

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA KHOA HỌC MÁY TÍNH**

**BÁO CÁO ĐỒ ÁN CUỐI KÌ CS431**

**VanillaNet - Sức mạnh của Tối giản trong Học Sâu**



**GV hướng dẫn: Nguyễn Vĩnh Tiệp**

**Nhóm 7:**

<b>Họ và tên</b>	<b>MSSV</b>
Huỳnh Anh Dũng	22520278
Hà Huy Hoàng	22520460
Nguyễn Duy Hoàng	22520467
Lê Phước Trung	20522069

**TP.HCM, ngày 9 tháng 1 năm 2025**

# Mục lục

<b>1. Giới thiệu</b>	<b>1</b>
<b>2. Kiến trúc Vanilla Neural</b>	<b>1</b>
<b>3. Huấn luyện mạng VanillaNet</b>	<b>2</b>
3.1. Chiến lược đào tạo sâu . . . . .	2
3.2. Chồng chất hàm kích hoạt . . . . .	3
<b>4. Thực nghiệm</b>	<b>5</b>
4.1. Bộ dữ liệu . . . . .	5
4.1.1. Nội dung . . . . .	5
4.1.2. Nhãn phân loại . . . . .	6
4.1.3. Cảnh quan . . . . .	7
4.1.4. Tăng cường dữ liệu . . . . .	7
4.2. Bài toán thực nghiệm . . . . .	8
4.3. Phương pháp . . . . .	8
4.3.1. Thay lớp đầu cuối . . . . .	8
4.3.2. Huấn luyện . . . . .	10
4.3.3. Phát hiện vật thể trong ô trượt (sliding window) . . . . .	11
4.4. So sánh . . . . .	11
<b>5. Kết luận</b>	<b>14</b>

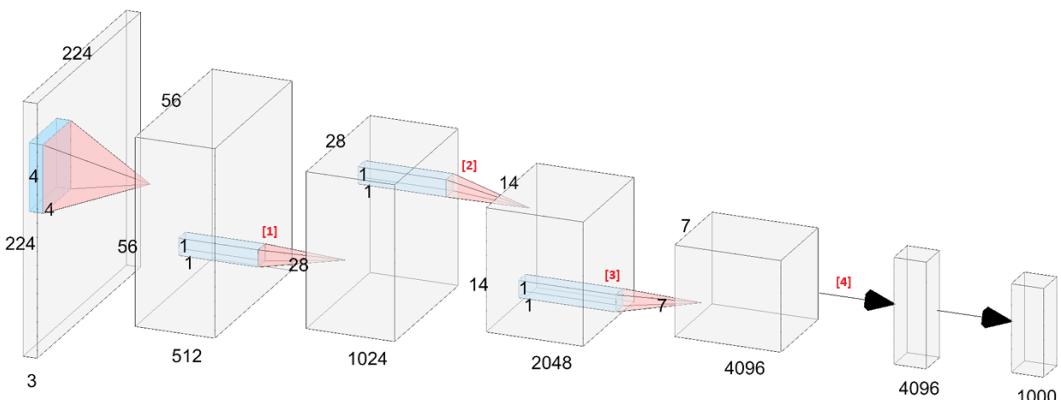
# 1. Giới thiệu

Triết lý nền tảng của các mô hình hiện nay là ‘càng nhiều càng khác biệt’ ('more is different'), điều này được chứng minh bằng các thành công vang dội trong lĩnh vực thị giác máy tính và xử lý ngôn ngữ tự nhiên. Một bước đột phá đáng chú ý trong các lĩnh vực này là sự phát triển của AlexNet [1], gồm 12 tầng và đạt hiệu suất hàng đầu trên chuẩn nhận dạng hình ảnh quy mô lớn. Xây dựng trên thành công này, ResNet [2] giới thiệu các ánh xạ đồng nhất thông qua skip connections, cho phép đào tạo các mạng nơ-ron sâu với hiệu suất cao trên nhiều ứng dụng thị giác máy tính, như phân loại hình ảnh, phát hiện đối tượng và phân đoạn ngữ nghĩa. Việc kết hợp các module được thiết kế bởi con người trong những mô hình này, cũng như sự tăng liên tục về độ phức tạp của mạng không thể phủ nhận rằng đã nâng cao hiệu suất.

Tuy nhiên, các thách thức về việc tối ưu hóa và sự phức tạp vốn có của các mô hình transformer kêu gọi một sự chuyển đổi mô hình hướng tới sự đơn giản. Trong báo cáo này, chúng em sẽ giới thiệu về **VanillaNet**, một cấu trúc mạng neural network mang tính tinh tế trong thiết kế. Bằng cách tránh độ sâu cao, các lối tắt, và các thao tác phức tạp như self-attention, VanillaNet vô cùng ngắn gọn nhưng lại đáng kinh ngạc về sức mạnh. Mỗi tầng được chế tác một cách cẩn thận để trở nên gọn gàng và thẳng thắn, với các hàm kích hoạt phi tuyến được loại bỏ sau quá trình huấn luyện để phục hồi kiến trúc ban đầu. VanillaNet vượt qua những thách thức của sự phức tạp vốn có, khiến nó trở nên lý tưởng cho các môi trường hạn chế tài nguyên.

## 2. Kiến trúc Vanilla Neural

Với sự phát triển của chip AI, vấn đề bottleneck về tốc độ suy luận của mạng neural không phải là FLOPs hay các tham số mà là do độ sâu và tính phức tạp trong thiết kế đã cản trở tốc độ. Để giải quyết vấn đề này, các tác giả của bài báo mà chúng em đang nghiên cứu đã đề xuất VanillaNet [3], có cấu trúc được thể hiện trong Hình 1. VanillaNet tuân theo thiết kế phổ biến của mạng neural với stem block, main body và lớp fully connected. Khác với các mạng sâu hiện có, VanillaNet chỉ sử dụng một lớp trong mỗi giai đoạn để thiết lập một mạng cực kỳ đơn giản với số lượng lớp ít nhất có thể.



Hình 1: Kiến trúc của mô hình VanillaNet-6

Chúng em sẽ làm rõ kiến trúc của VanillaNet trong phần này, lấy ví dụ là VanillaNet 6 như trong Hình 1. Đối với lớp stem, VanillaNet sẽ sử dụng một lớp tích chập  $4 \times 4 \times 3 \times C$  với stride là 4 để chuyển đổi ảnh đầu vào 3 kênh màu (RGB) thành  $C$  kênh đặc trưng. Tại giai đoạn 1, 2 và 3, một lớp maxpooling(stride=2) được sử dụng để giảm kích thước và feature map 2 lần đồng thời nó cũng tăng số lượng kênh lên gấp đôi sau mỗi giai đoạn. Ở giai đoạn 4, mạng không tăng số lượng kênh nữa mà thay vào đó là một lớp average pooling dùng để tóm tắt các thông tin đặc trưng đã học. Lớp cuối cùng là một lớp fully connected tương đương với output của mô hình phân loại này (lớp cuối cùng có kích thước  $1 \times 1 \times 1000$  vì mô hình này được huấn luyện trên bộ ImageNet-1K [4]). Kích thước nhân của mỗi lớp tích chập là  $1 \times 1$  bởi vì VanillaNet hướng tới việc sử dụng tối thiểu chi phí cho mỗi lần tính toán mỗi lớp mà vẫn giữ thông tin của các feature map. Hàm kích hoạt được áp dụng sau mỗi lớp tích chập  $1 \times 1$ . Để đơn giản cho quá trình training của mạng, batch normalization cũng được thêm vào sau mỗi lớp. Cần lưu ý rằng VanillaNet không có skip connections, vì theo như trong bài báo có đề cập là các tác giả thấy việc thêm skip connections cho thấy ít cải thiện hiệu suất. Theo nhóm bạn em, việc cải thiện hiệu suất bằng cách thêm skip connections không có hiệu quả cao bởi vì mạng như VanillaNet thường không có số lượng lớp rất sâu (như ResNet hoặc DenseNet). Do đó, các vấn đề như mất mát thông tin hoặc vanishing gradient ít nghiêm trọng hơn. Khi mạng không quá sâu, dòng chảy thông tin và gradient có thể được bảo toàn đủ tốt mà không cần skip connections. Điều này cũng mang lại một lợi ích khác là kiến trúc được đề xuất cực kỳ dễ thực hiện vì không có nhánh và các blocks bổ sung như khối squeeze và excitation.

Bởi vì kiến trúc của VanillaNet đơn giản và tương đối nồng nên tính phi tuyến tính yếu của nó gây ra hạn chế hiệu suất. Do đó, trong bài báo cũng đã đề xuất một loạt các kỹ thuật để giải quyết vấn đề này.

### 3. Huấn luyện mạng VanillaNet

Thông thường trong học sâu để nâng cao hiệu suất của các mô hình bằng cách giới thiệu khả năng mạnh mẽ hơn trong giai đoạn đào tạo [5]. Để đạt được mục tiêu này, bài báo đã đề xuất sử dụng kỹ thuật đào tạo sâu để nâng cao khả năng trong quá trình đào tạo trong VanillaNet được đề xuất, vì mạng sâu có tính phi tuyến tính mạnh hơn mạng nồng.

#### 3.1. Chiến lược đào tạo sâu

Ý tưởng chính của chiến lược đào tạo sâu là huấn luyện hai lớp tích chập với một hàm kích hoạt thay vì một lớp tích chập ở giai đoạn ban đầu của quá trình huấn luyện. Hàm kích hoạt được giảm dần dần thành một ánh xạ đồng nhất với số chu kỳ huấn luyện gia tăng. Ở cuối giai đoạn huấn luyện, hai lớp tích chập có thể dễ dàng kết hợp lại thành một lớp tích chập để giảm thời gian suy luận. Dạng ý tưởng này đã được sử dụng rộng rãi trong CNN [6, 7, 8, 9]. Sau đây, chúng em sẽ mô tả cách tiến hành kỹ thuật một cách chi tiết.

Cho hàm kích hoạt  $A(x)$  (các hàm thường sử dụng như ReLU và Tanh), chúng em kết hợp với một ánh xạ đồng nhất, có thể viết dưới dạng toán học như sau:

$$A'(x) = (1 - \lambda)A(x) + \lambda x \quad (1)$$

trong đó  $\lambda$  là một siêu tham số để cân bằng tính phi tuyến tính của việc hàm kích hoạt  $A'(x)$ .

Ký hiệu số chu kỳ hiện tại và tổng số chu kỳ của việc huấn luyện sâu lần lượt là  $e$  và  $E$ . Đặt  $\lambda = \frac{e}{E}$ . Vì vậy, khi bắt đầu giai đoạn huấn luyện ( $e = 0$ ),  $A'(x) = A(x)$ , có nghĩa rằng mạng sẽ có tính phi tuyến tính mạnh. Khi quá trình huấn luyện hội tụ, chúng ta sẽ có  $A'(x) = x$ , có nghĩa là giữa hai lớp tích chập không có bất cứ hàm kích hoạt nào. Sau đây, chúng em sẽ làm rõ cách nào có thể kết hợp hai lớp tích chập này.

Dầu tiên, chúng tôi chuyển đổi tất cả lớp batch normalization và lớp tích chập trước đó của nó thành một tích chập duy nhất. Chúng ta ký hiệu  $W \in \mathbb{R}^{C_{out} \times (C_{in} \times k \times k)}$ ,  $B \in \mathbb{R}^{C_{out}}$  như là ma trận trọng số và ma trận sai số của nhân tích chập với  $C_{in}$  kênh đầu vào,  $C_{out}$  kênh đầu ra và kích thước nhân  $k$ . Các giá trị tỷ lệ, dịch chuyển, trung bình và phương sai trong batch normalization được biểu diễn lần lượt là  $\gamma, \beta, \mu, \sigma \in R^{C_{out}}$ . Các ma trận trọng số và bias được hợp nhất như sau:

$$W'_i = \frac{\gamma_i}{\sigma_i} W_i, \quad B'_i = \frac{(B_i - \mu_i)\gamma_i}{\sigma_i} + \beta_i \quad (2)$$

trong đó  $i \in \{1, 2, 3, \dots, C_{out}\}$  ký hiệu cho giá trị trong kênh đầu ra thứ i.

Sau khi kết hợp lớp tích chập với batch normalization, chúng ta sẽ bắt đầu kết hợp hai lớp tích chập  $1 \times 1$ . Gọi  $x \in \mathbb{R}^{C_{in} \times H \times W}$  và  $y \in \mathbb{R}^{C_{out} \times H' \times W'}$  là các đặc trưng đầu vào và đặc trưng đầu ra, lớp tích chập có thể biểu diễn như:

$$y = W * x = W \cdot \text{im2col}(x) = W \cdot X \quad (3)$$

trong đó  $*$  là phép tích chập,  $\cdot$  là phép nhân ma trận và  $X \in \mathbb{R}^{(C_{in} \times 1 \times 1) \times (H' \times W')}$  xuất phát từ im2col [10] để chuyển đổi từ đầu vào thành một ma trận tương ứng với hình dạng của nhân. May mắn thay, đối với lớp convolution  $1 \times 1$ , các tác giả của bài báo thấy rằng im2col trở thành một phép reshape đơn giản bởi vì không cần trượt các hạt nhân có sự chồng chéo. Do đó, ký hiệu ma trận trọng số của hai lớp tích chập là  $W^1$  và  $W^2$ , hai lớp tích chập không có hàm kích hoạt có thể viết dưới dạng biểu thức sau:

$$y = W^1 * (W^2 * x) = W^1 \cdot W^2 \cdot \text{im2col}(x) = (W^1 \cdot W^2) * X \quad (4)$$

Do đó, lớp convolution  $1 \times 1$  có thể hợp nhất với nhau mà không giảm tốc độ suy luận.

### 3.2. Chồng chất hàm kích hoạt

Đã có những đề xuất về hàm kích hoạt cho mạng bao gồm hàm được sử dụng rộng rãi như Rectified Linear Unit (ReLU) và các biến thể của nó (PReLU [11], GeLU [12] và Swish [13]). Chúng chủ yếu tập trung vào nâng cao hiệu suất của mạng sâu vào phức tạp bằng cách sử dụng các hàm kích hoạt khác nhau. Tuy nhiên, như đã được chứng minh bởi các công trình đi trước [14, 15, 16], giới hạn về sức mạnh của các mạng nông và cơ bản thường là do tính phi tuyến tính yếu

của chúng, đây là điểm khác biệt so với các mạng sâu và phức tạp và do đó chưa được nghiên cứu đầy đủ.

Thực tế, có hai cách để cải thiện tính phi tuyến của mạng neural: xếp chồng các lớp kích hoạt phi tuyến hoặc tăng tính phi tuyến của từng lớp kích hoạt. Tuy nhiên, xu hướng của các mạng hiện tại thường chọn cách thứ nhất, điều này dẫn đến độ trễ cao khi khả năng tính toán song song vượt quá giới hạn.

Một ý tưởng trực quan để cải thiện tính phi tuyến của lớp kích hoạt là xếp chồng. Việc xếp chồng tuân tự các hàm kích hoạt là ý tưởng cốt lõi của các mạng neural sâu. Ngược lại, bài báo đã chuyển sang xếp chồng đồng thời các hàm kích hoạt. Ký hiệu một hàm kích hoạt đơn lẻ cho đầu vào  $x$  trong mạng nơron là  $A(x)$ , có thể là các hàm phổ biến như ReLU và Tanh. Việc xếp chồng đồng thời  $A(x)$  có thể được biểu diễn dưới dạng:

$$A_s(x) = \sum_{i=1}^n a_i A(x + b_i) \quad (5)$$

trong đó  $n$  là số lượng các hàm kích hoạt được xếp chồng và  $a_i, b_i$  lần lượt là hệ số tỷ lệ và độ lệch của từng hàm kích hoạt để tránh việc cộng dồn đơn giản. Tính phi tuyến của hàm kích hoạt có thể được cải thiện đáng kể bằng cách xếp chồng đồng thời. Phương trình 5 có thể được xem như một chuỗi trong toán học, đó là phép toán cộng nhiều đại lượng lại với nhau.

Để làm phong phú thêm khả năng xấp xỉ của chuỗi, VanillaNet cho phép hàm dựa trên chuỗi này học thông tin toàn cục bằng cách thay đổi các đầu vào từ các lân cận của chúng, điều này tương tự như BNEN [17]. Cụ thể, với một đặc trưng đầu vào  $x \in \mathbb{R}^{H \times W \times C}$ , trong đó  $H, W$  và  $C$  lần lượt là chiều rộng, chiều cao và số lượng kênh của nó, hàm kích hoạt được biểu diễn như sau:

$$A_s(x_h, w, c) = \sum_{i,j \in -n,n} a_{i,j,c} A(x_{i+h,j+w,c} + b_c) \quad (6)$$

trong đó  $h \in \{1, 2, \dots, H\}$ ,  $w \in \{1, 2, \dots, W\}$  và  $c \in \{1, 2, \dots, C\}$ . Để thấy rằng khi  $n = 0$ , hàm kích hoạt dựa trên chuỗi  $A_s(x)$  suy giảm thành hàm kích hoạt thông thường  $A(x)$ , điều này có nghĩa rằng phương pháp đề xuất có thể được xem như một sự mở rộng tổng quát của các hàm kích hoạt hiện có. VanillaNet sử dụng ReLU làm hàm kích hoạt cơ bản để xây dựng chuỗi này, vì nó hiệu quả trong việc suy luận trên GPU.

Chúng tôi phân tích thêm độ phức tạp tính toán của hàm kích hoạt được đề xuất so với lớp tích chập tương ứng của nó. Đối với một lớp tích chập với kích thước kernel  $K$ , số kênh đầu vào  $C_{in}$  và số kênh đầu ra  $C_{out}$ , độ phức tạp tính toán là:

$$\mathcal{O}(CONV) = H \times W \times C_{in} \times C_{out} \times k^2 \quad (7)$$

trong khi đó độ phức tạp tính toán của lớp chuỗi kích hoạt là:

$$\mathcal{O}(SA) = H \times W \times C_{in} \times n^2 \quad (8)$$

Do đó, ta có:

$$\frac{\mathcal{O}(CONV)}{\mathcal{O}(SA)} = \frac{H \times W \times C_{in} \times C_{out} \times k^2}{H \times W \times C_{in} \times n^2} = \frac{C_{out} \times k^2}{n^2} \quad (9)$$

Lấy giai đoạn 3 trong VanillaNet-6 làm ví dụ, trong đó  $k = 1$  (như đã nói ở phần 2),  $C_{out} = 4096$  và  $n = 3$  (giá trị mặc định trong code của bài báo [3]), tỷ lệ vào khoảng 455. Kết luận, chi phí tính toán của hàm kích hoạt được đề xuất vẫn thấp hơn nhiều so với các lớp tích chập. Phân tích chi tiết hơn về độ phức tạp thực nghiệm sẽ được trình bày trong phần sau.

## 4. Thực nghiệm

### 4.1. Bộ dữ liệu

Để tiến hành thực nghiệm hiệu năng mô hình VanillaNet, nhóm nghiên cứu quyết định sử dụng bộ dữ liệu “Ships in Satellite Imagery” [18] của tác giả Rhammell đăng tải trên Kaggle.

Hình ảnh từ vệ tinh cung cấp thâu quan đến nhiều lĩnh vực trong cuộc sống. Nhưng với sự phát triển đầy bùng nổ của ngành công nghiệp hàng không vũ trụ, việc lượng dữ liệu đó dồn vượt mức xử lý thủ công của con người là không thể tránh khỏi. Mục tiêu của bộ dữ liệu đề ra hướng đến việc giải quyết bài toán hóc búa trong việc phát hiện vị trí tàu thủy từ vệ tinh. Việc tự động hóa bài toán này trên được cho rằng sẽ thúc đẩy sự phát triển của bộ máy quản lý chuỗi cung ứng, cụ thể hơn, trong lĩnh vực xuất nhập cảnh hải cảng.

#### 4.1.1. Nội dung

Hình ảnh trong bộ dữ liệu được trích xuất từ bộ ảnh chụp của dịch vụ vệ tinh Planet, trong phạm vi vùng vịnh San Francisco và San Pedro bang California, bao gồm 4000 tấm ảnh có kích thước 80x80, lưu ở định dạng RGB. Mỗi ảnh đều mang nhãn phân loại bản thân ảnh là “ship” hoặc “no-ship”.

Một thư mục nén (.zip) được cung cấp, chứa toàn bộ tập dữ liệu dưới dạng hình ảnh .png. Tên tệp của từng hình ảnh tuân theo định dạng sau:

{label} \_ {scene id} \_ {longitude} \_ {latitude}.png

Chi tiết:

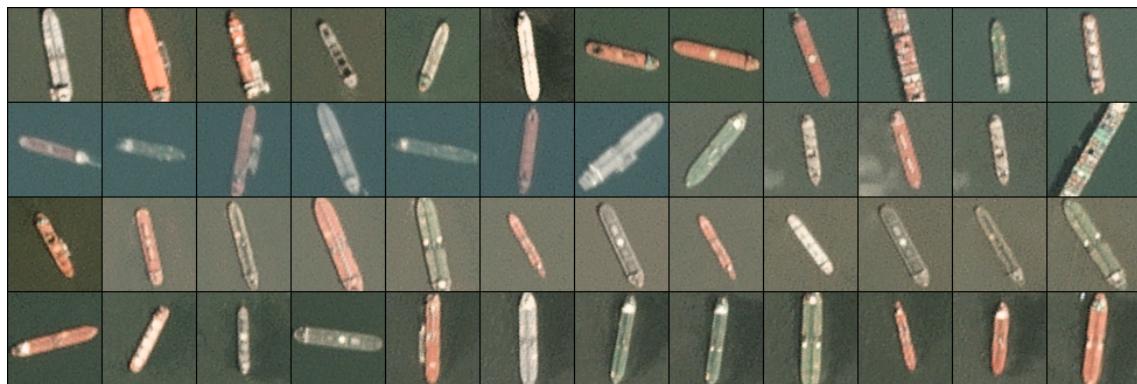
- label: Giá trị 1 hoặc 0, lần lượt đại diện cho lớp “ship” (tàu) và lớp “no-ship” (không phải tàu).
- scene id: Mã định danh duy nhất của cảnh trực quan PlanetScope mà hình ảnh được trích xuất. Mã này có thể được sử dụng với Planet API để tìm kiếm và tải xuống toàn bộ cảnh.
- longitude\_latitude: Tọa độ kinh độ và vĩ độ của điểm trung tâm hình ảnh, với giá trị được phân tách bằng dấu gạch dưới (\_).

Ngoài ra, còn có một file JSON (file này nhóm không sử dụng trong quá trình thực nghiệm) chứa:

- data: Dữ liệu pixel của từng ảnh.
  - 6400 phần tử đầu tiên: Giá trị kênh màu đỏ (red).
  - 6400 phần tử tiếp theo: Giá trị kênh màu xanh lá cây (green).
  - 6400 phần tử cuối cùng: Giá trị kênh màu xanh dương (blue).
- label: Nhãn của hình ảnh (1 hoặc 0).
- scene\_ids: Danh sách mã định danh cảnh tương ứng.
- locations: Tọa độ tương ứng.

#### 4.1.2. Nhãn phân loại

Lớp “ship” (tàu) bao gồm 1000 hình ảnh. Các hình ảnh trong lớp này được căn giữa vào thân của một con tàu duy nhất. Các hình ảnh bao gồm tàu có nhiều kích thước, hướng và điều kiện khí quyển khác nhau.



Hình 2: Một số ảnh trong bộ gán nhãn “ship”

Lớp “no-ship” (không phải tàu) bao gồm 3000 hình ảnh. Các hình ảnh trong lớp này được chia thành ba nhóm:

- Một phần ba đầu tiên: Là tập hợp ngẫu nhiên các đặc điểm địa hình khác nhau - nước, thảm thực vật, đất trống, tòa nhà, v.v. - không bao gồm bất kỳ phần nào của một con tàu.
- Một phần ba tiếp theo: Là các "phần của tàu"(partial ships) chỉ chứa một phần của con tàu, nhưng không đủ để đáp ứng định nghĩa đầy đủ của lớp "có tàu".
- Một phần ba cuối cùng: Là các hình ảnh từng bị dán nhãn sai bởi các mô hình học máy, thường do các điểm ảnh sáng hoặc các đặc điểm đường thẳng nổi bật.

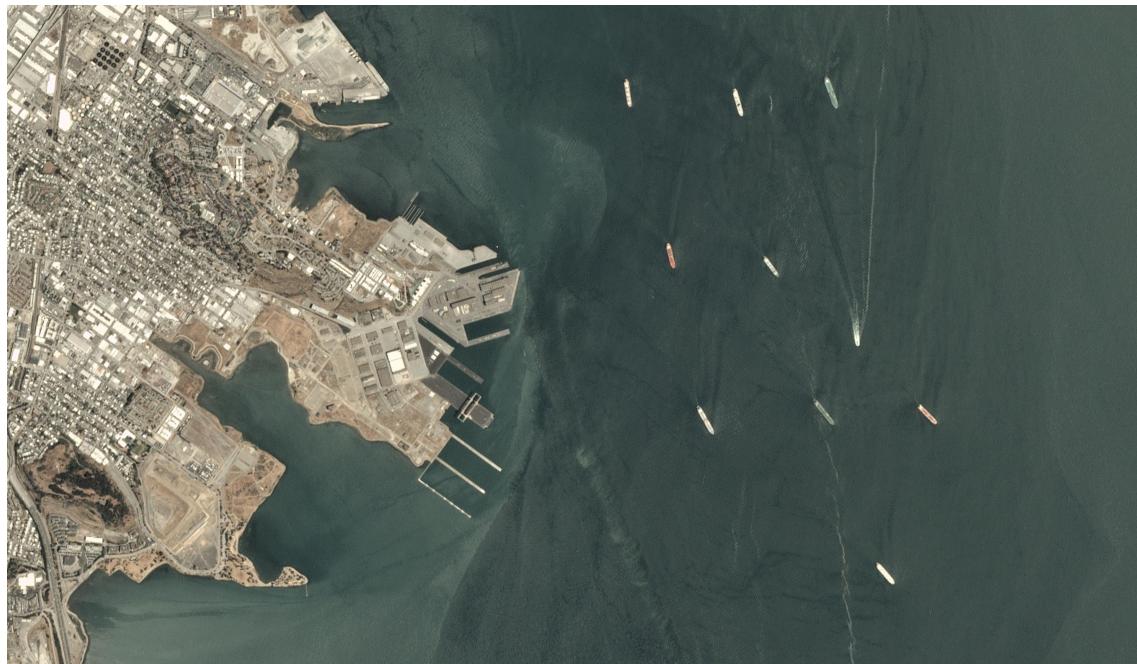
Các ví dụ về hình ảnh từ lớp này được hiển thị dưới đây.



Hình 3: Một số hình ảnh của nhãn “no-ship”

#### 4.1.3. Cảnh quan

Tám hình ảnh toàn cảnh đầy đủ được bao gồm trong thư mục scenes. Các cảnh này có thể được sử dụng để trực quan hóa hiệu suất của các mô hình phân loại đã được huấn luyện trên tập dữ liệu.



Hình 4: Một ảnh toàn cục trong bộ dữ liệu

#### 4.1.4. Tăng cường dữ liệu

Nhận thấy số lượng ảnh không thực sự nhiều và đa dạng để các mô hình tiền huấn luyện có thể biểu diễn được hết khả năng, cũng như xét về tỉ lệ lệch lớp giữa các nhãn “no-ship” và “ship” khá lớn, cụ thể là 3:1 lần lượt theo thứ tự đã nêu, nhóm đã áp dụng một số biện pháp tăng cường dữ liệu tuy đơn giản nhưng đủ để tăng thêm tính đa dạng cũng như tạo điều kiện cho mô hình tiếp xúc với nhiều loại phân bố nhiều khác nhau của cùng một khuôn mẫu, từ đó tăng cường tính phổ quát hơn cho mô hình.

Các biện pháp tăng cường nhóm sử dụng bao gồm:

- Horizontal Flip: Lật ảnh theo phương ngang.
- Vertical Flip: Lật ảnh theo phương đứng.
- Shift Scale Rotate: Xê dịch, biến đổi kích cỡ và xoay ảnh.
- Random Brightness Contrast: Thay đổi ngẫu nhiên độ sáng và độ tương phản ảnh.
- Gaussian Blur: Làm mờ Gaussian.
- Gauss Noise: Tạo nhiễu theo phân phối Gaussian.

Các biện pháp tăng cường đều được nhóm áp dụng theo tỉ xuất xảy ra ngẫu nhiên 50%, bộ ảnh sau khi tăng cường được nhóm lưu vào ổ cứng cho mục đích tái sử dụng.

## 4.2. Bài toán thực nghiệm

Nhằm thực nghiệm khả năng trích xuất đặc trưng của mô hình, nhóm quyết định giải bài toán phát hiện vật thể trên bộ dữ liệu vừa nêu.

Bài toán trên được nhóm cài đặt qua hai bài toán con: phân loại và gán nhãn cho sliding window trong cảnh quan.

## 4.3. Phương pháp

Để thực hiện bài toán phân loại tàu thủy và bối cảnh, nhóm cần xây dựng được một mô hình học sâu phù hợp có số lớp đầu ra là hai lớp. Ý tưởng chính vẫn là dựa trên mô hình VanillaNet để cấu nên một mô hình mới sao cho phù hợp với yêu cầu. Nhận thấy VanillaNet, với số lớp đầu ra 1000, không thực sự thỏa mãn được yêu cầu bài toán, nhóm quyết định thay lớp fully-connected đầu cuối của mô hình ban đầu thành lớp có chức năng tương tự nhưng cho đầu ra phân loại hai lớp.

### 4.3.1. Thay lớp đầu cuối

Nhờ tính chất được phát triển với Pytorch, một thư viện hỗ trợ *dynamic computing graph* (tạm dịch: đồ thị tính toán động), nhóm có thể dễ dàng xem cấu trúc của mô hình trên thực tế một cách đơn giản, chỉ với việc *print* mô hình ra màn hình hiển thị trong môi trường phát triển,

```

VanillaNet(
    (stages): ModuleList(
        (0): Block(
            (pool): Identity()
            (act): activation()
            (conv): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (1): Block(
            (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
            (act): activation()
            (conv): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
        )
        (2): Block(
            (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
            (act): activation()
            (conv): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1))
        )
        (3-5): 3 x Block(
            (pool): Identity()
            (act): activation()
            (conv): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1))
        )
        (6): Block(
            (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
            (act): activation()
            (conv): Conv2d(2048, 4096, kernel_size=(1, 1), stride=(1, 1))
        )
        (7): Block(
            (pool): Identity()
            (act): activation()
            (conv): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
        )
    )
    (stem): Sequential(
        (0): Conv2d(3, 512, kernel_size=(4, 4), stride=(4, 4))
        (1): activation()
    )
    (cls): Sequential(
        (0): AdaptiveAvgPool2d(output_size=(1, 1))
        (1): Dropout(p=0, inplace=False)
        (2): Conv2d(4096, 1000, kernel_size=(1, 1), stride=(1, 1))
    )
)

```

Hình 5: Sơ bộ về thuộc tính lớp VanillaNet trong Pytorch

so với mô hình nhóm thực hiện thay đổi.

```

VanillaNetCustomized(
    (backbone): VanillaNet(
        (stages): ModuleList(
            (0): Block(
                (pool): Identity()
                (act): activation()
                (conv): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
            )
            (1): Block(
                (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (act): activation()
                (conv): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
            )
            (2): Block(
                (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (act): activation()
                (conv): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1))
            )
            (3-5): 3 x Block(
                (pool): Identity()
                (act): activation()
                (conv): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1))
            )
            (6): Block(
                (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (act): activation()
                (conv): Conv2d(2048, 4096, kernel_size=(1, 1), stride=(1, 1))
            )
            (7): Block(
                (pool): Identity()
                (act): activation()
                (conv): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
            )
        )
        (stem): Sequential(
            (0): Conv2d(3, 512, kernel_size=(4, 4), stride=(4, 4))
            (1): activation()
        )
    )
    (cls): Linear(in_features=4096, out_features=2, bias=True)
)

```

Hình 6: Thuộc tính lớp VanillaNet sau khi thay lớp

Nhờ việc Pytorch xem các modules xuất hiện trong mô hình như thuộc tính của lớp mô hình đó, nhóm dễ dàng thực hiện việc loại bỏ và thay lớp bằng các lệnh `delattr` và lệnh gán cho thuộc tính `cls` của mô hình một lớp `Linear` mới có số đặc trưng đầu ra là hai.

Cài đặt ban đầu của VanillaNet sử dụng lớp tích chập kích thước nhân 1x1 để thay thế cho lớp fully-connected, nhóm nhận thấy rằng việc này không thực sự cần thiết, đặc biệt khi phương pháp huấn luyện sâu không được áp dụng trong thực nghiệm, nhóm đề xuất việc tái cấu trúc mô hình với lớp fully-connected đồng thời định nghĩa luồng dữ liệu `forward` mới bên trong của mô hình sao cho thỏa mãn cả cấu trúc xây dựng, cả đồ thị tính toán ban đầu.

#### 4.3.2. Huấn luyện

Mô hình đưa vào huấn luyện chỉ có lớp fully-connected đầu cuối được cập nhật tham số, các lớp còn lại sẽ được đóng bằng để đảm bảo khả năng trích xuất đặc trưng của mô hình.

Mô hình được đưa vào đối tượng Trainer của thư viện lightning-pytorch để tự động hóa việc huấn luyện. Nhóm sử dụng một số hàm callbacks như EarlyStopping và ModelCheckpoint để đảm bảo việc huấn luyện mô hình tiến gần trạng thái tối ưu nhất có thể.

Kết quả huấn luyện cho thấy mô hình VanillaNet 10 lớp cho biểu đồ hội tụ trên train\_loss khá ổn định, với biên độ dao động nhỏ; giá trị F1 trung bình sau các bước huấn luyện dao động ở ngưỡng khá cao trở lên, cụ thể là trên 0.8.

#### 4.3.3. Phát hiện vật thể trong ô trượt (sliding window)

Sau khi hoàn thành việc huấn luyện mô hình trên bộ dữ liệu ảnh cắt cảnh, nhóm tiếp tục thực nghiệm với bài toán phát hiện vật thể trên hình ảnh toàn cục từ vệ tinh. Để có thể đưa vào ứng dụng một mô hình phân loại cho bài toán phát hiện vật thể, nhóm đề xuất sử dụng kĩ thuật ô trượt với vùng chồng lấp để phân nhỏ bản thể lớn của tấm hình sang nhiều phân cảnh nhỏ phù hợp với tham số đầu vào của mô hình hơn mà vẫn biểu diễn đầy đủ nội dung ban đầu.

Sau khi thử một khoảng giá trị ngưỡng xác suất lớp phân loại, nhóm nhận định rằng khoảng giá trị 0.4 đến 0.5 là mức hiệu quả nhất cho mô hình VanillaNet trên bài toán này. Kết quả bao khung thu được được đánh giá cảm quan theo góc nhìn người nghiên cứu là đạt được độ chính xác tốt và ổn định, không đưa ra dự đoán vô nghĩa.

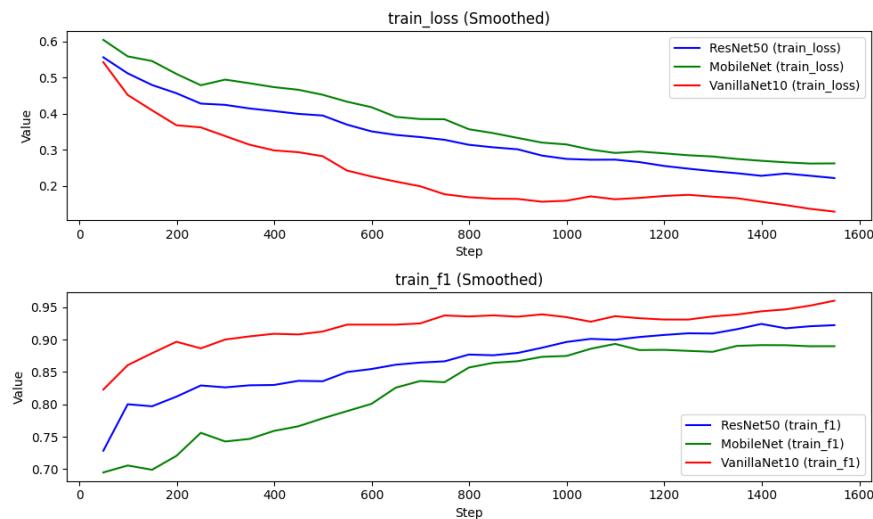
### 4.4. So sánh

Để có thể đưa ra nhận định trực quan và ý nghĩa về một phương pháp cải tiến, việc so sánh phương pháp đó với những phương pháp nổi tiếng và uy tín đã được tin dùng trong lĩnh vực là không thể thiếu được. Đến với phần so sánh thực nghiệm của nhóm, hai phương pháp MobileNet và ResNet50 là những phương pháp được nhóm lựa chọn, lần lượt đại diện cho nhóm mô hình nhỏ gọn và nhóm mang cấu trúc phức tạp.

Hai mô hình được đưa qua quá trình thực nghiệm tương tự như VanillaNet. Tuy vậy, có một số điểm đáng lưu ý. Với mô hình đầu tiên, MobileNet có thời gian hoàn thành mỗi batch huấn luyện nhỏ hơn so với VanillaNet, tuy nhiên độ chính xác lại thấp hơn và khả năng hội tụ của mô hình trên tập huấn luyện cũng nhỏ hơn đáng kể, dẫn đến tổng thời gian huấn luyện cho MobileNet trở nên lớn hơn VanillaNet khá nhiều. Còn với ResNet50, nhận định tương tự cũng có thể được áp dụng cho mô hình này, khả năng hội tụ cũng như độ chính xác có thể gọi là tương đồng với MobileNet mặc dù kết cấu mô hình có phần phức tạp hơn. Xét về thời gian hoàn thành mỗi batch huấn luyện, VanillaNet lép vế so với hai mô hình tiêu điểm là vì bản thân VanillaNet có số lượng tham số rất lớn so với độ phức tạp cấu trúc mô hình, dẫn đến quá trình truy hồi ngược phải mất nhiều thời gian hơn để hoàn thành; tuy vậy, cũng nhờ cài đặt đặc biệt mà khi đưa vào dự đoán, VanillaNet “bứt tốc” so với hai mô hình còn lại, sở dĩ như vậy là vì số FLOPs của VanillaNet hơn gấp nhiều lần so với MobileNet hay ResNet50.

Model Comparison:			
Model	Params(M)	FLOPs(B)	Depth
VanillaNet10	41.61	1167.36	4
MobileNet	2.23	41.75	6
ResNet50	23.51	528.86	5

Hình 7: So sánh VanillaNet10, ResNet50 và MobileNet



Hình 8: Biểu đồ tiêu chí huấn luyện của VanillaNet10, ResNet50 và MobileNet



Hình 9: Ảnh dự đoán của VanillaNet10 trên ảnh toàn cảnh



Sliding window: 100% |██████████| 7854/7854 [00:57<00:00, 135.56window/s]

Hình 10: Ảnh dự đoán của MobileNet trên ảnh toàn cảnh



Sliding window: 100% |  | 7854/7854 [01:04<00:00, 121.97window/s]

Hình 11: Ảnh dự đoán của ResNet50 trên ảnh toàn cảnh

## 5. Kết luận

Sau khi tìm hiểu công trình của đội ngũ nghiên cứu VanillaNet, cũng như thực nghiệm so sánh với một số mô hình đáng tin cậy trong một bài toán đầy tính thực tế, nhóm đưa ra một số nhận định. VanillaNet là một mô hình mang nặng tính đột phá, công trình nghiên cứu VanillaNet đặt gạch cho một lối đi chêch đường mòn, có thể nói là càng lúc càng sâu hơn vào lõi hút của sự phức tạp của xu hướng phát triển mạng học nơ-ron hiện nay. Đơn giản và mạnh mẽ là những từ tuy chưa thể tóm gọn hết độ đặc biệt của VanillaNet trong loạt các nghiên cứu đổi mới những năm trở lại, nhưng cũng đủ để diễn đạt được sức hút của loại mô hình này với hướng phát triển của tương lai.

## Tài liệu

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
- [3] H. Chen, Y. Wang, J. Guo, and D. Tao, “Vanillanet: the power of minimalism in deep learning,” 2023.
- [4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, (United States), pp. 248–255, IEEE Computer Society, 2009. Publisher Copyright: © 2009 IEEE.; 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009 ; Conference date: 20-06-2009 Through 25-06-2009.
- [5] H. Chen, Y. Wang, C. Xu, C. Xu, C. Xu, and T. Zhang, “Universal adder neural networks,” 2021.
- [6] X. Ding, X. Zhang, J. Han, and G. Ding, “Diverse branch block: Building a convolution as an inception-like unit,” 2021.
- [7] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, “Repvgg: Making vgg-style convnets great again,” 2021.
- [8] X. Ding, X. Zhang, Y. Zhou, J. Han, G. Ding, and J. Sun, “Scaling up your kernels to 31x31: Revisiting large kernel design in cnns,” 2022.
- [9] X. Ding, Y. Guo, G. Ding, and J. Han, “Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks,” 2019.
- [10] B. QoChuk and OpenGenus team, “im2col convolution.”
- [11] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imangenet classification,” 2015.

- [12] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” 2023.
- [13] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” 2017.
- [14] H. Mhaskar and T. Poggio, “Deep vs. shallow networks : An approximation theory perspective,” 2016.
- [15] R. Eldan and O. Shamir, “The power of depth for feedforward neural networks,” 2016.
- [16] M. Telgarsky, “Benefits of depth in neural networks,” 2016.
- [17] Y. Xu, L. Xie, C. Xie, J. Mei, S. Qiao, W. Shen, H. Xiong, and A. Yuille, “Batch normalization with enhanced linear transformation,” 2020.
- [18] R. Hammell, “Ships in satellite imagery,” 2018.