để so sánh một cách khách quan, em sử dụng cùng một tập các thang đo phổ biến là Precision, Recall và F1-score. Trong đó, Precision là tỉ lệ giữa số cây được phát hiện chính xác trên tổng số cây được phát hiện. Recall là tỉ lệ giữa số cây được phát hiện đúng trên tổng số cây thực sự trong ảnh. F1-score là trung bình điều hòa giữa hai giá trị Precision và Recall. Các tham số trên được biểu diễn trong các biểu thức:

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

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

$$F1 - score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

```
Precision: tree: 0.8658536585365854
Recall: tree: 0.8214876033057851
F1-score: tree: 0.8430873621713316
-----Done-----
```

Hình 3.5: Đánh giá kết quả

### 3.5 Định hướng phát triển trong tương lai

Từ kết quả thực nghiệm cho thấy, chương trình đã có những thành công nhất định. Song bên cạnh đó cũng còn khá nhiều nhược điểm cần cải tiến. Trong quá trình hoàn thành đồ án em đã đầu tư nhiều thời gian và công sức với bài toán này, và nhận thấy đây là bài toán có khả năng phát triển cao hơn nữa . Em xin đưa ra một số hướng phát triển tiếp theo cho bài toán như sau:

- Phát triển và cài đặt giao diện chương trình để có thể dễ dàng chạy và sử dụng hơn.
- Sử dụng thêm một vài phép xử lý ảnh để tăng cường dữ liệu phục vụ cho việc training
- Cải tiến, thay đổi các thuật toán để loại bỏ các bounding box trùng nhau nhanh hơn khi sử dụng thuật toán NMS; giúp chương trình chạy tối ưu hơn

# Tài liệu tham khảo

- [1] Vũ Hữu Tiệp. *Machine Learning cơ bản*. machinelearningcoban.com/ebook, 2018.
- [2] François Chollet. *Deep Learning with Python*. ©2018 Manning Publications Co.
- [3] Eli Stevens, Luca Antiga. *Deep Learning with PyTorch, Essential Excerpts*. ©2019 Manning Publications Co.
- [4] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Release 0.14.3, 8/2020
- [5] Tsung-Yi Lin, James Hays, Michael Maire, Serge Belongie, Pietro Perona, Deva Ramanan, Lubomir Bourdev, Ross Girshick, C. Lawrence Zitnick, Piotr Dollár. *Microsoft COCO: Common Objects in Context*. arXiv:1405.0312.
- [6] Ross Girshick. *Fast R-CNN*. arXiv:1504.08083.
- [7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. arXiv:1506.01497.
- [8] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. arXiv:1708.02002.
- [9] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. *Feature Pyramid Networks for Object Detection*. arXiv:1612.03144.