TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG

BÁO CÁO
HỌC MÁY VÀ TRÍ TUỆ NHÂN TẠO

GVHD: Huỳnh Thế Thiện
SV thực hiện:
Nguyễn Thành Thịnh(22119234)
Lê Vũ Huệ Trọng(22119246)

TP Hồ Chí Minh, Tháng 4 Năm 2025

Mục lục

1 Tổng quan về học sâu

Học sâu (Deep Learning) là một nhánh con của học máy (Machine Learning), sử dụng các mô hình mạng nơ-ron nhân tạo với nhiều lớp để học và phân tích dữ liệu phức tạp. Các mô hình học sâu đã đạt được thành tựu đáng kể trong các nhiệm vụ như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên, và phân tích dữ liệu thời gian.
Học sâu có khả năng học tự động các đặc trưng (features) từ dữ liệu mà không cần phải lập trình thủ công, giúp nó trở thành một công cụ mạnh mẽ trong nhiều lĩnh vực, từ chăm sóc sức khỏe, tài chính đến giao thông.

1.1 Các loại mạng nơ-ron trong học sâu

- **Mạng nơ-ron tích chập (CNN)**: Được sử dụng phổ biến trong các nhiệm vụ xử lý hình ảnh và video. Mạng CNN tận dụng tính chất không gian của dữ liệu hình ảnh để học các đặc trưng từ các lớp của mạng nơ-ron.
- **Mạng nơ-ron hồi tiếp (RNN)**: Được thiết kế để xử lý dữ liệu tuần tự như văn bản hoặc chuỗi thời gian. Mạng RNN có khả năng ghi nhớ thông tin từ các bước trước trong chuỗi dữ liệu, giúp hiểu các mối quan hệ trong các dữ liệu tuần tự.
- **Mạng nơ-ron chuyển tiếp đa lớp (MLP)**: Là dạng mạng nơ-ron cơ bản với các lớp ẩn, được sử dụng trong nhiều nhiệm vụ học máy đơn giản.

1.2 Ứng dụng của học sâu

Học sâu đã được áp dụng rộng rãi trong nhiều lĩnh vực:

- **Nhận diện hình ảnh**: Các mạng CNN có thể phát hiện và phân loại các đối tượng trong ảnh, ví dụ như nhận diện khuôn mặt, xe cộ, hay các bệnh lý trong y tế.
- **Xử lý ngôn ngữ tự nhiên (NLP)**: Học sâu giúp cải thiện các ứng dụng như dịch máy, phân tích cảm xúc, và trả lời câu hỏi tự động.
- **Hệ thống tự lái**: Học sâu được sử dụng trong các hệ thống lái xe tự động, giúp xe nhận diện và hiểu môi trường xung quanh.

1.3 Thách thức trong học sâu

Dù học sâu có nhiều ưu điểm, nhưng vẫn có một số thách thức lớn:

- **Yêu cầu dữ liệu lớn**: Các mô hình học sâu cần một lượng lớn dữ liệu để huấn luyện, điều này đôi khi không khả thi trong các lĩnh vực dữ liệu hạn chế.
- **Khó khăn trong giải thích mô hình**: Mạng nơ-ron, đặc biệt là các mạng sâu, thường được coi là "hộp đen", tức là khó có thể giải thích chi tiết cách mà mô hình đưa ra quyết định.
- **Tính toán phức tạp**: Các mô hình học sâu yêu cầu phần cứng mạnh mẽ (GPU) và thời gian huấn luyện dài, điều này có thể tạo ra chi phí cao.

2 Các thuật toán cơ bản

2.1 Lan truyền ngược (Backpropagation)

Lan truyền ngược (Backpropagation) là thuật toán quan trọng trong quá trình huấn luyện mạng nơ-ron. Thuật toán này giúp tính toán gradient của hàm mất mát (loss function) và sử dụng các gradient này để cập nhật trọng số (weights) của mạng nơ-ron sao cho mạng giảm thiểu sai số trong dự đoán.
Quá trình này có thể được chia thành các bước chính sau đây:

2.1.1 Forward Propagation

Đầu tiên, dữ liệu đầu vào được truyền qua các lớp trong mạng nơ-ron để tạo ra dự đoán (output). Đầu vào x được truyền qua các lớp của mạng, mỗi lớp thực hiện phép toán cộng trọng số và độ lệch (bias), sau đó áp dụng hàm kích hoạt (activation function). Công thức tính toán đầu ra của mạng là:

$$y_{\text{pred}} = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Trong đó: - x_i là giá trị đầu vào, - w_i là trọng số tương ứng với đầu vào x_i, - b là độ lệch (bias), - f là hàm kích hoạt.

2.1.2 Tính toán lỗi (Loss Calculation)

Hàm mất mát (loss function) được sử dụng để đo lường mức độ sai lệch giữa giá trị dự đoán và giá trị thực tế. Một trong những hàm mất mát phổ biến là **Mean Squared Error (MSE)**, công thức tính là:

$$L = \frac{1}{n}\sum_{k=1}^{n}(o_k - z_k)^2$$

Trong đó: - o_k là giá trị thực (ground truth) của điểm dữ liệu thứ k, - z_k là giá trị dự đoán của mạng nơ-ron cho điểm dữ liệu thứ k, - n là số lượng mẫu trong tập huấn luyện.

2.1.3 Lan truyền ngược (Backpropagation)

Ở bước này, thuật toán **Backpropagation** tính toán gradient của hàm mất mát đối với các trọng số trong mạng nơ-ron. Sử dụng **chuỗi đạo hàm (chain rule)** trong giải tích, chúng ta có thể tính toán gradient một cách hiệu quả qua các lớp của mạng.
Công thức tính gradient cho trọng số $w_{j,i}$ là:

$$\frac{\partial L}{\partial w_{j,i}} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial y_1} \cdot \frac{\partial y_1}{\partial w_{j,i}}$$

Trong đó: - $\frac{\partial L}{\partial z_1}$ là gradient của hàm mất mát theo đầu ra của nơ-ron, - $\frac{\partial z_1}{\partial y_1}$ là gradient của đầu ra của nơ-ron theo tính toán giữa các nơ-ron, - $\frac{\partial y_1}{\partial w_{j,i}}$ là gradient của đầu ra của nơ-ron theo trọng số.
Công thức trên có thể được mở rộng cho tất cả các trọng số trong mạng, và các gradient sẽ được tính toán cho từng lớp trong mạng.

2.1.4 Công thức chính thức của gradient

Công thức gradient cuối cùng sẽ được tính như sau, với z_k là đầu ra của mạng và o_k là giá trị thực (ground truth):

$$\frac{\partial L}{\partial w_{j,i}} = \frac{2}{n} \sum_{k=1}^{n} (z_k - o_k) \cdot z_k(1 - z_k) \cdot x_{k,i}$$

Trong đó: - $x_{k,i}$ là giá trị đầu vào của đặc trưng thứ i trong dữ liệu điểm thứ k, - z_k là giá trị dự đoán của mạng cho điểm dữ liệu thứ k, - o_k là giá trị thực của điểm dữ liệu thứ k, - n là số lượng mẫu trong tập huấn luyện.

2.1.5 Cập nhật trọng số (Weight Update)

Sau khi tính toán gradient, trọng số trong mạng nơ-ron sẽ được cập nhật bằng cách sử dụng **Gradient Descent**. Quy tắc cập nhật trọng số là:

$$w_{j,i}^{\text{new}} = w_{j,i}^{\text{old}} - \eta \cdot \frac{\partial L}{\partial w_{j,i}}$$

Trong đó: - η là **learning rate**, điều khiển tốc độ học (bước nhảy) của quá trình cập nhật trọng số, - $\frac{\partial L}{\partial w_{j,i}}$ là gradient của hàm mất mát theo trọng số $w_{j,i}$.
Quá trình này sẽ được lặp lại cho tất cả các trọng số trong mạng cho mỗi điểm dữ liệu hoặc mỗi batch dữ liệu trong tập huấn luyện.

2.1.6 Cải tiến trong quá trình Backpropagation

- **Gradient Clipping**: Được sử dụng để giới hạn gradient khi chúng quá lớn, tránh hiện tượng **exploding gradients**. - **Momentum**: Là một kỹ thuật tối ưu hóa giúp làm mượt quá trình cập nhật trọng số bằng cách duy trì một phần tốc độ cập nhật trước đó. - **Adam (Adaptive Moment Estimation)**: Kết hợp cả **Momentum** và **RMSProp**, giúp tự động điều chỉnh learning rate cho từng trọng số.

2.1.7 Ví dụ Tính Toán Cập Nhật Trọng Số

Giả sử bạn có hàm mất mát đơn giản:

$$Loss(w) = w^2 - 4w + 5$$

Để tìm giá trị tối ưu của w, bạn cần tính đạo hàm của hàm mất mát và cập nhật trọng số w bằng cách sử dụng **Gradient Descent**.
Đạo hàm của hàm mất mát là:

$$\frac{\partial Loss}{\partial w} = 2w - 4$$

Sau đó, sử dụng **Gradient Descent** để cập nhật trọng số:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot (2w - 4)$$

Quá trình này được lặp lại cho đến khi giá trị w hội tụ về giá trị tối ưu.
Với cách trình bày này, phần **Backpropagation** đã trở nên đầy đủ hơn với các công thức chi tiết và chứng minh. Bạn có thể sử dụng đoạn mã LaTeX trên để làm phong phú phần báo cáo

của mình. Nếu bạn cần thêm bất kỳ điều chỉnh nào hoặc muốn giải thích thêm, đừng ngần ngại yêu cầu!

2.2 Tối ưu hóa dựa trên gradient (Gradient-based Optimization)

Tối ưu hóa dựa trên gradient là phương pháp tối ưu hóa trong đó trọng số của mô hình được điều chỉnh dựa trên đạo hàm (gradient) của hàm mất mát. Mục tiêu của quá trình này là giảm thiểu sự sai lệch giữa đầu ra dự đoán và giá trị thực tế bằng cách cập nhật trọng số của mạng. Gradient-based optimization bao gồm nhiều kỹ thuật, như **Gradient Descent**, **Stochastic Gradient Descent (SGD)**, **Momentum**, **RMSProp**, và **Adam**, mỗi kỹ thuật có những ưu điểm và ứng dụng riêng.

2.2.1 Gradient Descent (GD)

Gradient Descent là thuật toán tối ưu cơ bản trong học sâu. Thuật toán này điều chỉnh trọng số của mạng theo hướng của gradient hàm mất mát, với mục tiêu tối thiểu hóa hàm mất mát.

Công thức Gradient Descent

Quy tắc cập nhật trọng số trong **Gradient Descent** được cho bởi:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Trong đó: - w là trọng số, - η là **learning rate**, là tốc độ học của mô hình, - $\frac{\partial L}{\partial w}$ là gradient của hàm mất mát đối với trọng số w.
Quá trình này lặp lại nhiều lần cho đến khi hàm mất mát hội tụ (đạt cực tiểu).
Ưu điểm:

- Đơn giản và dễ triển khai.
- Có thể áp dụng cho mọi loại mô hình học máy.

Nhược điểm:

- Tính toán chậm nếu dữ liệu quá lớn vì phải tính toán gradient trên toàn bộ tập dữ liệu (batch gradient descent).

2.2.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) là một biến thể của **Gradient Descent**. Thay vì tính toán gradient trên toàn bộ tập dữ liệu, **SGD** cập nhật trọng số sau khi tính toán gradient từ một mẫu ngẫu nhiên trong tập huấn luyện.

Công thức cập nhật trong SGD

$$w = w - \eta \cdot \frac{\partial L(x_i, y_i)}{\partial w}$$

Trong đó: - (x_i, y_i) là một mẫu ngẫu nhiên từ tập huấn luyện.
Ưu điểm:

- Tốc độ huấn luyện nhanh hơn do chỉ sử dụng một mẫu tại mỗi bước.

- Có thể tìm kiếm các cực tiểu toàn cục tốt hơn trong không gian tham số không lồi.

Nhược điểm:

- Cập nhật không ổn định, gây dao động trong hàm mất mát.
- Thường cần điều chỉnh learning rate và có thể sử dụng các kỹ thuật như Momentum để giảm dao động.

2.2.3 Momentum

Momentum là một kỹ thuật cải tiến trong tối ưu hóa, giúp giảm dao động trong quá trình tối ưu hóa và làm mượt quá trình cập nhật trọng số.

Công thức cập nhật trong Momentum

$$v_t = \beta v_{t-1} + (1 - \beta) \cdot \nabla L(w)$$
$$w_t = w_{t-1} - \eta v_t$$

Trong đó: - v_t là vận tốc (velocity) tại bước t, - β là tham số momentum (thường $\beta = 0.9$), - $\nabla L(w)$ là gradient của hàm mất mát đối với trọng số w, - η là learning rate.
Ưu điểm:

- Giảm thiểu dao động trong quá trình huấn luyện.
- Tăng tốc hội tụ, đặc biệt trong các thung lũng hẹp.

Nhược điểm:

- Cần phải điều chỉnh tham số momentum β cho hợp lý.
- Momentum có thể "vượt qua" điểm cực tiểu trong một số trường hợp nếu learning rate quá cao.

2.2.4 RMSProp

RMSProp (Root Mean Square Propagation) là một biến thể của SGD, trong đó tốc độ học được điều chỉnh cho mỗi tham số dựa trên bình phương của gradient.

Công thức cập nhật trong RMSProp

$$v_t = \beta v_{t-1} + (1 - \beta) \cdot (\nabla L(w))^2$$
$$w_t = w_{t-1} - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot \nabla L(w)$$

Trong đó: - v_t là trung bình bình phương của gradient, - ϵ là hằng số nhỏ để tránh chia cho 0 (thường $\epsilon = 10^{-8}$), - β là hệ số làm mượt (thường $\beta = 0.9$), - $\nabla L(w)$ là gradient của hàm mất mát đối với trọng số w, - η là learning rate.
Ưu điểm:

- Điều chỉnh learning rate cho từng tham số, giúp cải thiện tốc độ hội tụ.
- Hiệu quả trong các bài toán với dữ liệu thưa hoặc không đồng đều.

Nhược điểm:

- Cần chọn giá trị của β và ϵ sao cho phù hợp.

2.2.5 Adam (Adaptive Moment Estimation)

Adam là thuật toán tối ưu hóa kết hợp cả **Momentum** và **RMSProp**. Adam duy trì thông tin về vận tốc và sự thay đổi của gradient, giúp cải thiện tốc độ hội tụ và giảm thiểu dao động.

Công thức cập nhật trong Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot \nabla L(w)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \cdot (\nabla L(w))^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

Trong đó: - m_t là trung bình động của gradient (tương tự momentum), - v_t là trung bình động của bình phương gradient (tương tự RMSProp), - β_1 và β_2 là các tham số điều chỉnh cho momentum và RMSProp, - $\hat{m}_t$ và $\hat{v}_t$ là các giá trị hiệu chỉnh cho độ lệch (bias correction), - η là learning rate, và ϵ là hằng số rất nhỏ để tránh chia cho 0.
Ưu điểm:

- Kết hợp các ưu điểm của **Momentum** và **RMSProp**.
- Tự động điều chỉnh learning rate cho từng tham số, giúp tối ưu hóa hiệu quả hơn.
- Thường cho kết quả hội tụ nhanh và ổn định.

Nhược điểm:

- Thực tế cho thấy Adam có thể bị kẹt tại các điểm cực tiểu phụ (local minima).
- Cần phải điều chỉnh các siêu tham số như β_1, β_2, và η.

2.2.6 Lựa chọn Learning Rate và Các Kỹ Thuật Điều Chỉnh Learning Rate

Learning rate (tốc độ học) là siêu tham số quan trọng quyết định bước đi trong quá trình tối ưu hóa. Việc lựa chọn learning rate phù hợp ảnh hưởng lớn đến tốc độ và kết quả tối ưu hóa: - **Quá nhỏ**: Quá trình học rất chậm và cần nhiều vòng lặp để đạt được kết quả tốt. - **Quá lớn**: Thuật toán có thể vượt qua cực tiểu, làm giá trị hàm mất mát dao động hoặc phân kỳ. Các kỹ thuật điều chỉnh **learning rate** như **step decay** và **exponential decay** có thể giúp việc tối ưu hóa trở nên hiệu quả hơn, giúp giảm thiểu các vấn đề liên quan đến việc lựa chọn giá trị learning rate ban đầu.

2.2.7 Tóm tắt

Tối ưu hóa dựa trên gradient là một phương pháp cực kỳ quan trọng trong học sâu. Các biến thể của gradient descent như **SGD**, **Momentum**, **RMSProp**, và **Adam** đã giúp cải thiện đáng kể tốc độ huấn luyện và hiệu suất của các mô hình học sâu, với từng phương pháp phù hợp cho những bài toán khác nhau.

3 Các kiến trúc mạng học sâu nổi bật

3.1 LeNet-5: Kiến trúc mạng nơ-ron tích chập đầu tiên

3.1.1 Giới thiệu

LeNet-5 là một trong những mạng nơ-ron tích chập (CNN) đầu tiên, được phát triển bởi Yann LeCun vào những năm 1990. Mạng này được thiết kế đặc biệt cho bài toán nhận dạng chữ viết tay, như trong bộ dữ liệu MNIST, và đã tạo ra bước ngoặt trong học sâu, giúp khơi mào cho sự phát triển của các kiến trúc CNN sau này.

3.1.2 Kiến trúc LeNet-5

LeNet-5 bao gồm tổng cộng **7 lớp chính**, bao gồm các lớp tích chập, lớp lấy mẫu, và các lớp kết nối đầy đủ. Mỗi lớp trong LeNet-5 có một chức năng cụ thể giúp mạng có thể học và trích xuất đặc trưng từ ảnh đầu vào.

3.1.3 Các lớp trong LeNet-5:

1. **Lớp đầu vào:** LeNet-5 yêu cầu đầu vào là ảnh **32×32 pixels** với độ sâu màu xám (grayscale). Để phù hợp với bộ dữ liệu MNIST (28×28 pixels), ảnh sẽ được mở rộng thêm viền (padding).
2. **C1 – Lớp tích chập:** LeNet-5 sử dụng **6 bộ lọc kích thước 5×5** với bước stride là 1 và không dùng padding. Đầu ra của lớp C1 là một tensor có kích thước **28×28×6**, mỗi bộ lọc học được một đặc trưng riêng từ ảnh đầu vào, ví dụ như các cạnh và đường viền.
3. **S2 – Lớp lấy mẫu (Pooling):** S2 thực hiện **average pooling** với cửa sổ 2×2 và bước stride là 2, giúp giảm kích thước của đầu ra xuống **14×14×6**, làm giảm độ phân giải và tăng tính bất biến với các biến dạng nhỏ của ảnh.
4. **C3 – Lớp tích chập tiếp theo:** C3 có **16 bộ lọc 5×5**, với đầu ra kích thước **10×10×16**. Các kết nối giữa các bản đồ đặc trưng của S2 và C3 là không hoàn toàn kết nối, giúp giảm số lượng tham số và tăng hiệu quả tính toán.
5. **S4 – Lớp lấy mẫu:** Lại sử dụng **average pooling** với cửa sổ 2×2, bước stride là 2, cho đầu ra **5×5×16**.
6. **F5 – Lớp kết nối đầy đủ:** Lớp này gồm **120 nơ-ron**, thực hiện kết nối đầy đủ với đầu ra của S4. Mạng sẽ học các đặc trưng tổng hợp ở lớp này.
7. **F6 – Lớp kết nối đầy đủ thứ hai:** Lớp này có **84 nơ-ron**, tiếp tục kết hợp và trích chọn đặc trưng để chuẩn bị cho lớp đầu ra.
8. **Output – Lớp đầu ra:** Lớp đầu ra có **10 nơ-ron**, mỗi nơ-ron đại diện cho một chữ số (0–9). Hàm kích hoạt được sử dụng là **Radial Basis Function (RBF)** hoặc **Softmax** trong các phiên bản sau này của LeNet-5.

3.1.4 Sơ đồ kiến trúc LeNet-5

3.1.5 Ưu điểm:

- **Giảm số tham số:** Việc chia sẻ trọng số trong các lớp tích chập giúp giảm số lượng tham số cần huấn luyện, giúp mô hình ổn định và dễ dàng tổng quát hóa.

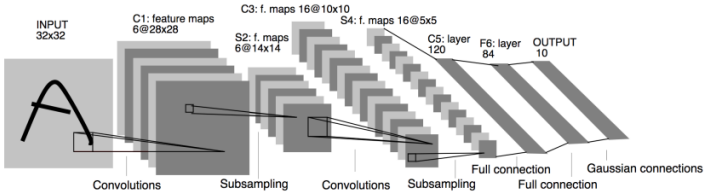

Figure 1: Sơ đồ kiến trúc LeNet-5. (Ảnh từ slide trang 49)

- **Bền vững với biến dạng:** Các lớp pooling giúp mạng nơ-ron bền vững hơn với các thay đổi nhỏ trong ảnh đầu vào.
- **Khả năng học phân cấp:** Các lớp thấp học các đặc trưng đơn giản như cạnh và đường thẳng, trong khi các lớp cao học các đặc trưng phức tạp hơn như các ký tự và hình dạng.

3.1.6 Hạn chế:

- **Hạn chế về kích thước ảnh đầu vào:** LeNet-5 yêu cầu ảnh đầu vào cố định kích thước 32×32, điều này gây khó khăn khi áp dụng mạng cho các ảnh có độ phân giải khác hoặc phức tạp hơn.
- **Mạng nơ-ron nông:** Với chỉ 2 lớp tích chập, LeNet-5 khó lòng học được các đặc trưng phức tạp từ dữ liệu lớn như các ảnh hiện đại.

3.1.7 Ứng dụng:

LeNet-5 được thiết kế dành cho **nhận dạng chữ viết tay**, nên ứng dụng nổi bật nhất của nó là trong việc nhận dạng chữ số và ký tự từ hình ảnh. Trong nghiên cứu học máy, LeNet-5 đã trở thành mô hình kinh điển cho bài toán nhận dạng chữ số MNIST, đạt độ chính xác cao với độ chính xác lên đến 99% trên tập kiểm thử.
LeNet-5 đã được triển khai trong thực tế vào những năm 90, đặc biệt là trong các hệ thống nhận dạng mã bưu điện và séc ngân hàng [**?**].

3.2 AlexNet

3.2.1 Giới thiệu

AlexNet là một mạng nơ-ron tích chập (CNN) được phát triển bởi Alex Krizhevsky, Ilya Sutskever và Geoffrey Hinton vào năm 2012. Mạng này đã giành chiến thắng trong cuộc thi **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**, vượt qua các phương pháp truyền thống với độ chính xác vượt trội, đồng thời làm nổi bật vai trò của học sâu trong lĩnh vực nhận dạng ảnh. AlexNet được coi là một bước đột phá trong lĩnh vực học máy và học sâu nhờ vào việc sử dụng các kiến trúc mạng nơ-ron sâu và mạnh mẽ, kết hợp với tài nguyên phần cứng GPU, giúp huấn luyện mạng nhanh chóng và hiệu quả hơn so với các phương pháp trước đây.
AlexNet có cấu trúc tương đối phức tạp so với LeNet-5 và các mạng trước đó, với 8 lớp học được (gồm 5 lớp convolutional và 3 lớp fully connected). Mạng này đã chứng minh hiệu quả vượt trội trong việc phân loại các ảnh phức tạp và đa dạng của bộ dữ liệu ImageNet.

3.2.2 Kiến trúc AlexNet

Kiến trúc của AlexNet bao gồm tổng cộng **8 lớp chính** (5 lớp convolutional và 3 lớp fully connected). Mỗi lớp có nhiệm vụ trích xuất các đặc trưng hình ảnh ở các mức độ khác nhau, từ các đặc trưng cơ bản như cạnh và hình dạng cho đến các đặc trưng trừu tượng hơn như các đối tượng.

- **Lớp đầu vào:** AlexNet yêu cầu ảnh đầu vào có kích thước **224×224×3** với mỗi pixel có giá trị RGB (3 kênh màu).
- **Các lớp convolutional:** AlexNet sử dụng tổng cộng **5 lớp convolutional** để trích xuất đặc trưng từ ảnh đầu vào. Các lớp này sử dụng bộ lọc với kích thước khác nhau để học các đặc trưng cấp thấp (cạnh, góc, hình dạng) đến các đặc trưng cấp cao hơn.
  - Lớp 1: 11×11 bộ lọc, bước stride 4, 96 bộ lọc. Đầu ra kích thước 55×55×96.
  - Lớp 2: 5×5 bộ lọc, bước stride 1, 256 bộ lọc. Đầu ra kích thước 27×27×256.
  - Lớp 3: 3×3 bộ lọc, bước stride 1, 384 bộ lọc. Đầu ra kích thước 13×13×384.
  - Lớp 4: 3×3 bộ lọc, bước stride 1, 384 bộ lọc. Đầu ra kích thước 13×13×384.
  - Lớp 5: 3×3 bộ lọc, bước stride 1, 256 bộ lọc. Đầu ra kích thước 13×13×256.
- **Các lớp fully connected:** Sau các lớp convolutional, AlexNet có **3 lớp fully connected** để kết hợp các đặc trưng trừu tượng được học từ các lớp trước và phân loại chúng thành các lớp mục tiêu.
  - Lớp 6: 4096 nơ-ron.
  - Lớp 7: 4096 nơ-ron.
  - Lớp 8: 1000 nơ-ron (số lớp tương ứng với số lớp trong bộ dữ liệu ImageNet).
- **Lớp đầu ra:** Lớp đầu ra sử dụng hàm kích hoạt **Softmax** để đưa ra xác suất cho 1000 lớp đầu ra.

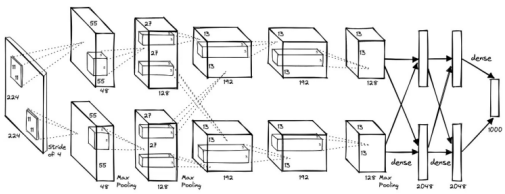

Figure 2: Sơ đồ kiến trúc AlexNet.

3.2.3 Ưu điểm:

- **Hiệu suất vượt trội:** AlexNet đã giành chiến thắng trong cuộc thi ILSVRC 2012 với độ chính xác vượt trội, giảm thiểu lỗi phân loại so với các phương pháp truyền thống như SVM.
- **Tăng tốc nhờ GPU:** Một trong những yếu tố quan trọng giúp AlexNet đạt hiệu suất cao là việc sử dụng **GPU** (Graphics Processing Unit) để huấn luyện mạng, giúp tăng tốc độ huấn luyện mạng nơ-ron rất nhiều so với việc sử dụng CPU.
- **Tính tổng quát tốt:** AlexNet đã thể hiện khả năng tổng quát hóa rất tốt khi áp dụng vào nhiều bài toán phân loại ảnh khác nhau, đặc biệt là trên bộ dữ liệu ImageNet.

3.2.4 Hạn chế:

- **Số lượng tham số lớn:** Mặc dù hiệu quả, AlexNet có rất nhiều tham số cần học (khoảng 60 triệu tham số), điều này khiến mạng rất tốn bộ nhớ và tài nguyên tính toán.
- **Khó khăn trong việc mở rộng:** Mặc dù AlexNet có thể học rất tốt với các bộ dữ liệu lớn, việc mở rộng mạng để xử lý các dữ liệu có độ phân giải cao hoặc yêu cầu tính toán phức tạp có thể gặp khó khăn vì số lượng tham số quá lớn.

AlexNet đã được sử dụng trong nhiều ứng dụng nhận dạng ảnh và phân loại ảnh. Mặc dù hiện nay có những kiến trúc tiên tiến hơn như ResNet và Inception, AlexNet vẫn là một trong những kiến trúc cơ bản và có ảnh hưởng sâu rộng trong lĩnh vực học sâu.

3.2.5 Ứng dụng thực tế:

AlexNet đã được áp dụng thành công trong các hệ thống nhận dạng đối tượng, phân loại ảnh y khoa, và phân tích các bộ dữ liệu ảnh đa dạng khác. Ngoài ra, AlexNet cũng được sử dụng rộng rãi trong các nghiên cứu và ứng dụng liên quan đến thị giác máy tính và nhận dạng mẫu.

3.3 VGG-16

3.3.1 Giới thiệu

VGG-16 là một kiến trúc mạng nơ-ron sâu nổi tiếng được giới thiệu bởi nhóm nghiên cứu của Visual Geometry Group tại Đại học Oxford trong cuộc thi ImageNet Large Scale Visual Recognition Challenge (ILSVRC) năm 2014. Kiến trúc này đã đạt được thành tích ấn tượng trong bài toán phân loại ảnh và được coi là một trong những bước tiến lớn trong học sâu, đặc biệt là trong việc xây dựng các mạng CNN sâu hơn.
VGG-16 sử dụng các lớp **convolutional** (tích chập) với các bộ lọc kích thước nhỏ (3×3), kết hợp với các lớp **fully connected** để học các đặc trưng từ ảnh đầu vào. Điều đặc biệt của VGG-16 là sự đơn giản trong thiết kế – tất cả các lớp convolutional đều sử dụng kích thước kernel 3×3 với bước stride là 1 và padding là 1, giúp giữ kích thước không gian của ảnh ổn định trong quá trình tính toán.

3.3.2 Kiến trúc VGG-16

Kiến trúc của VGG-16 bao gồm tổng cộng **16 lớp có tham số**, gồm các lớp convolutional và các lớp fully connected. Dưới đây là mô tả chi tiết về các lớp trong VGG-16:

- **Lớp đầu vào:** Ảnh đầu vào có kích thước **224×224×3**, với mỗi pixel có giá trị màu RGB (3 kênh màu).
- **Các lớp convolutional:** VGG-16 sử dụng các lớp **convolutional** với kernel 3×3, bước stride là 1, và padding là 1 để giữ nguyên kích thước của ảnh qua các lớp. Mạng có tổng cộng **13 lớp convolutional**, chia thành 5 khối (blocks) khác nhau. Cấu trúc chi tiết như sau:
  - Khối 1: 2 lớp convolutional, mỗi lớp có 64 bộ lọc 3×3.
  - Khối 2: 2 lớp convolutional, mỗi lớp có 128 bộ lọc 3×3.
  - Khối 3: 3 lớp convolutional, mỗi lớp có 256 bộ lọc 3×3.
  - Khối 4: 3 lớp convolutional, mỗi lớp có 512 bộ lọc 3×3.

– Khối 5: 3 lớp convolutional, mỗi lớp có 512 bộ lọc 3×3.

- **Các lớp fully connected:** Sau khi qua các lớp convolutional, đầu ra sẽ được phẳng hóa và đưa qua **3 lớp fully connected**. Lớp đầu tiên có 4096 nơ-ron, lớp thứ hai có 4096 nơ-ron, và lớp thứ ba có 1000 nơ-ron tương ứng với các lớp phân loại (với 1000 lớp trong ImageNet).
- **Lớp đầu ra:** Lớp đầu ra sử dụng hàm kích hoạt **Softmax** để đưa ra xác suất cho 1000 lớp đầu ra.

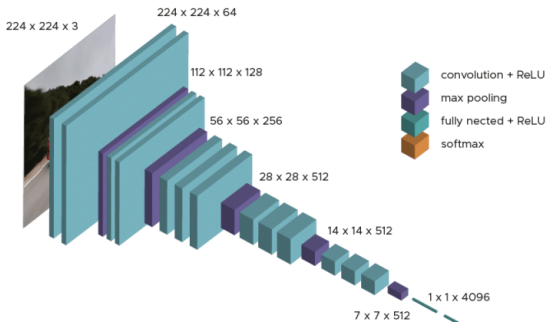

Figure 3: Sơ đồ kiến trúc VGG-16.

3.3.3 Ưu điểm:

- **Kiến trúc đơn giản và dễ hiểu:** VGG-16 sử dụng các lớp convolutional với kernel 3×3 trong suốt mạng, giúp mô hình dễ dàng hiểu và triển khai.
- **Hiệu suất cao trên bộ dữ liệu lớn:** VGG-16 đã đạt được kết quả xuất sắc trong bài toán phân loại ảnh của ImageNet, chứng tỏ khả năng học tốt các đặc trưng phức tạp từ ảnh.
- **Tính tổng quát tốt:** Mặc dù đơn giản, VGG-16 vẫn có khả năng tổng quát tốt trên các bài toán nhận diện ảnh không liên quan đến bộ dữ liệu huấn luyện.

3.3.4 Hạn chế:

- **Số lượng tham số lớn:** VGG-16 có số lượng tham số rất lớn, lên đến hơn 138 triệu tham số. Điều này làm cho quá trình huấn luyện trở nên tốn kém về tài nguyên tính toán và bộ nhớ.
- **Khó mở rộng:** Mặc dù VGG-16 đạt hiệu suất cao trên nhiều bài toán, nhưng việc mở rộng mạng nơ-ron này (chẳng hạn như với các ảnh có kích thước lớn hơn) có thể dẫn đến sự gia tăng đáng kể về số tham số.
- **Quá trình huấn luyện chậm:** Do số lượng tham số lớn, việc huấn luyện VGG-16 trên bộ dữ liệu lớn yêu cầu nhiều thời gian và tài nguyên phần cứng.

3.3.5 Ứng dụng

VGG-16 là một trong những kiến trúc CNN phổ biến nhất được sử dụng trong các bài toán nhận dạng ảnh, phân loại ảnh, và phát hiện đối tượng. Mặc dù hiện nay có các mạng CNN tiên tiến hơn như ResNet và Inception, VGG-16 vẫn là một trong những kiến trúc cơ bản được áp dụng rộng rãi trong học sâu.
VGG-16 đã được sử dụng trong nhiều hệ thống nhận dạng ảnh khác nhau, chẳng hạn như nhận dạng đối tượng trong video, phân loại ảnh y khoa, và trong các ứng dụng nhận diện biển số xe, các hệ thống giám sát an ninh.
—

3.4 GoogLeNet

3.4.1 Giới thiệu

GoogLeNet, được phát triển bởi Google, là một mạng nơ-ron tích chập sâu nổi tiếng, được giới thiệu trong bài báo *Going Deeper with Convolutions* tại cuộc thi **ILSVRC 2014**. GoogLeNet đã đạt được thành tích vượt trội trong việc phân loại hình ảnh, đặc biệt là khi so với các mạng nơ-ron tích chập (CNN) trước đó như AlexNet. GoogLeNet đã giảm thiểu đáng kể số lượng tham số mà vẫn duy trì được hiệu suất cao.
Một trong những điểm nổi bật của GoogLeNet là việc sử dụng **mô-đun Inception**, giúp mạng có thể xử lý các thông tin đầu vào ở nhiều độ phân giải khác nhau trong cùng một lớp, từ đó giảm số lượng tham số mà không làm giảm hiệu suất mạng. Điều này làm cho GoogLeNet rất hiệu quả về mặt tính toán.

3.4.2 Kiến trúc GoogLeNet

GoogLeNet sử dụng tổng cộng **22 lớp**, được xây dựng chủ yếu từ các mô-đun **Inception**. Mỗi mô-đun Inception giúp mạng có thể học các đặc trưng ở các độ phân giải khác nhau. Kiến trúc GoogLeNet bao gồm các lớp **convolutional**, **pooling**, và các lớp **fully connected**. Dưới đây là các chi tiết về kiến trúc của GoogLeNet:

- **Lớp đầu vào:** Ảnh đầu vào có kích thước **224×224×3** với mỗi pixel có giá trị RGB (3 kênh màu).
- **Lớp convolutional và pooling:** GoogLeNet sử dụng các lớp **convolutional** 7×7 với bước stride 2 và 64 bộ lọc đầu tiên. Sau đó, các lớp tiếp theo sẽ sử dụng các lớp **Inception module** để xử lý thông tin ở nhiều kích thước kernel khác nhau.
- **Mô-đun Inception:** Mỗi mô-đun Inception bao gồm các lớp convolutional với các kích thước kernel khác nhau (1×1, 3×3, 5×5) và một lớp pooling 3×3. Kết quả từ các lớp này được ghép lại (concatenated) thành một tensor duy nhất, giúp mạng có thể học các đặc trưng ở nhiều độ phân giải mà không tăng số lượng tham số.
- **Lớp fully connected:** GoogLeNet sử dụng một lớp fully connected duy nhất với 1000 nơ-ron để phân loại ảnh, tương ứng với 1000 lớp của bộ dữ liệu ImageNet.

3.4.3 Ưu điểm:

- **Hiệu suất cao với số lượng tham số thấp:** GoogLeNet đạt hiệu suất rất cao trong các bài toán phân loại ảnh nhưng có số lượng tham số thấp hơn nhiều so với các mạng trước đó như AlexNet hay VGG-16.

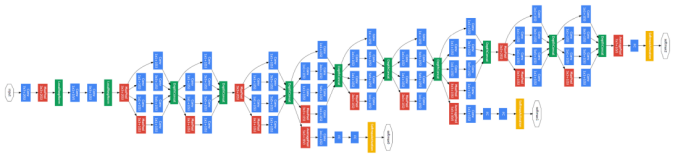

Figure 4: Sơ đồ kiến trúc GoogLeNet.

- **Sử dụng mô-đun Inception hiệu quả:** Các mô-đun Inception giúp giảm thiểu số lượng tham số, nhưng vẫn giữ được khả năng học các đặc trưng ở nhiều mức độ khác nhau.
- **Tính tổng quát và hiệu quả trong huấn luyện:** GoogLeNet có thể huấn luyện tốt với bộ dữ liệu lớn mà không cần phải sử dụng quá nhiều tài nguyên tính toán.

3.4.4 Hạn chế:

- **Độ phức tạp trong triển khai:** Cấu trúc mô-đun Inception có thể gây khó khăn trong việc triển khai và điều chỉnh cho các bài toán mới.
- **Thời gian huấn luyện lâu:** Mặc dù có số lượng tham số ít hơn, nhưng GoogLeNet vẫn yêu cầu nhiều thời gian huấn luyện và tài nguyên tính toán, đặc biệt là khi áp dụng vào các bộ dữ liệu lớn.

3.4.5 Ứng dụng

GoogLeNet đã được ứng dụng rộng rãi trong các bài toán nhận dạng hình ảnh, phân loại ảnh và nhận dạng đối tượng. Các mô-đun Inception được áp dụng không chỉ trong phân loại ảnh mà còn trong các bài toán nhận dạng video, nhận dạng đối tượng trong ảnh và phân tích dữ liệu y tế.
GoogLeNet được sử dụng trong nhiều hệ thống nhận dạng đối tượng và phân loại ảnh. Nó cũng đã được áp dụng trong nhận dạng video và các ứng dụng giám sát an ninh, nơi việc nhận dạng các đối tượng trong cảnh quay video là rất quan trọng.

3.5 Inception-v3

3.5.1 Giới thiệu

Inception-v3 là phiên bản cải tiến của kiến trúc GoogLeNet, được giới thiệu bởi Google trong năm 2015. Kiến trúc này nhằm giải quyết một số vấn đề còn tồn tại trong GoogLeNet và cải thiện hiệu suất cũng như khả năng huấn luyện. Inception-v3 tiếp tục sử dụng mô-đun Inception nhưng với các cải tiến trong thiết kế, giúp mạng đạt được hiệu suất cao hơn trong bài toán phân loại ảnh.
Một trong những cải tiến chính trong Inception-v3 là việc sử dụng **factorization** (phân tách) để giảm số lượng tham số trong các lớp convolutional, đồng thời giữ lại tính linh hoạt và khả năng học các đặc trưng đa dạng từ dữ liệu. Inception-v3 cũng sử dụng các kỹ thuật cải tiến như **Batch Normalization** và **RMSprop** để tăng tốc độ huấn luyện và cải thiện độ chính xác.

3.5.2 Kiến trúc Inception-v3

Inception-v3 có cấu trúc mạng gồm tổng cộng **48 lớp**. Kiến trúc này bao gồm các lớp **convolutional**, **pooling**, và **fully connected**. Các lớp convolutional trong Inception-

v3 được thiết kế với các kích thước kernel khác nhau, và các mô-đun Inception giúp xử lý các đặc trưng ở các độ phân giải khác nhau. Dưới đây là một số điểm nổi bật trong kiến trúc của Inception-v3:

- **Lớp đầu vào:** Ảnh đầu vào có kích thước **299×299×3**, lớn hơn so với GoogLeNet (224×224), giúp mạng có thể học các đặc trưng chi tiết hơn từ ảnh.
- **Các lớp convolutional và pooling:** Inception-v3 sử dụng các lớp convolutional với các kích thước kernel khác nhau (1×1, 3×3, 5×5) trong các mô-đun Inception để học các đặc trưng từ ảnh đầu vào. Ngoài ra, mạng sử dụng các lớp **average pooling** để giảm kích thước đầu ra và giúp cải thiện khả năng tổng quát của mô hình.
- **Mô-đun Inception:** Các mô-đun Inception trong Inception-v3 đã được cải tiến so với GoogLeNet, sử dụng phân tách các lớp convolution lớn thành các lớp convolution nhỏ hơn để giảm số lượng tham số và tăng hiệu quả tính toán. Các lớp này cũng sử dụng **1×1 convolution** để giảm chiều sâu của các dữ liệu trước khi áp dụng các lớp convolution lớn.
- **Batch Normalization:** Inception-v3 sử dụng kỹ thuật **Batch Normalization** để chuẩn hóa các đầu ra của các lớp convolutional, giúp làm giảm hiện tượng **vanishing gradients** và tăng tốc quá trình huấn luyện.
- **Lớp fully connected:** Cuối cùng, các đặc trưng được trích xuất từ các lớp convolutional và pooling được đưa vào các lớp fully connected để phân loại ảnh thành các lớp mục tiêu. Lớp đầu ra có 1000 nơ-ron, tương ứng với các lớp trong bộ dữ liệu ImageNet.

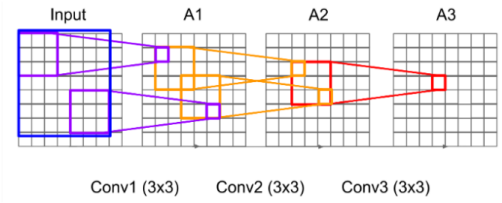

Figure 5: Sơ đồ kiến trúc Inception-v3

3.5.3 Ưu điểm:

- **Hiệu suất cao:** Inception-v3 đạt hiệu suất vượt trội trong bài toán phân loại ảnh trên bộ dữ liệu ImageNet, với độ chính xác cao và số lượng tham số thấp hơn so với các mạng sâu khác.
- **Giảm số lượng tham số:** Mạng sử dụng kỹ thuật phân tách và mô-đun Inception để giảm số lượng tham số cần huấn luyện mà vẫn duy trì hiệu quả tính toán cao.
- **Khả năng tổng quát tốt:** Inception-v3 có khả năng tổng quát hóa tốt hơn khi xử lý các ảnh có độ phân giải khác nhau và các biến dạng trong dữ liệu.

3.5.4 Hạn chế:

- **Phức tạp trong thiết kế:** Cấu trúc mô-đun Inception và các kỹ thuật phân tách làm cho việc triển khai Inception-v3 trở nên phức tạp hơn so với các mạng đơn giản hơn như AlexNet.
- **Yêu cầu tài nguyên tính toán lớn:** Dù Inception-v3 có số lượng tham số thấp hơn nhiều so với các mạng khác, nhưng nó vẫn yêu cầu tài nguyên tính toán lớn và thời gian huấn luyện lâu, đặc biệt khi áp dụng cho các tập dữ liệu rất lớn.

3.5.5 Ứng dụng

Inception-v3 được sử dụng trong nhiều ứng dụng phân loại ảnh, nhận dạng đối tượng, và các bài toán liên quan đến thị giác máy tính. Các cải tiến trong kiến trúc giúp nó có thể xử lý hiệu quả hơn các hình ảnh có độ phân giải cao và các dữ liệu phức tạp.
Inception-v3 đã được áp dụng trong nhiều hệ thống nhận dạng đối tượng và phân loại ảnh, bao gồm các ứng dụng trong giám sát an ninh, phân tích y khoa, và nhận dạng biển số xe. Nó cũng được sử dụng trong các bài toán nhận dạng video và phân tích dữ liệu hình ảnh trong các ứng dụng tự động hóa.

3.6 ResNet

3.6.1 Giới thiệu

ResNet (Residual Networks) là một kiến trúc mạng nơ-ron sâu được giới thiệu bởi Kaiming He và các cộng sự vào năm 2015, trong bài báo *Deep Residual Learning for Image Recognition*. ResNet đã giải quyết được một vấn đề quan trọng trong học sâu, đó là **vanishing gradient** (biến mất gradient) khi huấn luyện các mạng nơ-ron rất sâu. ResNet sử dụng các **Residual Blocks** với các kết nối bỏ qua (skip connections) để giúp thông tin có thể lưu thông dễ dàng hơn trong suốt quá trình huấn luyện, từ đó làm cho việc huấn luyện các mạng rất sâu trở nên khả thi.
Một điểm nổi bật của ResNet là khả năng huấn luyện mạng với hàng trăm hoặc thậm chí hàng nghìn lớp mà không gặp phải vấn đề gradient biến mất. Điều này đã giúp ResNet trở thành một trong những kiến trúc mạng nơ-ron phổ biến nhất trong các bài toán nhận dạng hình ảnh và các ứng dụng học sâu khác.

3.6.2 Kiến trúc ResNet

ResNet có cấu trúc gồm các **Residual Blocks**, mỗi block bao gồm một hoặc nhiều lớp convolutional, với các kết nối bỏ qua giữa các lớp. Các kết nối bỏ qua này giúp giảm thiểu vấn đề gradient biến mất và cho phép mạng học các đặc trưng sâu mà không bị suy giảm theo chiều sâu.
Kiến trúc ResNet có thể được điều chỉnh với số lượng lớp khác nhau, từ **ResNet-18** (18 lớp) đến **ResNet-152** (152 lớp), với mỗi phiên bản có số lượng lớp sâu khác nhau. Dưới đây là mô tả về các thành phần chính trong kiến trúc ResNet:

- **Lớp đầu vào:** Ảnh đầu vào có kích thước **224×224×3** với mỗi pixel có giá trị RGB (3 kênh màu).
- **Các lớp convolutional:** ResNet bắt đầu với một lớp convolutional với bộ lọc 7×7, bước stride 2, và 64 bộ lọc. Sau đó, mạng sẽ sử dụng các lớp **Residual Blocks** để học các đặc trưng.

- **Residual Blocks:** Mỗi Residual Block gồm các lớp convolutional (thường là 3×3 hoặc 1×1) và một kết nối bỏ qua (skip connection). Kết nối bỏ qua cho phép thông tin đi qua các lớp mà không bị thay đổi, giúp duy trì thông tin trong quá trình huấn luyện mạng sâu.
- **Lớp fully connected:** Cuối cùng, các đặc trưng học được từ các Residual Blocks được đưa vào một lớp fully connected để phân loại ảnh thành các lớp mục tiêu.
- **Lớp đầu ra:** Lớp đầu ra có 1000 nơ-ron (cho bộ dữ liệu ImageNet), sử dụng hàm kích hoạt **Softmax** để đưa ra xác suất cho mỗi lớp.

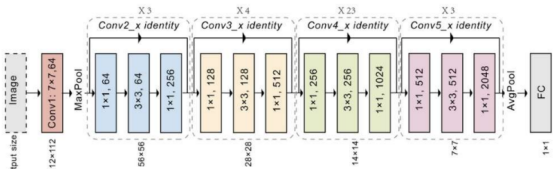

Figure 6: Sơ đồ kiến trúc ResNet

3.6.3 Ưu điểm:

- **Giải quyết vấn đề vanishing gradient:** ResNet sử dụng các kết nối bỏ qua để giảm thiểu vấn đề **vanishing gradient** khi huấn luyện mạng sâu, giúp huấn luyện các mạng với hàng trăm hoặc hàng nghìn lớp một cách hiệu quả.
- **Tính linh hoạt cao:** ResNet có thể mở rộng linh hoạt từ **ResNet-18** đến **ResNet-152**, giúp phù hợp với nhiều bài toán và dữ liệu khác nhau.
- **Hiệu suất cao trên bộ dữ liệu lớn:** ResNet đã đạt được kết quả ấn tượng trong các bài toán phân loại ảnh lớn như ImageNet và các bài toán nhận dạng đối tượng.

3.6.4 Hạn chế:

- **Sự phức tạp trong thiết kế:** Mặc dù có nhiều cải tiến, nhưng việc thiết kế và triển khai ResNet vẫn yêu cầu sự hiểu biết sâu sắc về các kết nối residual và cách chúng ảnh hưởng đến việc huấn luyện mạng.
- **Thời gian huấn luyện lâu:** Mặc dù số lượng tham số được giảm thiểu, nhưng ResNet vẫn yêu cầu thời gian huấn luyện lâu và tài nguyên tính toán lớn khi huấn luyện các mạng rất sâu.

3.6.5 Ứng dụng

ResNet đã được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau, từ nhận dạng hình ảnh, phân loại đối tượng cho đến các ứng dụng trong y học, tự động hóa và phân tích video.
ResNet được sử dụng trong nhiều hệ thống nhận dạng đối tượng, phân loại ảnh y khoa, nhận dạng video, và trong các ứng dụng tự động hóa, đặc biệt là trong các ngành công nghiệp và giám sát an ninh.

3.7 Các kỹ thuật cải tiến trong học sâu:Batch Normalization

3.7.1 Vấn đề Internal Covariate Shift

Trong quá trình huấn luyện mạng nơ-ron, các giá trị đầu ra của mỗi lớp có thể thay đổi theo thời gian khi các trọng số và độ lệch (bias) được cập nhật, dẫn đến một hiện tượng gọi là **internal covariate shift**. Điều này làm giảm tốc độ huấn luyện vì mạng phải liên tục học lại cách phân phối đầu ra tại mỗi lớp trong quá trình cập nhật trọng số.

3.7.2 Giải pháp: Batch Normalization

Batch Normalization được giới thiệu để giải quyết vấn đề internal covariate shift. Kỹ thuật này chuẩn hóa các đầu ra của mỗi lớp trong mạng với mỗi mini-batch, giúp mạng ổn định và tăng tốc quá trình huấn luyện.
Quy trình chuẩn hóa trong Batch Normalization bao gồm ba bước chính: 1. **Tính toán giá trị trung bình và phương sai trong mini-batch:**

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

Trong đó x_i là các giá trị của đầu ra lớp trong mini-batch có kích thước m.
2. **Chuẩn hóa các giá trị đầu ra:**

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Với ϵ là một hằng số nhỏ để tránh chia cho 0, đảm bảo tính ổn định số học.
3. **Biến đổi scale và shift với các tham số học được γ và β:**

$$y_i = \gamma \hat{x}_i + \beta$$

Trong đó, γ và β là các tham số có thể học, giúp điều chỉnh lại dữ liệu sau khi chuẩn hóa.

3.7.3 Lợi ích của Batch Normalization

- **Giảm Internal Covariate Shift:** Quá trình chuẩn hóa giúp giảm bớt sự thay đổi trong phân phối đầu ra của mỗi lớp, làm quá trình huấn luyện ổn định hơn.
- **Tăng tốc quá trình huấn luyện:** Vì các lớp được chuẩn hóa, mạng không còn phải làm lại việc học phân phối đầu ra từ đầu, từ đó tăng tốc quá trình huấn luyện.
- **Cho phép sử dụng Learning Rate cao hơn:** Vì các đầu ra đã được chuẩn hóa, quá trình tối ưu hóa có thể sử dụng learning rate lớn hơn mà không gặp phải các vấn đề liên quan đến gradient vanishing hoặc exploding.
- **Hiệu quả Regularization:** Batch Normalization có tác dụng tương tự như regularization, giúp giảm thiểu hiện tượng overfitting trong huấn luyện.

3.7.4 Ứng dụng trong quá trình inference

Khi mạng đã được huấn luyện xong và chuyển sang giai đoạn inference (dự đoán), thay vì sử dụng các thống kê của mini-batch, ta sử dụng các giá trị trung bình và phương sai ước tính trong quá trình huấn luyện. Các giá trị này được tính bằng cách sử dụng moving averages, giúp việc chuẩn hóa ổn định khi áp dụng mô hình vào dữ liệu mới.

3.7.5 Vị trí trong mạng nơ-ron

Batch Normalization thường được sử dụng sau các lớp **convolutional** hoặc **fully connected**, nhưng trước khi áp dụng các hàm kích hoạt như ReLU, giúp ổn định quá trình học và cải thiện hiệu quả tối ưu hóa.

3.7.6 Ảnh hưởng và ứng dụng

Batch Normalization đã trở thành một phần không thể thiếu trong các mạng học sâu hiện đại và được áp dụng rộng rãi trong nhiều kiến trúc mạng như **AlexNet**, **VGG-16**, **ResNet**, và các mô hình học sâu khác. Nó giúp mạng huấn luyện nhanh hơn và ổn định hơn, đồng thời giảm thiểu hiện tượng overfitting.

3.8 Thuật toán học sâu (Deep Learning)

Học sâu (Deep Learning) là một nhánh con của học máy (Machine Learning), tập trung vào việc học các đặc trưng và biểu diễn của dữ liệu thông qua các mạng nơ-ron nhân tạo sâu. Các mô hình học sâu có thể học từ dữ liệu thô mà không cần sự can thiệp của con người, giúp cải thiện hiệu suất trong nhiều bài toán phức tạp.

3.8.1 Mạng nơ-ron tích chập (CNN)

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) là một loại mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu dạng hình ảnh và video. Các mạng CNN có thể học các đặc trưng của hình ảnh qua các lớp convolutional và pooling. Kiến trúc này đã giúp mạng học sâu đạt được kết quả xuất sắc trong các bài toán nhận diện hình ảnh và phân loại đối tượng.

3.8.1.1 Kiến trúc của CNN

Mạng CNN bao gồm các thành phần cơ bản sau:

- **Lớp Convolutional (Convolutional Layer):** Lớp này áp dụng các bộ lọc (filters) để phát hiện các đặc trưng trong hình ảnh, như cạnh, góc, hoặc các mẫu hình học cơ bản. Các bộ lọc có thể học được từ dữ liệu trong quá trình huấn luyện.
- **Lớp Pooling (Pooling Layer):** Lớp này giúp giảm chiều dữ liệu và giảm độ phức tạp tính toán, đồng thời giữ lại các đặc trưng quan trọng. Thường sử dụng kỹ thuật max pooling, giúp giảm kích thước của hình ảnh mà không làm mất thông tin quan trọng.
- **Lớp Fully Connected (Fully Connected Layer):** Lớp này kết nối tất cả các nơ-ron của lớp trước đó với tất cả các nơ-ron trong lớp hiện tại. Các lớp fully connected giúp thực hiện phân loại hoặc hồi quy dựa trên các đặc trưng đã học được.
- **Hàm kích hoạt (Activation Function):** Các hàm như ReLU, sigmoid hoặc softmax thường được sử dụng trong CNN để giới thiệu phi tuyến tính vào mô hình.

3.8.1.2 Quy trình hoạt động của CNN

Mạng CNN hoạt động qua nhiều bước:

- **Convolution Operation:** Dữ liệu đầu vào (hình ảnh) được đưa qua các bộ lọc trong lớp convolutional. Mỗi bộ lọc học các đặc trưng khác nhau trong hình ảnh. Sau khi áp dụng bộ lọc, mạng sẽ thu được các bản đồ đặc trưng (feature maps).
- **Activation Function:** Các bản đồ đặc trưng sẽ được đưa qua một hàm kích hoạt, thường là hàm ReLU, để tạo ra các giá trị phi tuyến tính.

- **Pooling:** Sau mỗi lớp convolution, các bản đồ đặc trưng sẽ được giảm kích thước thông qua pooling. Max pooling là một kỹ thuật phổ biến, giúp giảm chiều của bản đồ đặc trưng mà không làm mất quá nhiều thông tin.
- **Fully Connected Layers:** Cuối cùng, các lớp fully connected sử dụng các đặc trưng học được từ các lớp trước đó để đưa ra kết quả phân loại hoặc hồi quy.

3.8.1.3 Ứng dụng của CNN CNN đã được ứng dụng rộng rãi trong nhiều lĩnh vực như nhận diện hình ảnh, phân loại đối tượng, nhận dạng chữ viết tay, và phát hiện các đối tượng trong video. CNN là nền tảng của các hệ thống nhận diện hình ảnh hiện đại, như các mô hình phân loại ảnh trong bộ dữ liệu ImageNet.

3.8.1.4 Các kiến trúc CNN nổi bật

- **LeNet-5:** Là mạng CNN đầu tiên được phát triển bởi Yann LeCun để nhận diện chữ viết tay.
- **AlexNet:** Được phát triển bởi Alex Krizhevsky, giành chiến thắng trong cuộc thi ImageNet năm 2012, sử dụng GPU để tăng tốc tính toán.
- **VGG-16:** Được phát triển bởi Visual Geometry Group, sử dụng các lớp convolutional với kích thước bộ lọc cố định 3x3.
- **ResNet:** Sử dụng các residual blocks để giải quyết vấn đề gradient biến mất trong mạng nơ-ron sâu.

3.8.2 Mạng nơ-ron hồi tiếp (RNN)

Mạng nơ-ron hồi tiếp (Recurrent Neural Networks - RNN) là một loại mạng nơ-ron đặc biệt dùng để xử lý dữ liệu tuần tự, như chuỗi thời gian hoặc văn bản. Điểm đặc biệt của RNN là chúng có khả năng ghi nhớ thông tin từ các bước trước và sử dụng chúng để dự đoán các giá trị trong tương lai.

3.8.2.1 Kiến trúc của RNN RNN có một đặc điểm quan trọng: mỗi nơ-ron trong mạng nhận đầu vào không chỉ từ dữ liệu hiện tại mà còn từ trạng thái ẩn (hidden state) của các bước thời gian trước đó. Điều này giúp mạng có khả năng học các phụ thuộc dài hạn trong dữ liệu tuần tự.
Phương trình của một RNN cơ bản có thể được mô tả như sau:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = \text{softmax}(W_{hy}h_t + b_y)$$

Trong đó: - h_t là trạng thái ẩn tại thời điểm t, - x_t là đầu vào tại thời điểm t, - y_t là đầu ra tại thời điểm t.

3.8.2.2 Các vấn đề với RNN Mặc dù RNN rất mạnh trong việc xử lý dữ liệu tuần tự, nhưng chúng gặp phải vấn đề **vanishing gradients** khi huấn luyện các mạng dài, tức là gradient trở nên quá nhỏ khi truyền ngược qua nhiều bước thời gian. Điều này khiến việc huấn luyện mạng trở nên khó khăn. Để khắc phục, các biến thể của RNN như **LSTM (Long Short-Term Memory)** và **GRU (Gated Recurrent Units)** được phát triển.

3.8.2.3 Ứng dụng của RNN RNN được sử dụng trong nhiều bài toán như xử lý ngôn ngữ tự nhiên (NLP), dịch máy, phân tích chuỗi thời gian, và nhận diện giọng nói. Các ứng dụng phổ biến bao gồm dịch ngữ nghĩa, dự báo thị trường tài chính, và phân loại văn bản.

3.8.2.4 LSTM và GRU LSTM và GRU là hai loại RNN có khả năng học tốt hơn các phụ thuộc dài hạn trong chuỗi thời gian, bằng cách sử dụng các cơ chế cửa (gates) để điều khiển việc lưu trữ và quên thông tin trong trạng thái ẩn.

3.8.3 Transformer

Transformer là kiến trúc mạng nơ-ron mới nổi trong lĩnh vực học sâu, được giới thiệu trong bài báo *Attention is All You Need* (2017). Đây là kiến trúc đã thay thế RNN và LSTM trong nhiều bài toán xử lý ngôn ngữ tự nhiên (NLP), vì khả năng tính toán song song mạnh mẽ và hiệu quả xử lý các phụ thuộc dài hạn trong chuỗi dữ liệu.

3.8.3.1 Cơ chế Attention Một trong những yếu tố chính của Transformer là cơ chế **Self-Attention**. Cơ chế này cho phép mỗi từ trong chuỗi học được mối quan hệ với tất cả các từ khác trong chuỗi mà không cần phải duy trì trạng thái tuần tự như trong RNN.
Phương trình cơ bản của **Scaled Dot-Product Attention** là:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Trong đó: - Q, K, và V lần lượt là ma trận truy vấn (Query), khóa (Key) và giá trị (Value), - d_k là kích thước của các khóa.

3.8.3.2 Kiến trúc của Transformer Transformer bao gồm hai thành phần chính: **Encoder** và **Decoder**. Mỗi thành phần này đều sử dụng nhiều lớp **Self-Attention** và **Feedforward Neural Networks** để xử lý dữ liệu. Encoder mã hóa thông tin đầu vào, còn Decoder tạo ra đầu ra từ thông tin mã hóa.

3.8.3.3 Ứng dụng của Transformer Transformer đã đạt được thành công lớn trong nhiều bài toán NLP, đặc biệt là các mô hình như **BERT** và **GPT**. Các ứng dụng bao gồm dịch máy, phân tích văn bản, sinh văn bản tự động, và trả lời câu hỏi.

3.8.3.4 BERT và GPT - **BERT (Bidirectional Encoder Representations from Transformers):** Là một mô hình học sâu tiền huấn luyện, học được đại diện ngữ nghĩa của văn bản trong cả hai hướng trái phải. - **GPT (Generative Pretrained Transformer):** Là một mô hình tạo sinh văn bản, có khả năng tạo ra văn bản tự nhiên từ một đoạn văn bản đầu vào.

4 Các Thuật Toán Liên Quan

Phần này trình bày chi tiết các thuật toán và kỹ thuật quan trọng được sử dụng trong mô hình để phân loại hình ảnh. Mỗi thuật toán sẽ được giới thiệu về mặt lý thuyết, công thức toán học và cách ứng dụng cụ thể trong mô hình

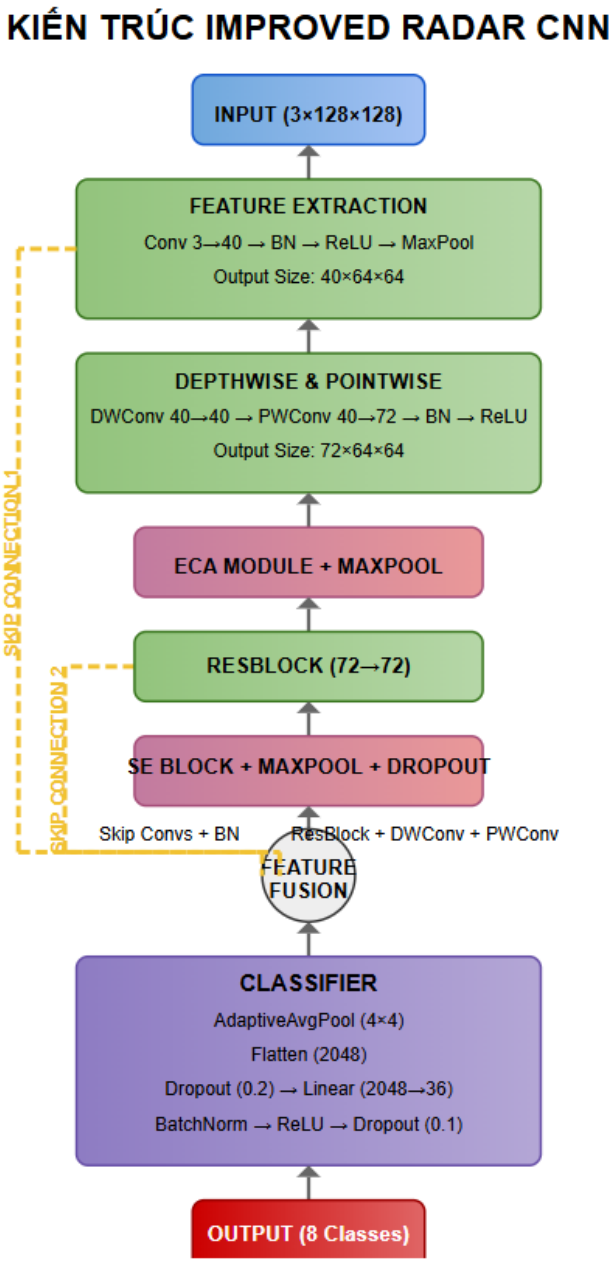

Figure 7: Tổng quan kiến trúc mô hình

4.1 ECAModule (Efficient Channel Attention)

Mục tiêu ECAModule (Efficient Channel Attention) là một mô-đun chú ý kênh giúp mạng học các đặc trưng quan trọng từ dữ liệu một cách hiệu quả. Trong các mạng học sâu, việc học và chú ý đến các đặc trưng quan trọng từ dữ liệu đầu vào là một nhiệm vụ rất quan trọng để cải thiện hiệu suất của mô hình, đặc biệt trong các mạng nơ-ron sâu hoặc các kiến trúc phức tạp.

4.1.1 Giải thích chi tiết

Chú ý kênh (channel attention) là một trong các phương pháp phổ biến để cải thiện khả năng của mô hình trong việc tự động nhận diện các đặc trưng quan trọng từ mỗi kênh của đầu vào. ECAModule cải tiến phương pháp chú ý kênh bằng cách sử dụng lớp Conv1D thay vì các phương pháp phức tạp như Squeeze-and-Excitation (SE), giúp giảm độ phức tạp tính toán mà vẫn giữ được hiệu quả chú ý.

4.1.1.1 Công thức tính trọng số kênh

Giả sử đầu vào có kích thước $X \in \mathbb{R}^{B \times C \times H \times W}$, trong đó:

- B là kích thước batch,
- C là số lượng kênh,
- H và W là chiều cao và chiều rộng của ảnh đầu vào.

ECAModule sử dụng phép trung bình (average pooling) để giảm chiều cao và chiều rộng của ảnh đầu vào, sau đó áp dụng một lớp Conv1D để tính toán trọng số cho từng kênh.
Công thức tính trọng số kênh $\mathbf{w}_c$ là:

$$\mathbf{w}_c = \sigma(\mathbf{W} \cdot \text{avgpool}(X)),$$

Trong đó:

- $\mathbf{W}$ là trọng số của lớp Conv1D,
- σ là hàm kích hoạt sigmoid,
- avgpool(X) là phép pooling trung bình trên các chiều không phải kênh.

Kết quả của phép toán này là một vector trọng số cho mỗi kênh, giúp mô hình học được mức độ quan trọng của từng kênh.

4.1.1.2 Đầu ra

Đầu ra của ECAModule là sản phẩm của đầu vào X với trọng số $\mathbf{w}_c$:

$$X' = X \cdot \mathbf{w}_c.$$

Điều này có nghĩa là mỗi kênh của đầu vào được nhân với trọng số chú ý, giúp khuếch đại các đặc trưng quan trọng và làm giảm các đặc trưng không quan trọng.

4.1.2 Lợi ích

- Giảm độ phức tạp tính toán so với các phương pháp chú ý truyền thống.
- Cải thiện hiệu quả của mô hình trong việc học các đặc trưng quan trọng từ dữ liệu.

4.2 SEBlock (Squeeze-and-Excitation Block)

4.2.1 Mục tiêu

SEBlock là một mô-đun chú ý không gian (spatial attention) nhằm tăng cường khả năng học các trọng số cho các kênh đầu vào của mô hình. Được giới thiệu lần đầu tiên trong bài báo "Squeeze-and-Excitation Networks" (Hu et al., 2018), SEBlock đã trở thành một trong những mô-đun chú ý nổi bật trong các mạng học sâu.

4.2.2 Giải thích chi tiết

SEBlock làm việc bằng cách áp dụng một quá trình gọi là "Squeeze" để giảm kích thước thông tin theo chiều không gian, sau đó sử dụng một quá trình "Excitation" để học các trọng số cho từng kênh.

4.2.2.1 Squeeze (Chèn) Trong bước này, SEBlock sử dụng phép toán AdaptiveAvgPool2d(1) để giảm chiều không gian của đầu vào X, tạo ra một vector đặc trưng $\mathbf{z}_c$ có kích thước C, trong đó C là số lượng kênh. Điều này giúp tóm tắt thông tin của mỗi kênh và giảm thiểu độ phức tạp.
Công thức cho phép toán này là:

$$\mathbf{z}_c = \text{GlobalAvgPool}(X),$$

Trong đó $\mathbf{z}_c$ chứa thông tin tổng quát của mỗi kênh.

4.2.2.2 Excitation (Kích thích) Sau khi thực hiện phép "Squeeze", SEBlock tiếp tục thực hiện một quá trình "Excitation" để học các trọng số cho mỗi kênh. Đây là một quá trình hai lớp fully connected (FC), trong đó:

- Lớp đầu tiên sử dụng hàm kích hoạt ReLU để học các đặc trưng phi tuyến của kênh.
- Lớp thứ hai sử dụng hàm kích hoạt sigmoid để chuẩn hóa giá trị trọng số trong khoảng từ 0 đến 1.

Công thức cho trọng số kích thích $\mathbf{s}_c$ là:

$$\mathbf{s}_c = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot \mathbf{z}_c)),$$

Trong đó W_1 và W_2 là các trọng số của các lớp fully connected.

4.2.2.3 Đầu ra Kết quả cuối cùng của SEBlock là sản phẩm của đầu vào X và vector trọng số $\mathbf{s}_c$:

$$X' = X \cdot \mathbf{s}_c.$$

Điều này giúp khuếch đại các đặc trưng quan trọng của các kênh và giảm thiểu sự ảnh hưởng của các kênh không quan trọng.

4.2.3 Lợi ích

- Tăng cường khả năng chú ý không gian, giúp mô hình học các trọng số kênh một cách chính xác hơn.
- Giảm thiểu độ phức tạp tính toán nhờ việc sử dụng pooling và các lớp fully connected nhỏ.

4.3 ResBlock (Residual Block)

4.3.1 Mục tiêu

ResBlock là một phần quan trọng trong kiến trúc ResNet (Residual Networks), giúp giải quyết vấn đề gradient vanishing trong các mạng nơ-ron sâu.

4.3.2 Giải thích chi tiết

ResBlock là một kỹ thuật giúp truyền thông tin qua các lớp trong mạng nơ-ron mà không bị mất đi sự quan trọng của các đặc trưng. Điều này được thực hiện thông qua các "shortcut connections" (kết nối bỏ qua), cho phép các đặc trưng từ các lớp trước được truyền thẳng đến các lớp sau mà không bị thay đổi.

4.3.2.1 Cấu trúc Mỗi ResBlock bao gồm hai lớp convolutional (Conv2d), với các phép biến đổi không gian 3×3. Các lớp này giúp mô hình học được các đặc trưng không gian quan trọng của dữ liệu.
Công thức của ResBlock là:

$$X' = F(X, \{W_i\}) + X,$$

Trong đó:

- $F(X, \{W_i\})$ là kết quả của các lớp convolutional,
- X là đầu vào gốc,
- X' là đầu ra của ResBlock.

4.3.2.2 Đầu ra Đầu ra của ResBlock là tổng của các đặc trưng được học qua các lớp convolutional và đầu vào gốc, giúp thông tin quan trọng được truyền qua mạng mà không bị mất mát.

4.3.3 Lợi ích

- Giúp giảm thiểu vấn đề gradient vanishing trong các mạng nơ-ron sâu.
- Tăng cường khả năng học của mạng nơ-ron sâu, cho phép các mô hình học hiệu quả hơn.

4.4 Skip Connections

4.4.1 Mục tiêu

Skip connections là một kỹ thuật giúp cải thiện quá trình huấn luyện các mạng nơ-ron sâu bằng cách cho phép thông tin đi thẳng từ các lớp trước đến các lớp sau mà không bị thay đổi.

4.4.2 Giải thích chi tiết

Skip connections (hay còn gọi là shortcut connections) cho phép các đặc trưng quan trọng từ các lớp trước được truyền trực tiếp tới các lớp sau mà không cần qua các phép toán phức tạp. Điều này giúp giảm thiểu vấn đề gradient vanishing và cải thiện khả năng học của mô hình.

4.4.2.1 Công thức Giả sử X là đầu vào của một mạng và $F(X)$ là đầu ra của mạng, công thức của skip connection là:

$$Y = F(X) + X.$$

Điều này cho phép thông tin từ các lớp trước được truyền qua các lớp sau một cách hiệu quả hơn.

4.4.3 Lợi ích

- Cải thiện việc truyền tải gradient qua các lớp, giúp giảm thiểu vấn đề gradient vanishing.
- Tăng cường khả năng học của mô hình, đặc biệt trong các mạng nơ-ron sâu.

4.5 RAdam (Rectified Adam)

4.5.1 Mục tiêu

RAdam là một phiên bản cải tiến của thuật toán Adam, giúp cải thiện độ ổn định trong quá trình huấn luyện, đặc biệt trong các bài toán có dữ liệu lớn và phức tạp.

4.5.2 Giải thích chi tiết

RAdam cải tiến phương pháp Adam bằng cách điều chỉnh learning rate tự động, giúp quá trình huấn luyện ổn định hơn trong các bước đầu của quá trình huấn luyện.

4.5.2.1 Công thức cập nhật trọng số Phương pháp Adam cập nhật trọng số theo công thức:

$$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon},$$

Trong đó:

- m_t là moment đầu tiên,
- v_t là moment thứ hai,
- η là learning rate,
- ϵ là một hằng số nhỏ để tránh chia cho 0.

4.5.2.2 Điều chỉnh learning rate RAdam điều chỉnh learning rate trong quá trình huấn luyện bằng cách sử dụng một hệ số rectification r_t để làm cho quá trình huấn luyện ổn định hơn.

4.5.3 Lợi ích

- Cải thiện độ ổn định của quá trình huấn luyện.
- Giúp mô hình hội tụ nhanh hơn và ổn định hơn.

4.6 OneCycleLR (One Cycle Learning Rate Scheduling)

4.6.1 Mục tiêu

OneCycleLR là một chiến lược điều chỉnh learning rate trong quá trình huấn luyện, nhằm giúp mô hình hội tụ nhanh hơn và đạt kết quả tối ưu.

4.6.2 Giải thích chi tiết

OneCycleLR giúp điều chỉnh learning rate trong quá trình huấn luyện theo một chu kỳ, bắt đầu bằng việc tăng tỷ lệ học từ giá trị nhỏ đến giá trị tối đa, rồi giảm dần xuống giá trị ban đầu.

4.6.2.1 Công thức Trong giai đoạn đầu, OneCycleLR tăng learning rate từ một giá trị nhỏ đến giá trị tối đa $\eta_{\max}$:

$$\eta_t = \eta_{\min} + (\eta_{\max} - \eta_{\min}) \cdot \text{LR\_schedule}(t),$$

Sau đó, learning rate sẽ giảm xuống trong các giai đoạn tiếp theo.

4.6.3 Lợi ích

- Giúp mô hình hội tụ nhanh hơn.
- Cải thiện hiệu suất huấn luyện và giảm thời gian huấn luyện.

5 Kiến trúc ImprovedRadarCNN

5.1 Tổng quan

ImprovedRadarCNN là một mô hình CNN tiên tiến được thiết kế đặc biệt cho việc phân loại tín hiệu radar với hiệu suất cao nhưng vẫn giữ số lượng tham số thấp (297,723). Mô hình kết hợp nhiều kỹ thuật hiện đại trong deep learning như:

- **Depthwise Separable Convolution** để giảm số lượng tham số
- **Attention Mechanisms** (ECA và SE) để tập trung vào đặc trưng quan trọng
- **Skip Connections** để giải quyết vấn đề vanishing gradient
- **Residual Blocks** để hỗ trợ học sâu
- **Regularization** thông qua các lớp Dropout đặt ở vị trí chiến lược

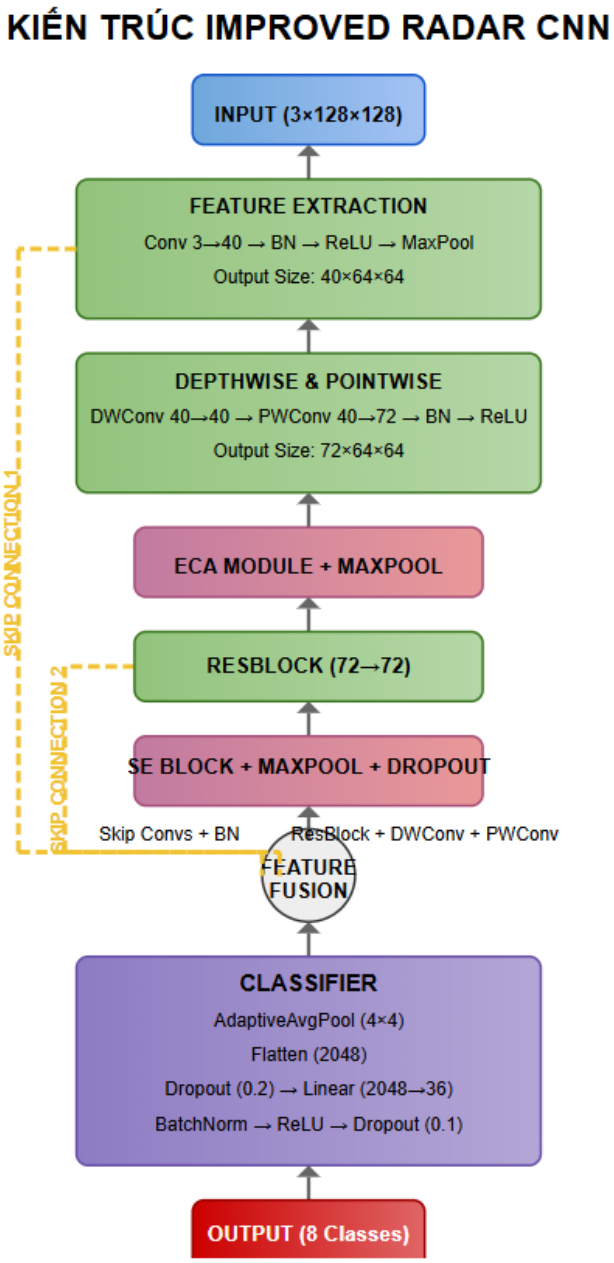

Figure 8: Tổng quan kiến trúc mô hình

5.2 Các thành phần chính

5.2.1 Feature Extraction

- **Đầu vào**: Ảnh 3 kênh kích thước 128×128
- **Convolutional layer đầu tiên**: $3 \rightarrow 40$ kênh

- **Xử lý**: BatchNorm, ReLU, MaxPool
- **Đầu ra**: Feature map $40 \times 64 \times 64$

5.2.2 Efficient Feature Learning

- **Depthwise Convolution**: $40 \rightarrow 40$ (giữ nguyên số kênh)
- **Pointwise Convolution**: $40 \rightarrow 72$ (tăng số kênh)
- **ECA Module**: Module attention đầu tiên
  - Đặc điểm: Sử dụng 1D convolution với kernel size thích ứng
  - Mục đích: Tái hiệu chuẩn trọng số kênh với chi phí tính toán thấp

5.2.3 Feature Enhancement

- **ResBlock**: Giữ nguyên số kênh ($72 \rightarrow 72$)
  - Cải thiện quá trình học bằng kết nối tắt
- **Depthwise & Pointwise Convolution**: Tiếp tục xử lý đặc trưng
- **SE Block**: Module attention thứ hai
  - Đặc điểm: Sử dụng hai fully connected layer (squeeze và excitation)
  - Mục đích: Học mối quan hệ giữa các kênh đặc trưng

5.2.4 Advanced Feature Processing

- **ResBlock thứ hai**: Tăng khả năng biểu diễn
- **Convolution cuối**: $72 \rightarrow 128$ kênh
- **Dropout**: Giảm overfitting (tỷ lệ 0.05)

5.2.5 Skip Connections

- **Skip Connection 1**: Từ output của MaxPool đầu tiên
  - Chuyển đổi: $40 \rightarrow 128$ kênh với stride=8
- **Skip Connection 2**: Từ output của ResBlock đầu tiên
  - Chuyển đổi: $72 \rightarrow 128$ kênh với stride=4
- **Feature Fusion**: Kết hợp đặc trưng từ đường đi chính và hai skip connection

5.2.6 Classifier

- **Pooling**: AdaptiveAvgPool xuống kích thước 4×4
- **Flatten**: Chuyển thành vector 2048 chiều
- **Fully Connected Layers**:
  - FC1: $2048 \rightarrow 36$ với BatchNorm và ReLU
  - FC2: $36 \rightarrow 8$ (output layer)
- **Regularization**: Dropout với tỷ lệ 0.2 và 0.1

5.3 Kỹ thuật đào tạo

Mô hình được đào tạo với các kỹ thuật hiện đại:

5.3.1 Data Augmentation

- RandomHorizontalFlip, RandomRotation, ColorJitter
- Mixup và CutMix với $\alpha = 0.1$

5.3.2 Optimization

- AdamW optimizer với learning rate 0.0004
- Weight decay $1e-4$
- CosineAnnealingWarmRestarts scheduler

5.3.3 Regularization

- Label smoothing với $\epsilon = 0.05$
- Gradient clipping với max_norm=0.5
- Nhiều lớp Dropout với tỷ lệ khác nhau

5.3.4 Training Strategy

- Early stopping với patience=25
- Mixed precision training
- Stratified data splitting

5.4 Ưu điểm của kiến trúc

1. **Hiệu quả về tham số**: Số lượng tham số thấp (297,723) nhưng vẫn đạt hiệu suất cao
2. **Khả năng chống overfitting**: Nhiều kỹ thuật regularization được áp dụng
3. **Khả năng trích xuất đặc trưng**: Kết hợp skip connections với attention mechanisms
4. **Hiệu quả tính toán**: Sử dụng depthwise separable convolution để giảm chi phí
5. **Khả năng tổng quát hóa**: Thiết kế để hoạt động tốt trên các loại tín hiệu radar khác nhau

Table 1: Thông số chi tiết của các layer trong ImprovedRadarCNN

Layer	**Type**	**Output Shape**	**Tham số**
Input	-	$3 \times 128 \times 128$	0
Conv1	Conv2D	$40 \times 128 \times 128$	1,080
BatchNorm1	BatchNorm2D	$40 \times 128 \times 128$	80
MaxPool1	MaxPool2D	$40 \times 64 \times 64$	0
DWConv1	DepthwiseConv2D	$40 \times 64 \times 64$	360
BatchNorm2	BatchNorm2D	$40 \times 64 \times 64$	80
PWConv1	PointwiseConv2D	$72 \times 64 \times 64$	2,880
BatchNorm3	BatchNorm2D	$72 \times 64 \times 64$	144
ECA	ECAModule	$72 \times 64 \times 64$	≈ 50
MaxPool2	MaxPool2D	$72 \times 32 \times 32$	0
ResBlock1	ResBlock	$72 \times 32 \times 32$	94,248
DWConv2	DepthwiseConv2D	$72 \times 32 \times 32$	648
BatchNorm4	BatchNorm2D	$72 \times 32 \times 32$	144
PWConv2	PointwiseConv2D	$72 \times 32 \times 32$	5,184
BatchNorm5	BatchNorm2D	$72 \times 32 \times 32$	144
SE	SEBlock	$72 \times 32 \times 32$	$\approx 10{,}368$
MaxPool3	MaxPool2D	$72 \times 16 \times 16$	0
Dropout1	Dropout2D	$72 \times 16 \times 16$	0
ResBlock2	ResBlock	$72 \times 16 \times 16$	94,248
DWConv3	DepthwiseConv2D	$72 \times 16 \times 16$	648
BatchNorm6	BatchNorm2D	$72 \times 16 \times 16$	144
PWConv3	PointwiseConv2D	$128 \times 16 \times 16$	9,216
BatchNorm7	BatchNorm2D	$128 \times 16 \times 16$	256
MaxPool4	MaxPool2D	$128 \times 8 \times 8$	0
Dropout2	Dropout2D	$128 \times 8 \times 8$	0
AdaptiveAvgPool	AdaptiveAvgPool2D	$128 \times 4 \times 4$	0
Flatten	Flatten	2048	0
Dropout3	Dropout	2048	0
FC1	Linear	36	73,764
BatchNorm8	BatchNorm1D	36	72
Dropout4	Dropout	36	0
FC2	Linear	8	296
		Tổng số tham số:	**297,723**

6 Lý do sử dụng kiến trúc mạng CNN này

Kiến trúc mạng CNN này được thiết kế dựa trên các kỹ thuật tiên tiến giúp tối ưu hóa hiệu suất và khả năng tổng quát. Dưới đây là lý do chi tiết vì sao các thành phần trong kiến trúc này được chọn.

6.1 Depthwise Separable Convolutions (Giảm tham số và tính toán)

Depthwise Separable Convolution là một biến thể của lớp convolution truyền thống, giúp giảm đáng kể số lượng tham số và tính toán. Thay vì thực hiện một phép toán convolution bình thường (mỗi bộ lọc áp dụng cho tất cả các kênh), depthwise separable convolution chia thành hai bước:

- **Depthwise Convolution**: Mỗi bộ lọc chỉ áp dụng cho một kênh riêng biệt (không phải toàn bộ các kênh).
- **Pointwise Convolution**: Dùng một bộ lọc 1×1 để kết hợp các kênh sau bước depthwise.

Lý do sử dụng:

- **Tiết kiệm tài nguyên tính toán**: Việc giảm số lượng tham số có thể giúp giảm thời gian huấn luyện và bộ nhớ cần thiết, đồng thời giữ được hiệu suất mô hình.
- **Giảm overfitting**: Mô hình nhẹ hơn, ít tham số hơn, vì vậy dễ dàng hơn trong việc ngăn ngừa overfitting, nhất là khi dữ liệu huấn luyện không quá phong phú.

6.2 Residual Connections (Kết nối dư)

Residual Connection là một kỹ thuật trong đó đầu ra của một lớp được cộng trực tiếp vào đầu vào của lớp sau, thay vì truyền qua tất cả các phép toán như bình thường.
Lý do sử dụng:

- **Giảm vấn đề vanishing gradient**: Residual connections giúp tín hiệu gradient không bị suy giảm quá nhiều trong quá trình lan truyền ngược (backpropagation), giúp huấn luyện các mô hình sâu hơn mà không gặp phải vấn đề vanishing gradient.
- **Cải thiện hiệu suất huấn luyện**: Với kết nối dư, mô hình có thể học các đặc trưng phức tạp hơn mà không làm mất đi thông tin từ các lớp trước đó, dẫn đến việc hội tụ nhanh hơn trong quá trình huấn luyện.
- **Tăng khả năng học đặc trưng**: Việc cho phép mô hình học các "đặc trưng dư thừa" giúp mạng học được các đặc trưng phức tạp hơn và cải thiện khả năng phân loại.

6.3 Attention Mechanisms (Cơ chế chú ý)

Attention Mechanism cho phép mô hình tập trung vào các vùng quan trọng trong dữ liệu đầu vào. Trong mạng CNN, cơ chế chú ý thường được sử dụng để xác định các khu vực đặc biệt có ảnh hưởng mạnh mẽ đến quyết định phân loại.
Lý do sử dụng:

- **Chú ý đến các đặc trưng quan trọng**: Mạng có thể tự động học được các phần quan trọng của hình ảnh và tập trung vào chúng, giúp mô hình phân loại chính xác hơn.

- **Tăng tính trực quan và hiệu suất**: Attention mechanism giúp mạng CNN "hiểu" đâu là các vùng có thông tin quan trọng nhất trong hình ảnh, từ đó cải thiện độ chính xác của phân loại và giảm thiểu lỗi phân loại cho các đối tượng không quan trọng.
- **Linh hoạt**: Cơ chế chú ý không phụ thuộc vào kiến trúc đặc biệt nào và có thể được áp dụng cho nhiều loại mô hình khác nhau, làm cho mô hình linh hoạt và có thể thích ứng với nhiều loại dữ liệu và bài toán khác nhau.

6.4 Batch Normalization (Chuẩn hóa theo batch)

Batch Normalization giúp chuẩn hóa các giá trị của đầu vào trong mỗi lớp, làm giảm sự phụ thuộc vào việc lựa chọn các siêu tham số (hyperparameters) như learning rate. Điều này giúp tăng tốc độ hội tụ của mô hình và cải thiện độ chính xác cuối cùng.
Lý do sử dụng:

- **Cải thiện sự hội tụ**: Việc chuẩn hóa đầu vào giúp tránh các vấn đề như gradient vanishing/exploding, khiến mô hình có thể học nhanh hơn và ổn định hơn.
- **Giảm overfitting**: Batch normalization cũng giúp giảm overfitting bằng cách cung cấp một chút nhiễu trong quá trình huấn luyện, điều này giúp mô hình học được các đặc trưng chung thay vì học quá chi tiết (overfitting).

6.5 Global Average Pooling (Giảm kích thước đặc trưng)

Global Average Pooling là một kỹ thuật để giảm số lượng tham số trong mô hình bằng cách tính giá trị trung bình của mỗi kênh trong toàn bộ không gian đầu vào, thay vì áp dụng một lớp pooling thông thường.
Lý do sử dụng:

- **Giảm số lượng tham số**: Global average pooling giúp giảm độ phân giải không gian của đầu vào xuống một chiều, từ đó giảm số lượng tham số trong mô hình và làm mô hình trở nên nhẹ hơn.
- **Cải thiện khả năng tổng quát**: Bằng cách tính trung bình toàn bộ không gian, mô hình sẽ học được các đặc trưng tổng quát của dữ liệu, thay vì bị phụ thuộc vào các chi tiết nhỏ trong hình ảnh, giúp tăng khả năng tổng quát và giảm overfitting.

6.6 Fully Connected (FC) Layers và Dropout

Fully Connected Layers giúp mạng học các mối quan hệ phức tạp giữa các đặc trưng đã trích xuất qua các lớp convolution và pooling.
Dropout được áp dụng để giảm thiểu overfitting trong quá trình huấn luyện.
Lý do sử dụng:

- **Kết nối mạnh mẽ giữa các đặc trưng**: Các lớp fully connected giúp mô hình kết hợp các đặc trưng đã học từ các lớp trước để đưa ra quyết định phân loại cuối cùng.
- **Giảm overfitting**: Dropout giúp giảm nguy cơ overfitting trong quá trình huấn luyện bằng cách ngắt kết nối ngẫu nhiên giữa các đơn vị, điều này giúp mô hình không quá phụ thuộc vào các đặc trưng riêng biệt.

6.7 Tổng hợp Lý do

Mô hình này được thiết kế để tối ưu hóa các yếu tố sau:

- **Hiệu suất**: Các kỹ thuật như depthwise separable convolution và batch normalization giúp tăng tốc độ huấn luyện và giảm thời gian tính toán.
- **Chất lượng phân loại**: Sự kết hợp của residual connections và attention mechanisms giúp cải thiện khả năng học và phân loại chính xác, đặc biệt trong các bài toán phức tạp.
- **Khả năng tổng quát**: Các kỹ thuật như dropout, global average pooling và batch normalization giúp giảm overfitting và cải thiện khả năng tổng quát của mô hình đối với dữ liệu chưa thấy.

Tóm lại, kiến trúc mạng CNN này là một sự kết hợp tối ưu giữa các kỹ thuật hiện đại giúp giảm thiểu chi phí tính toán, đồng thời cải thiện khả năng phân loại và tính tổng quát của mô hình, đặc biệt là trong các bài toán nhận diện hình ảnh hoặc radar.

7 Kết Qủa Thử Nghiệm

7.1 Thiết lập thực nghiệm

Mô hình được huấn luyện và đánh giá trên tập dữ liệu phân loại tín hiệu radar với 8 lớp:

- **Tập dữ liệu:** 5120 mẫu huấn luyện và 1280 mẫu kiểm tra (tỷ lệ 80:20)
- **Các lớp:** 'B-FM', 'Barker', 'CPFSK', 'DSB-AM', 'GFSK', 'LFM', 'Rect', 'SSB-AM'
- **Kích thước đầu vào:** 128×128×3
- **Batch size:** 32
- **Optimizer:** RAdam với learning rate ban đầu 0.001 và weight decay 5e-5
- **Learning rate scheduler:** CosineAnnealingWarmRestarts
- **Epochs:** Tối đa 100 (dừng tại epoch 69)
- **Loss function:** Cross Entropy Loss
- **Thiết bị:** NVIDIA CUDA GPU

Thông số	Giá trị
Tổng số tham số	297.769
Độ chính xác trên tập validation	88,85%
Số epochs thực hiện	69(dừng sớm)

Table 2: Hiệu suất tổng thể của mô hình

7.2 Quá trình huấn luyện

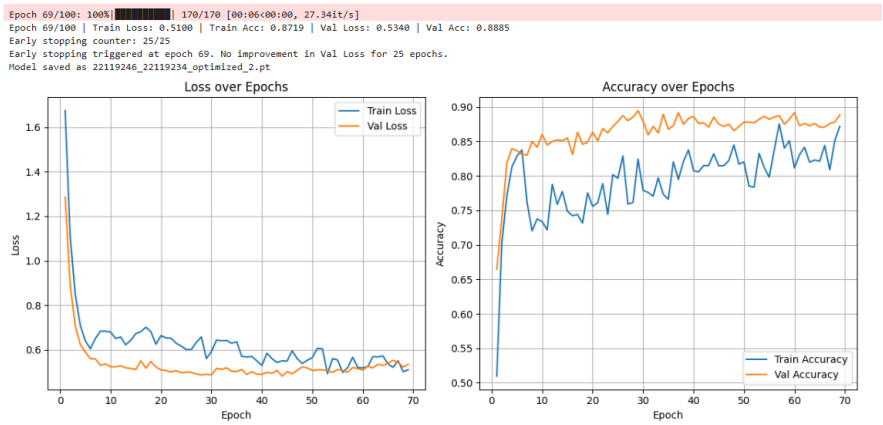

Figure 9: Đường cong loss và accuracy trong quá trình huấn luyện

Trong quá trình huấn luyện, mô hình đạt được kết quả tốt với độ chính xác trên tập huấn luyện là 0.8719 và trên tập validation là 0.8885. Đáng chú ý, độ chính xác trên tập validation cao hơn so với tập huấn luyện, cho thấy mô hình không bị overfitting và có khả năng tổng quát hóa tốt.

Đường biểu diễn Train Loss và Validation Loss đều có xu hướng giảm ổn định trong suốt quá trình huấn luyện, và không có dấu hiệu tăng trở lại, thể hiện mô hình học ổn định và không bị hiện tượng divergence. Validation loss duy trì thấp hơn train loss trong phần lớn thời gian, phù hợp với kỳ vọng khi áp dụng các kỹ thuật như Label Smoothing hoặc EMA.

Về độ chính xác (Accuracy), đường biểu diễn Validation Accuracy luôn nằm trên đường Train Accuracy, với mức độ dao động nhẹ ở train accuracy, gợi ý mô hình có thể còn underfitting nhẹ. Tuy nhiên, overall mô hình đã đạt được sự cân bằng rất tốt giữa độ chính xác và khả năng tổng quát hóa.

Việc sử dụng Early Stopping sau 69 epoch cũng cho thấy mô hình được dừng lại đúng thời điểm, tránh việc tiếp tục huấn luyện gây overfitting hoặc lãng phí tài nguyên Bảng 3 thể hiện một số mốc quan trọng trong quá trình huấn luyện:

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.66543	0.5050	1.2492	0.6836
5	0.6170	0.8372	0.4745	0.8538
10	0.5750	0.7883	0.4582	0.8674
45	0.5180	0.8179	0.5251	0.8800
69	0.5100	0.8719	0.5340	0.8885

Table 3: Giá trị loss và accuracy tại các epochs đại diện

7.3 Ma trận nhầm lẫn

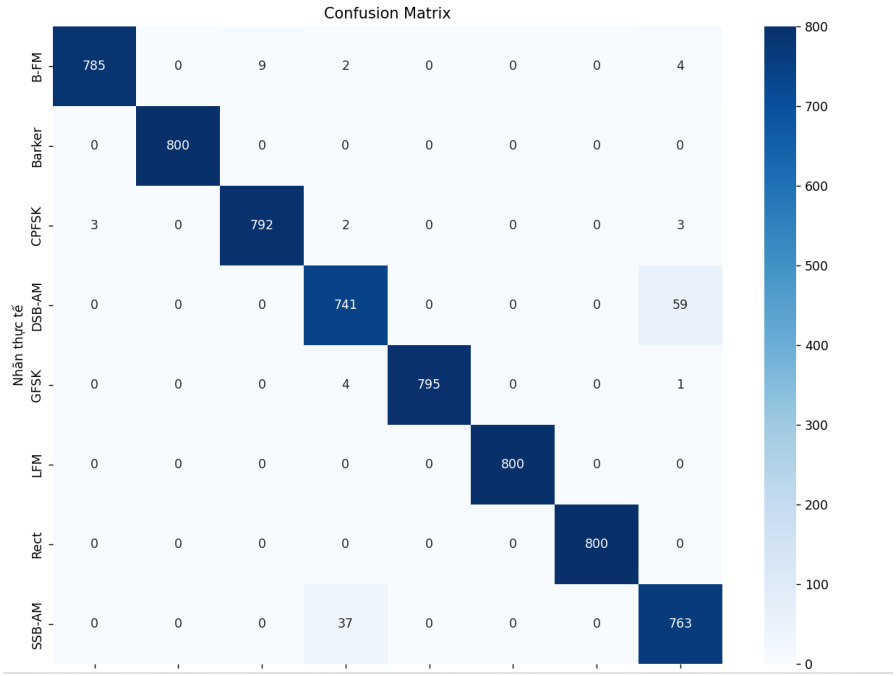

Figure 10: Ma trận nhầm lẫn của mô hình trên tập validation

- Lớp 'Barker', 'LFM' và 'Rect' có độ chính xác gần như hoàn hảo (100%)
- Lớp 'DSB-AM' và 'SSB-AM' có sự nhầm lẫn đáng kể với nhau, với 59 mẫu 'DSB-AM' bị nhầm là 'SSB-AM' và 37 mẫu 'SSB-AM' bị nhầm là 'DSB-AM'
- Có một số nhầm lẫn nhỏ giữa các lớp khác như 'B-FM' và 'CPFSK', 'GFSK' và 'SSB-AM'

```
Chỉ số hiệu suất theo class:
B-FM: Precision=0.9962, Recall=0.9812, F1-score=0.9887
Barker: Precision=1.0000, Recall=1.0000, F1-score=1.0000
CPFSK: Precision=0.9888, Recall=0.9900, F1-score=0.9894
DSB-AM: Precision=0.9427, Recall=0.9263, F1-score=0.9344
GFSK: Precision=1.0000, Recall=0.9938, F1-score=0.9969
LFM: Precision=1.0000, Recall=1.0000, F1-score=1.0000
Rect: Precision=1.0000, Recall=1.0000, F1-score=1.0000
SSB-AM: Precision=0.9193, Recall=0.9537, F1-score=0.9362
```

Figure 11: Kết quả test trên tập validation

Kết quả chi tiết theo class
B-FM: Precision: 0.9962, Recall: 0.9812, F1-score: 0.9887 Hiệu suất rất cao, mô hình phân loại tốt với sai số rất thấp.

Barker: Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000 Hiệu suất hoàn hảo, không có lỗi.
CPFSK: Precision: 0.9888, Recall: 0.9900, F1-score: 0.9894 Hiệu suất rất cao, gần hoàn hảo.
DSB-AM: Precision: 0.9427, Recall: 0.9263, F1-score: 0.9344 Hiệu suất thấp hơn các class khác, cần cải thiện.
GFSK: Precision: 1.0000, Recall: 0.9938, F1-score: 0.9969 Gần hoàn hảo, sai số rất nhỏ ở Recall.
LFM: Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000 Hiệu suất hoàn hảo, không có lỗi.
Rect: Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000 Hiệu suất hoàn hảo, không có lỗi.
SSB-AM: Precision: 0.9193, Recall: 0.9537, F1-score: 0.9362 Hiệu suất thấp nhất, đặc biệt ở Precision, cần cải thiện khả năng phân biệt..

8 Phân tích và Đánh giá

8.1 Phân Tích

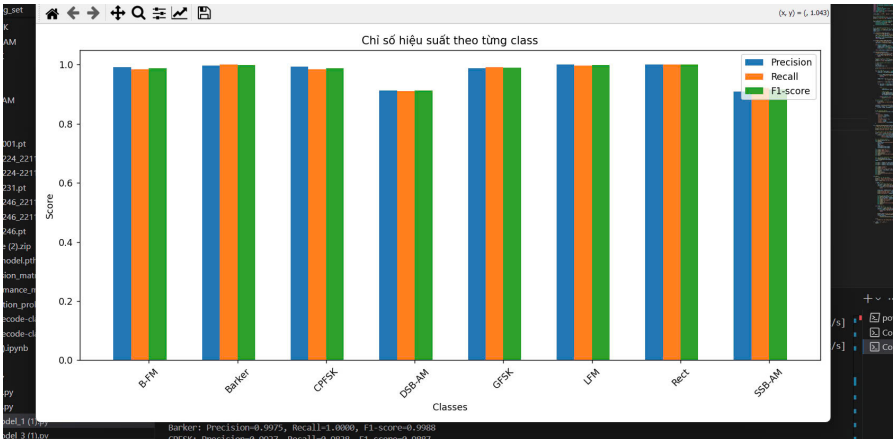

Figure 12: Bảng chỉ số hiệu suất theo từng class

Mô hình đạt hiệu suất xuất sắc trên các class như Barker, LFM, và Rect với F1-score đạt 1.0000. Tuy nhiên, các class DSB-AM và SSB-AM có hiệu suất thấp hơn, đặc biệt ở Precision và F1-score, cho thấy mô hình gặp khó khăn trong việc phân biệt các class này với các class khác.
Độ chính xác tổng thể cao (0.9806) nhưng cần kiểm tra xem tập dữ liệu có bị mất cân bằng hay không, vì điều này có thể ảnh hưởng đến hiệu suất trên các class thiểu số.
Xu hướng quá trình huấn luyện

Figure 13: Quá trình huấn luyện Loss và Accuracy

Độ lỗi (loss) giảm nhanh trong khoảng 15-20 epoch đầu tiên, sau đó giảm chậm hơn Độ chính xác (accuracy) trên tập kiểm định có xu hướng đạt mức ổn định sau khoảng 30 epoch Khoảng cách giữa train accuracy và validation accuracy khá nhỏ, cho thấy mô hình không bị overfitting nghiêm trọng
Việc kết hợp các kỹ thuật hiện đại như ECA, SE Block, cùng với các phương pháp tăng cường dữ liệu và chiến lược huấn luyện tiên tiến đã giúp mô hình đạt hiệu suất cao trong nhiệm vụ phân loại tín hiệu radar, mặc dù số lượng tham số khá khiêm tốn.

8.2 Phân tích điểm mạnh

Hiệu quả tham số

- **Tính nhỏ gọn**: Mô hình đạt được tỷ lệ nén cao với chỉ 297,723 tham số, phù hợp cho triển khai trên các thiết bị với tài nguyên hạn chế.
- **Tích chập khả tách theo chiều sâu**: Giảm đáng kể số lượng phép tính (FLOPs) bằng cách tách tích chập tiêu chuẩn thành tích chập theo chiều sâu và tích chập điểm:

$$\text{Standard Conv FLOPs} = H \times W \times C_{in} \times C_{out} \times K^2 \tag{1}$$

$$\text{Depthwise Sep Conv FLOPs} = H \times W \times C_{in} \times K^2 + H \times W \times C_{in} \times C_{out} \tag{2}$$

  với thực tế $K^2 \ll C_{out}$ trong mạng hiện đại, đạt được tỷ lệ nén $\approx K^2/C_{out}$ lần.

- **Phân bổ tham số có chọn lọc**: Tập trung tham số vào các lớp quan trọng nhất, với số kênh tăng dần ($40 \to 72 \to 128$) để cân bằng giữa khả năng biểu diễn và hiệu quả:

```
self.features = nn.Sequential(
    nn.Conv2d(3, 40, kernel_size=3, stride=1, padding=1, bias=False),
    # ... more layers ...
    nn.Conv2d(72, 128, kernel_size=1, bias=False),
)
```

- **Tái sử dụng đặc trưng**: Kết nối tắt cho phép lan truyền thông tin không qua biến đổi phi tuyến, giảm nhu cầu về tham số để học lại các đặc trưng đã trích xuất.

Độ hội tụ

- **Động lực hội tụ**: Sử dụng AdamW với hiệu chỉnh biên độ (moment correction) và cập nhật theo tỷ lệ bình phương trung bình:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4}$$

  cho phép vượt qua các điểm yên ngựa (saddle points) và đạt hội tụ nhanh hơn SGD thông thường.

- **Khắc phục vấn đề mất mát gradient**: Các kết nối tắt và cấu trúc ResBlock cho phép dòng gradient chảy trực tiếp qua mạng, giảm thiểu vấn đề tiêu biến (vanishing) gradient:

```
def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.bn2(self.conv2(out))
    out += self.shortcut(x)  # Gradient flow bypass
    out = F.relu(out)
    return out
```

- **Lập lịch tốc độ học**: CosineAnnealingWarmRestarts tối ưu quá trình rời khỏi các cực tiểu cục bộ và hội tụ về cực tiểu toàn cục:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right) \tag{5}$$

  với chu kỳ $T_0 = 15$ và hệ số nhân $T_{mult} = 1$.

- **Độ ổn định tăng cường**: Cắt xén gradient (max_grad_norm = 0.5) ngăn chặn cập nhật quá lớn, ổn định quá trình huấn luyện và giảm dao động dư:

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
```

Khả năng tổng quát hóa

- **Chính quy hóa đa dạng**: Kết hợp đồng thời các kỹ thuật Dropout ở nhiều mức (0.05-0.2), BatchNorm, weight decay (1e-4) và label smoothing (0.05):

```
nn.Dropout2d(0.1),
# ...
nn.Dropout(0.2),
# ...
criterion = nn.CrossEntropyLoss(label_smoothing=0.05)
optimizer = optim.AdamW(model.parameters(), lr=0.0004, weight_decay=1e-4)
```

- **Sự dung hòa dữ liệu**: MixUp và CutMix áp dụng có chọn lọc (chỉ sau epoch 5) với xác suất hợp lý (0.2), tạo phân phối giữa các mẫu huấn luyện mượt mà hơn:

```
if epoch > 5:
    if np.random.rand() > 0.8:  # Probability 0.2
        images, targets_a, targets_b, lam = mixup_data(images, labels)
    # ... other augmentations
```

- **Dừng sớm thông minh**: Theo dõi riêng val_loss để dừng sớm và val_acc để lưu mô hình tốt nhất, với giá trị kiên nhẫn cao (patience=25) cho phép phát hiện cải thiện thực sự:

```
if val_acc > best_acc:
    best_acc = val_acc
    torch.save(model.state_dict(), 'best_model.pth')

if val_loss < best_loss:
    best_loss = val_loss
    early_stop_counter = 0
    torch.save(model.state_dict(), 'best_model_early_stop.pth')
```

- **Phân tầng dữ liệu**: Chia tập dữ liệu theo phân tầng bảo toàn phân phối lớp nguyên bản, tránh mất cân bằng ngẫu nhiên:

```
for cls, indices in class_indices.items():
    np.random.shuffle(indices)
    split = int(0.85 * len(indices))
    train_indices.extend(indices[:split])
    val_indices.extend(indices[split:])
```

Hiệu quả tính toán

- **Song song hóa GPU**: Cấu hình hợp lý cho cudNN và cuBLAS để tối ưu các phép toán song song:

```
os.environ["CUBLAS_WORKSPACE_CONFIG"] = ":4096:8"
torch.backends.cudnn.benchmark = False
```

- **Huấn luyện độ chính xác hỗn hợp**: Sử dụng số thực FP16 kết hợp với GradScaler để giảm sử dụng bộ nhớ và tăng tốc độ tính toán lên đến 3x:

```
scaler = GradScaler(device='cuda' if torch.cuda.is_available() else 'cpu')
with autocast(device_type='cuda' if torch.cuda.is_available() else 'cpu'):
    outputs = model(images)
    loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
```

- **Tối ưu I/O pipeline**: Sử dụng DataLoader với pin_memory và num_workers phù hợp, giảm thời gian chờ CPU-GPU:

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
                          num_workers=4, pin_memory=True)
```

- **Tái sử dụng bộ nhớ**: Quản lý chủ động bộ nhớ GPU với cache clearing, tránh rò rỉ bộ nhớ trong quá trình huấn luyện:

```
if torch.cuda.is_available():
    torch.cuda.empty_cache()
```

Năng lực trích xuất đặc trưng

- **Cơ chế chú ý kênh**: Mô-đun ECA và SE tối ưu trọng số của các kênh đặc trưng với chi phí tính toán thấp:

```
class ECAModule(nn.Module):
    def __init__(self, channels, gamma=2, b=1):
        # ... initialization ...

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).squeeze(-1).transpose(-1, -2)
        y = self.conv(y).transpose(-1, -2).unsqueeze(-1)
        y = self.sigmoid(y)
        return x * y.expand_as(x)
```

- **Biểu diễn đa tỷ lệ**: Kết hợp thông tin từ nhiều mức độ trừu tượng khác nhau thông qua skip connections, bảo toàn cả chi tiết cục bộ và ngữ cảnh toàn cục:

```
def forward(self, x):
    x_skip1 = self.features[0:4](x)
    x = self.features[4:12](x_skip1)
    x_skip2 = self.features[12:14](x)
    x = self.features[14:](x_skip2)
    skip1 = self.skip_connection(x_skip1)
    skip2 = self.skip_connection2(x_skip2)
    x = x + skip1 + skip2  # Multi-scale feature fusion
    x = F.relu(x)
    x = self.classifier(x)
    return x
```

- **Độ bất biến biểu diễn**: Sự kết hợp giữa pooling, stride và tích chập nhóm tạo ra các đặc trưng có tính bất biến cao đối với biến đổi không gian nhỏ.

- **Tăng cường dữ liệu có mục tiêu**: Các biến đổi được chọn lọc (RandomHorizontalFlip, RandomRotation, ColorJitter) với biên độ nhỏ, phản ánh biến thể thực tế của tín hiệu radar:

```
transforms.RandomHorizontalFlip(p=0.2),
transforms.RandomRotation(5),
transforms.ColorJitter(brightness=0.05, contrast=0.05, saturation=0.05),
```

Khả năng mở rộng và ứng dụng thực tế

- **Mô hình trace tối ưu**: Xuất mô hình bằng torch.jit.trace để tạo đồ thị tính toán tối ưu, giảm overhead khi triển khai:

```
example_input = torch.randn(1, 3, 128, 128).to(device)
traced_model = torch.jit.trace(model, example_input)
traced_model.save("22119246_22119234_optimized_2.pt")
```

- **Tính tất định**: Cấu hình cẩn thận để đảm bảo khả năng tái tạo kết quả giữa các lần chạy, cần thiết cho triển khai đáng tin cậy:

```
def seed_everything(seed=42):
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
```

- **Độ phức tạp độc lập kích thước đầu vào**: Sử dụng AdaptiveAvgPool2d trong classifier giúp mô hình xử lý linh hoạt các kích thước đầu vào khác với kích thước huấn luyện:

```
self.classifier = nn.Sequential(
    nn.AdaptiveAvgPool2d(4),   # Works with any input size
    nn.Flatten(),
    # ... more layers ...
)
```

- **Kiến trúc module hóa**: Thiết kế các thành phần có thể tái sử dụng (ResBlock, ECAModule, SEBlock) cho phép dễ dàng điều chỉnh và mở rộng theo yêu cầu ứng dụng.

8.3 Phân tích điểm yếu của mô hình

Dựa trên kết quả hiệu suất và thiết kế mô hình, các điểm yếu chính được xác định như sau:
Thiết kế mô hình

- **Group Convolution**: Sử dụng group convolution (groups=40, 72) làm giảm khả năng học mối quan hệ giữa các kênh, ảnh hưởng đến hiệu suất trên DSB-AM (F1-score 0.9344) và SSB-AM (F1-score 0.9362).
- **Dropout thấp**: Dropout ở các tầng đầu (0.05, 0.1) không đủ mạnh để ngăn overfitting trên các class thiểu số.
- **Thiếu Spatial Attention**: Chỉ có channel attention (ECA, SE), thiếu spatial attention, làm giảm khả năng phân biệt các đặc trưng không gian.

Xử lý dữ liệu

- **Data Augmentation yếu**: Các phép biến đổi (RandomHorizontalFlip p=0.2, RandomRotation 5 độ) quá nhẹ, không đủ đa dạng dữ liệu cho DSB-AM và SSB-AM.
- **Chưa cân bằng class**: Không sử dụng WeightedRandomSampler, dẫn đến hiệu suất thấp trên các class thiểu số.
- **Thiếu tiền xử lý**: Không chuyển đổi tín hiệu radar sang miền tần số (spectrogram), có thể làm mất thông tin quan trọng.

Huấn luyện

- **Mixup/CutMix chưa mạnh**: Xác suất áp dụng thấp (0.2), không đủ để tăng tính tổng quát hóa trên các class khó.
- **Learning Rate chưa tối ưu**: Learning rate 0.0004 có thể hơi cao, ảnh hưởng đến khả năng học chi tiết trên các class khó.

8.4 Hướng cải tiến

Thiết kế mô hình

- **Giảm group convolution**: Giảm hoặc bỏ group convolution (groups=1) để tăng khả năng học mối quan hệ giữa các kênh, cải thiện hiệu suất trên DSB-AM và SSB-AM.
- **Tăng Dropout**: Nâng Dropout ở các tầng đầu từ 0.05, 0.1 lên 0.15 hoặc 0.2 để giảm overfitting trên các class thiểu số.
- **Thêm Spatial Attention**: Bổ sung cơ chế chú ý không gian (như CBAM) để cải thiện khả năng phân biệt đặc trưng không gian.
- **Thêm tầng học đặc trưng**: Thêm 1-2 ResBlock hoặc tăng số kênh (từ 72 lên 96) để học các đặc trưng phức tạp hơn.

Xử lý dữ liệu

- **Tăng cường Data Augmentation**: Áp dụng các phép biến đổi mạnh hơn (RandomAffine, thêm Gaussian noise) cho DSB-AM và SSB-AM để tăng tính đa dạng dữ liệu.
- **Cân bằng class**: Sử dụng WeightedRandomSampler trong DataLoader để tăng tỷ lệ mẫu của DSB-AM và SSB-AM.
- **Tiền xử lý tín hiệu**: Chuyển đổi tín hiệu radar sang miền tần số (spectrogram) hoặc áp dụng bộ lọc (Gabor, Sobel) để làm nổi bật đặc trưng.

Huấn luyện

- **Tăng Mixup/CutMix**: Nâng xác suất áp dụng Mixup và CutMix từ 0.2 lên 0.3-0.4, tăng alpha từ 0.1 lên 0.2 để tăng tính tổng quát hóa.
- **Tinh chỉnh Learning Rate**: Giảm learning rate từ 0.0004 xuống 0.0002 và tăng T_0 của scheduler từ 15 lên 20 để học chi tiết hơn.

8.5 Khả năng ứng dụng thực tế

Mô hình phân loại tín hiệu radar đạt độ chính xác 0.9806 với thiết kế nhẹ (297,723 tham số), có tiềm năng ứng dụng trong nhiều lĩnh vực:

Quân sự và quốc phòng

- **Phát hiện và phân loại tín hiệu radar**: Nhận diện các loại tín hiệu (B-FM, Barker, CPFSK, v.v.) trong hệ thống phòng thủ, hỗ trợ phát hiện mục tiêu hoặc nguồn phát tín hiệu đối phương.
- **Chiến tranh điện tử**: Phân tích tín hiệu để đối phó với hệ thống radar thù địch, đặc biệt với tín hiệu LFM và Rect (F1-score 1.0).

Giao thông và hàng không

- **Kiểm soát không lưu**: Phân loại tín hiệu radar để quản lý không lưu, nhận diện máy bay hoặc vật thể bay.
- **Hệ thống tránh va chạm**: Ứng dụng trong ô tô, tàu thủy để phát hiện chướng ngại vật qua tín hiệu radar.

Viễn thông

- **Quản lý phổ tần số**: Phân loại tín hiệu để tối ưu hóa sử dụng phổ tần số, giảm nhiễu.
- **Phát hiện tín hiệu bất thường**: Phát hiện tín hiệu radar không mong muốn trong mạng viễn thông.

Hạn chế và hướng phát triển

- Hiệu suất trên DSB-AM và SSB-AM cần cải thiện để áp dụng trong các kịch bản yêu cầu độ chính xác cao.
- Mô hình nhẹ phù hợp cho thiết bị nhúng (hệ thống radar di động), nhưng cần tối ưu để xử lý thời gian thực.

9 Kết luận

Mô hình phân loại tín hiệu radar ImprovedRadarCNN đã chứng minh được tiềm năng vượt trội trong việc giải quyết bài toán nhận diện các loại tín hiệu radar với độ chính xác tổng thể ấn tượng đạt 0.9806. Với thiết kế kiến trúc nhẹ nhàng (chỉ 297,723 tham số), mô hình không chỉ thể hiện hiệu suất cao trên các lớp tín hiệu như Barker, LFM, và Rect (đạt F1-score hoàn hảo 1.0), mà còn cho thấy khả năng tối ưu hóa tài nguyên, phù hợp với các hệ thống nhúng trong thực tế. Các kỹ thuật hiện đại như ECAModule, SEBlock, cùng với chiến lược huấn luyện sử dụng Mixup và CutMix đã góp phần quan trọng vào việc nâng cao tính tổng quát hóa, giúp mô hình đạt được sự cân bằng tốt giữa hiệu suất và độ phức tạp.

Tuy nhiên, hành trình phát triển mô hình vẫn còn những thách thức cần vượt qua. Hiệu suất trên các lớp tín hiệu như DSB-AM (F1-score 0.9344) và SSB-AM (F1-score 0.9362) chưa đạt được độ chính xác đồng đều như kỳ vọng, cho thấy mô hình vẫn gặp khó khăn trong việc phân biệt các tín hiệu có đặc trưng tương tự nhau. Những hạn chế này xuất phát từ các yếu tố như việc sử dụng group convolution làm giảm khả năng học mối quan hệ phức tạp giữa các kênh, mức độ tăng cường dữ liệu chưa đủ mạnh, và sự thiếu hụt cơ chế chú ý không gian để khai thác các đặc trưng quan trọng trong tín hiệu radar. Dù vậy, đây cũng chính là những cơ hội để mô hình tiến xa hơn trong tương lai.

Với các hướng cải tiến đã đề xuất, bao gồm việc giảm group convolution, tăng cường Dropout, bổ sung cơ chế chú ý không gian, và tối ưu hóa dữ liệu cũng như quá trình huấn luyện, mô hình hoàn toàn có thể đạt được hiệu suất cao hơn, đảm bảo độ chính xác đồng đều trên tất cả các lớp tín hiệu. Hơn thế nữa, nhờ thiết kế nhẹ và hiệu quả, mô hình này hứa hẹn sẽ trở thành một công cụ mạnh mẽ trong các ứng dụng thực tế, từ lĩnh vực quân sự (phát hiện mục tiêu, chiến tranh điện tử), giao thông (kiểm soát không lưu, tránh va chạm), đến viễn thông (quản lý phổ tần số, phát hiện tín hiệu bất thường). ImprovedRadarCNN không chỉ là một bước tiến trong nghiên cứu phân loại tín hiệu radar, mà còn mở ra cánh cửa để khám phá thêm nhiều tiềm năng công nghệ trong thế giới hiện đại, nơi mà sự chính xác và hiệu quả luôn là mục tiêu hàng đầu.