BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA ĐIỆN – ĐIỆN TỬ

BÁO CÁO CUỐI KỲ

MÔN HỌC: CƠ SỞ VÀ ỨNG DỤNG AI

ĐỀ TÀI

PHÂN LOẠI ẢNH BẰNG PYTHON – SỬ DỤNG DEEP LEARNING

Hướng dẫn: **TS. HUỲNH THẾ THIỆN**

Sinh viên: **Nguyễn Trung Đức**

MSSV: 22161116

Nguyễn Duy Hải

MSSV: 22161011

Tp. Hồ Chí Minh – 5/2025

ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐIỆN - ĐIỆN TỬ
www.utex.hcmute.edu.vn

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc
Tp. Hồ Chí Minh, Tháng 05, Năm 2025

TÓM TẮT

Trong kỷ nguyên của truyền thông không dây, việc nhận dạng và phân loại tín hiệu điều chế là yếu tố then chốt trong nhiều ứng dụng, từ giám sát thông minh đến các hệ thống nhận thức vô tuyến. Tuy nhiên, bài toán nhận dạng tín hiệu điều chế lại gặp phải không ít thách thức do sự đa dạng và tính phức tạp của các tín hiệu này. Để giải quyết vấn đề này, nghiên cứu gần đây đã đề xuất một phương pháp đột phá, sử dụng mạng nơ-ron tích chập (CNN) để nhận diện tín hiệu điều chế thông qua hình ảnh. Thay vì trích xuất các đặc trưng thủ công, phương pháp này chuyển đổi tín hiệu điều chế thành các ảnh phổ thời gian–tần số, mở ra cơ hội ứng dụng các kỹ thuật học sâu để phát hiện các đặc trưng phức tạp trong không gian tần số và thời gian. Dự án của chúng tôi áp dụng phương pháp này để phân loại tín hiệu radar từ các ảnh phổ, sử dụng kiến trúc EfficientMiniNet—một mô hình CNN nhẹ với các đặc điểm nổi bật như Depthwise Separable Convolution và Residual Connections, giúp tối ưu hiệu quả và giảm thiểu số lượng tham số (chỉ với ~260.000 tham số). Kết quả thử nghiệm trên tám loại tín hiệu điều chế cho thấy mô hình đạt hơn 87% độ chính xác trên tập huấn luyện và trên 85% độ chính xác trên tập kiểm tra, vượt qua các yêu cầu khắt khe. Phương pháp này không chỉ tận dụng ưu thế tự động trích xuất đặc trưng của mạng nơ-ron tích chập mà còn thể hiện khả năng tổng quát hóa vượt trội trong việc nhận diện các tín hiệu từ đơn giản đến phức tạp. Điều này mở ra một hướng đi mới đầy triển vọng cho việc ứng dụng học sâu trong lĩnh vực xử lý tín hiệu không dây, đồng thời mở rộng khả năng ứng dụng vào các hệ thống radar và các công nghệ nhận dạng tín hiệu trong tương lai.

Tp. Hồ Chí Minh, Tháng 5, Năm 2025

Nhóm thực hiện

DANH MỤC HÌNH ẢNH

DANH MỤC CÁC BẢNG

MỤC LỤC

CHƯƠNG 1 : TỔNG QUAN VỀ HỌC SÂU

1.1 GIỚI THIỆU TRÍ TUỆ NHÂN TẠO VÀ HỌC MÁY

Trí tuệ nhân tạo (AI - Artificial Intelligence) là lĩnh vực nghiên cứu nhằm phát triển các hệ thống máy tính có khả năng thực hiện các nhiệm vụ đòi hỏi trí thông minh của con người như nhận thức, suy luận, học tập, và giải quyết vấn đề.

Học máy (Machine Learning - ML) là một nhánh của AI, tập trung vào việc phát triển các thuật toán cho phép máy tính học từ dữ liệu và cải thiện hiệu suất theo thời gian mà không cần lập trình cụ thể. Thay vì viết các quy tắc cứng, học máy cho phép mô hình tìm ra quy luật từ dữ liệu.

Học máy bao gồm ba nhóm chính:

- Học có giám sát (Supervised Learning):

- Trong học có giám sát, dữ liệu đầu vào được gán nhãn rõ ràng. Mục tiêu là xây dựng một mô hình có thể dự đoán được đầu ra (nhãn) cho dữ liệu mới dựa trên các mẫu học. Ví dụ: phân loại email là "spam" hay "không spam", dự đoán giá nhà từ các đặc điểm như diện tích, vị trí,...
- Một số thuật toán tiêu biểu: Hồi quy tuyến tính (Linear Regression), Cây quyết định (Decision Tree), SVM, Mạng nơ-ron (Neural Networks).

- Học không giám sát (Unsupervised Learning):

- Ở dạng này, dữ liệu đầu vào không có nhãn. Mục tiêu là tìm ra cấu trúc hoặc mẫu ẩn trong dữ liệu. Ví dụ: phân nhóm khách hàng theo hành vi mua sắm, giảm số chiều dữ liệu để trực quan hóa.
- Các thuật toán phổ biến: K-means, PCA (Phân tích thành phần chính), DBSCAN, Autoencoder.

- Học tăng cường (Reinforcement Learning):

- Là quá trình học dựa trên hành động và phần thưởng. Một "tác nhân" (agent) sẽ tương tác với môi trường, nhận phản hồi dưới dạng phần thưởng (reward) và dần dần học cách hành động tối ưu để tối đa hóa tổng phần thưởng tích lũy.

- Ứng dụng tiêu biểu: robot học cách di chuyển, AI chơi game như cờ vua, Go, hay các trò chơi điện tử.

1.2 KHÁI NIỆM HỌC SÂU (DEEP LEARNING)

Học sâu (Deep Learning) là một nhánh của học máy, đóng vai trò quan trọng trong lĩnh vực trí tuệ nhân tạo hiện đại. Sự phát triển mạnh mẽ của các ứng dụng thông minh – như xe tự lái, nhận diện khuôn mặt, hay trợ lý ảo – là minh chứng rõ rệt cho hiệu quả của các mô hình học sâu.

Mối quan hệ giữa AI, Học máy và Học sâu

- Trí tuệ nhân tạo (AI): là lĩnh vực tổng quát nhất, nghiên cứu cách tạo ra trí thông minh nhân tạo
- Học máy (Machine Learning): là một nhánh của AI, tìm cách để máy học từ dữ liệu và đưa ra quyết định.
- Học sâu (Deep Learning): là một cách tiếp cận mới trong học máy, sử dụng các mạng nơ-ron nhiều lớp để mô hình hóa và học dữ liệu phức tạp.

Các thành phần cơ bản của một hệ thống học máy

- Dữ liệu đầu vào: như hình ảnh (cho nhận dạng ảnh), âm thanh (cho nhận dạng giọng nói), v.v.
- Nhãn (label): đầu ra mong muốn tương ứng với mỗi đầu vào.
- Hàm mất mát (loss function): đo độ sai lệch giữa đầu ra dự đoán và đầu ra thật.
- Thuật toán học: điều chỉnh các tham số mô hình để tối thiểu hóa sai số.

Tư tưởng cốt lõi của học sâu

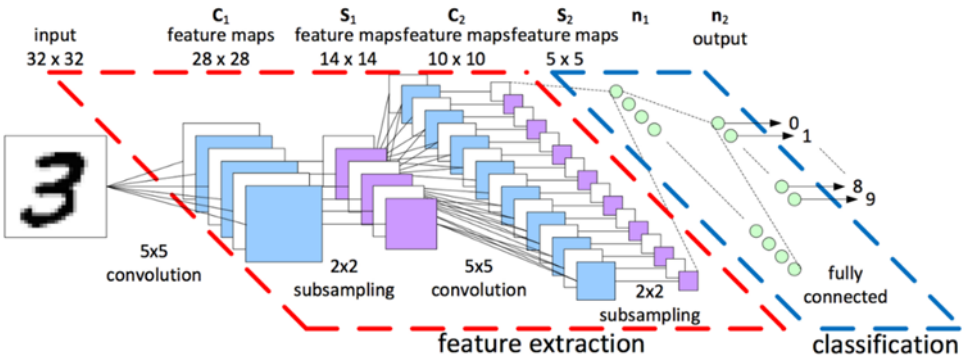
Học sâu tập trung vào việc biểu diễn dữ liệu ở nhiều mức trừu tượng khác nhau thông qua nhiều lớp mạng nơ-ron nối tiếp. Mỗi lớp trong mạng có nhiệm vụ biến đổi và mã hóa dữ liệu đầu vào theo cách giúp các lớp sau dễ dàng hơn trong việc phân loại hoặc dự đoán.

- Biểu diễn dữ liệu (representation): là cách tổ chức và mã hóa thông tin. Ví dụ: một màu có thể được biểu diễn bằng RGB hoặc HSV – hai cách mã hóa khác nhau cho cùng một thực thể.
- Mô hình học sâu giúp tìm ra cách biểu diễn tối ưu nhất để thực hiện các tác vụ cụ thể như phân loại, dự đoán hay nhận dạng.

Cấu trúc của mạng học sâu

- Gồm nhiều lớp nơ-ron nhân tạo, thường từ 10 đến vài trăm lớp.
- Các lớp có thể là:

- Lớp tích chập (Convolutional layer): trích xuất đặc trưng không gian trong ảnh.
- Lớp kết nối đầy đủ (Fully connected layer): kết nối toàn bộ các nơ-ron.
- Lớp hồi quy (Recurrent layer): dùng trong xử lý chuỗi (text, âm thanh).



Hình 1.1: Mô hình học sâu

Mô phỏng bộ não người

- Mạng nơ-ron nhân tạo lấy cảm hứng từ cấu trúc của nơ-ron sinh học.
- Tuy nhiên, hiện nay chưa có bằng chứng khẳng định các mô hình học sâu hoạt động giống hoàn toàn với bộ não con người.
- Các thuật toán như lan truyền ngược (backpropagation) tuy hiệu quả nhưng vẫn chỉ là cách tiếp cận gần đúng, chưa mô phỏng được chính xác cách não bộ học tập.

Ví dụ minh họa

Trong bài toán nhận diện chữ viết tay, dữ liệu đầu vào là hình ảnh của các con số. Các lớp trong mạng học sâu sẽ lần lượt trích xuất cạnh, đường cong, hình dạng tổng thể,… và cuối cùng phân loại thành các con số cụ thể.

1.3 ƯU ĐIỂM VÀ THÁCH THỨC CỦA HỌC SÂU

1.3.1 Ưu điểm

a. Tự động trích xuất đặc trưng từ dữ liệu

Khác với các phương pháp học máy truyền thống cần kỹ sư lựa chọn và thiết kế đặc trưng thủ công (feature engineering), học sâu có khả năng học trực tiếp đặc trưng từ dữ liệu thô (hình ảnh, âm thanh, văn bản…) thông qua các tầng mạng nơ-ron.

b. Khả năng xử lý dữ liệu phi cấu trúc

Học sâu rất hiệu quả trong việc xử lý các dạng dữ liệu phức tạp như:

- Hình ảnh (image)
- Âm thanh (audio)
- Văn bản (text)
- Video

c. Hiệu suất cao trong các tác vụ khó

Các mô hình học sâu thường đạt độ chính xác vượt trội trong những bài toán phức tạp như nhận diện hình ảnh, dịch máy, hoặc chơi game.

d. Khả năng mở rộng với dữ liệu lớn

Khi có nhiều dữ liệu, các mô hình học sâu thường học được các đặc trưng tổng quát hơn và cho kết quả tốt hơn so với các phương pháp học máy truyền thống.

e. Khả năng biểu diễn dữ liệu phức tạp

Mỗi lớp trong mạng sâu biểu diễn một mức trừu tượng khác nhau, cho phép mô hình học được các đặc trưng sâu sắc và có ý nghĩa hơn từ dữ liệu.

1.3.2 Thách thức

a.Yêu cầu dữ liệu lớn để huấn luyện

Các mô hình học sâu cần số lượng dữ liệu huấn luyện khổng lồ để đạt được hiệu quả tốt. Thiếu dữ liệu có thể khiến mô hình học sai hoặc overfit.

b. Chi phí tính toán cao

Việc huấn luyện và vận hành mạng học sâu đòi hỏi phần cứng mạnh (GPU, TPU) và thời gian xử lý lâu, đặc biệt với các mạng rất sâu hoặc dữ liệu lớn.

c. Khó giải thích và thiếu minh bạch

Học sâu là một "hộp đen (black box)", rất khó để lý giải vì sao mô hình lại đưa ra một quyết định cụ thể. Điều này là trở ngại lớn trong các lĩnh vực yêu cầu sự minh bạch như y tế, tài chính.

d. Dễ bị overfitting

Do có nhiều tham số, nếu dữ liệu không đủ đa dạng hoặc không được xử lý kỹ, mô hình có thể ghi nhớ dữ liệu thay vì học các quy luật tổng quát – gây hiện tượng quá khớp.

e. Tối ưu mô hình phức tạp

Việc thiết kế kiến trúc mạng, lựa chọn siêu tham số (hyperparameters) và huấn luyện tối ưu đòi hỏi kinh nghiệm và kỹ thuật cao.

1.4 CÁC ỨNG DỤNG PHỔ BIẾN CỦA HỌC SÂU

Thị giác máy tính là một lĩnh vực trong khoa học máy tính nghiên cứu các thuật toán và chương trình nhằm xử lý ảnh số, phân tích và nhận dạng ảnh, hoặc trích xuất các đặc trưng và thông tin có ý nghĩa từ ảnh cũng như các dữ liệu đa chiều trong thế giới thực. Có nhiều kỹ thuật được áp dụng trong thị giác máy tính, trong đó học sâu (deep learning) là một phương pháp nổi bật hiện nay, mang lại hiệu quả vượt trội trong nhiều ứng dụng thực tiễn. Các mạng học sâu đã được ứng dụng thành công trong

nhiều bài toán của thị giác máy tính như phân loại ảnh, phát hiện đối tượng, phân loại đối tượng, chuyển đổi phong cách ảnh, theo dõi đối tượng,.

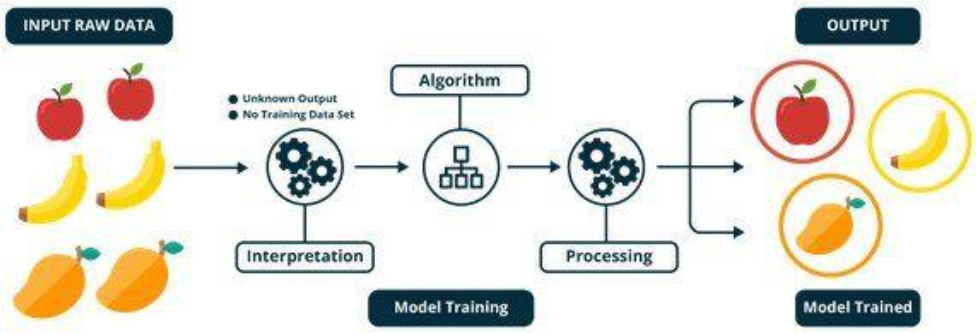
1.4.1 Phân loại ảnh

Phân loại ảnh là một trong những bài toán điển hình và cơ bản trong lĩnh vực thị giác máy tính. Trong trường hợp mỗi ảnh chỉ chứa một đối tượng chính, bài toán này còn được gọi là phân loại đối tượng. Thực chất, hai khái niệm này có thể được sử dụng thay thế cho nhau trong nhiều ngữ cảnh.

Phân loại ảnh được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau. Chẳng hạn, trong y tế, thuật toán có thể được sử dụng để phân loại ảnh chụp X-Ray nhằm hỗ trợ chẩn đoán bệnh. Đây là ví dụ điển hình của bài toán phân loại nhị phân, trong đó đầu ra chỉ bao gồm hai lớp: có hoặc không, dương tính hoặc âm tính, tương ứng với giá trị 1 hoặc 0.

Một ví dụ kinh điển khác thường được sử dụng để minh họa cho các mô hình học máy và học sâu là bài toán nhận dạng chữ số viết tay (ví dụ như bộ dữ liệu MNIST). Về bản chất, đây cũng là một bài toán phân loại, nhưng thuộc loại đa lớp, với 10 lớp tương ứng với các chữ số từ 0 đến 9.

Các mô hình phân loại ảnh yêu cầu phải được huấn luyện với một tập dữ liệu đủ lớn và đa dạng để đạt được độ chính xác cao và khả năng tổng quát tốt trên các dữ liệu chưa từng thấy.

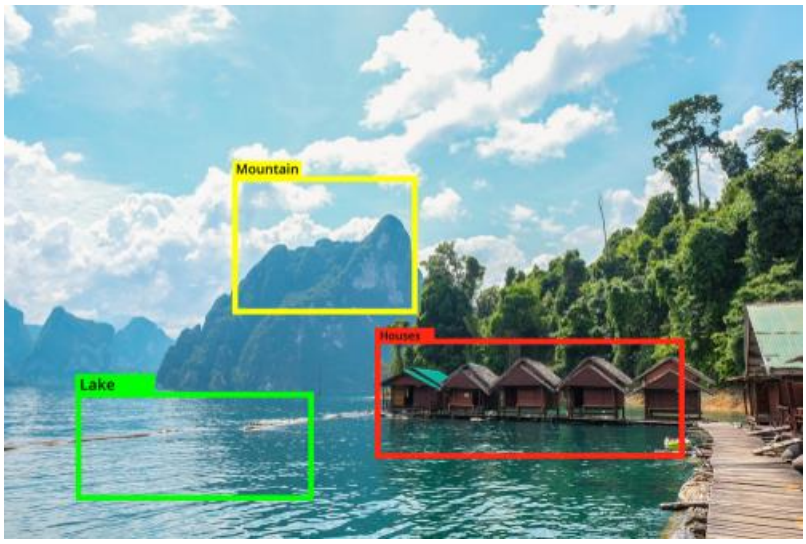


Hình 1.2: Minh họa phân loại ảnh

1.4.2 Phát hiện đối tượng

Tách đối tượng là một bước quan trọng trong nhiều bài toán thị giác máy tính, đặc biệt khi ảnh đầu vào chứa nhiều đối tượng khác nhau. Trong trường hợp này, trước khi tiến hành phân loại, hệ thống cần xác định vị trí (định vị) của từng đối tượng trong ảnh. Việc xác định vị trí giúp mô hình biết chính xác khu vực nào trong ảnh chứa thông tin cần thiết để phân loại.

Khi ảnh có chứa nhiều đối tượng, mô hình cần thực hiện đồng thời hai nhiệm vụ: định vị (tìm vị trí của từng đối tượng) và phân loại (xác định lớp của đối tượng đó). Đây chính là bài toán phát hiện đối tượng (object detection), nơi mỗi đối tượng được biểu diễn bằng một hộp giới hạn (bounding box) kèm theo nhãn lớp.

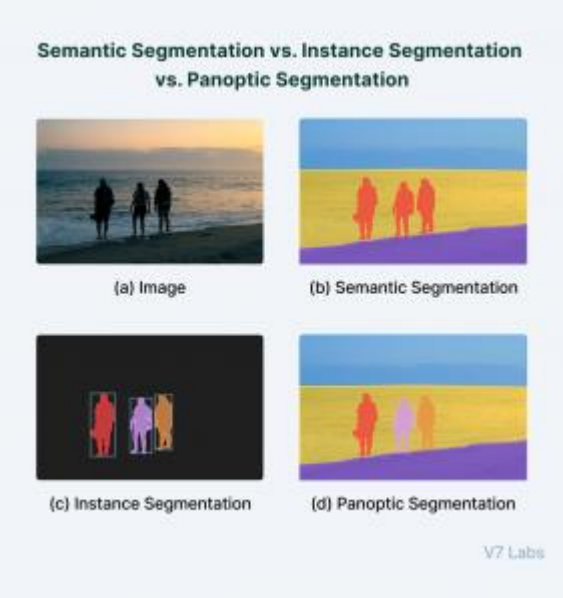


Hình 1.3: Minh họa phát hiện đối tượng

1.4.3 Phân đoạn đối tượng

Trong bài toán phân đoạn đối tượng (object segmentation), các đối tượng trong ảnh không chỉ được phát hiện mà còn được khoanh vùng chính xác đến từng điểm ảnh (pixel). Các đường biên của từng đối tượng sẽ được xác định rõ ràng, giúp tách biệt chúng ra khỏi nền hoặc các đối tượng khác trong ảnh.

Khác với bài toán nhận dạng hay phát hiện đối tượng, nơi mà đối tượng được khoanh vùng bằng hộp giới hạn (bounding box) và được gán nhãn lớp, phân đoạn đối tượng tập trung vào việc xác định hình dạng và biên của đối tượng, mà không nhất thiết phải phân loại đối tượng đó.



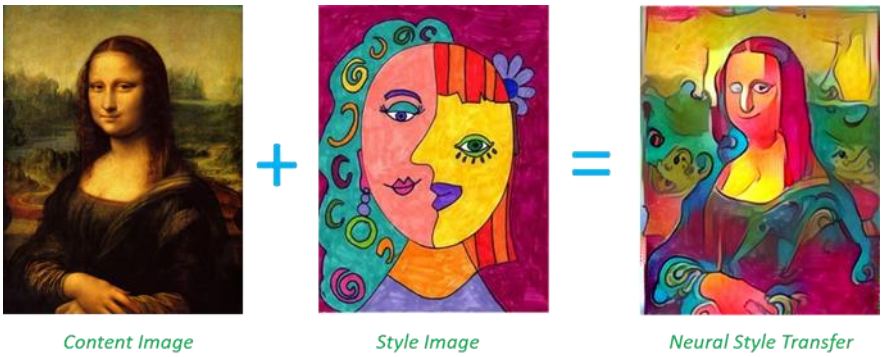
Hình 1.4: Minh họa phân đoạn đối tượng

1.4.4 Chuyển mẫu

Chuyển mẫu (style transfer) là một kỹ thuật trong thị giác máy tính, trong đó một mô hình học sâu được huấn luyện để áp dụng phong cách (style) của một hoặc nhiều ảnh vào một ảnh nội dung mới, từ đó tạo ra một ảnh tổng hợp mang đặc điểm của cả hai. Phong cách ở đây có thể là màu sắc, nét vẽ, họa tiết,... thường được trích xuất từ các tác phẩm nghệ thuật nổi tiếng hoặc các ảnh có tính chất đặc biệt.

Kỹ thuật chuyển mẫu thường được xem là một dạng của chuyển đổi ảnh (image transformation), trong đó mô hình hoạt động giống như một bộ lọc thông minh, không chỉ thay đổi màu sắc hay kết cấu mà còn giữ được cấu trúc nội dung của ảnh gốc.

Style transfer đã được ứng dụng rộng rãi trong nhiều lĩnh vực như nghệ thuật số, thiết kế, ứng dụng chỉnh sửa ảnh trên điện thoại di động, v.v.



Hình 1.5: Minh họa chuyển mẫu

1.4.5 Theo dõi đối tượng

Theo dõi đối tượng (object tracking) là bài toán trong thị giác máy tính nhằm xác định và theo dõi vị trí của một hoặc nhiều đối tượng trong chuỗi các ảnh liên tiếp theo thời gian, như các khung hình trong một đoạn video. Thông thường, trong khung hình đầu tiên, đối tượng mục tiêu sẽ được xác định và khoanh vùng bằng một hộp giới hạn (bounding box). Sau đó, các thuật toán sẽ liên tục cập nhật vị trí của đối tượng đó trong các khung hình tiếp theo dựa trên đặc trưng hình ảnh và chuyển động.

Object tracking là một trong những bài toán quan trọng và phổ biến trong xử lý ảnh động hoặc video, với nhiều ứng dụng thực tiễn như giám sát an ninh, phân tích hành vi, thể thao, và đặc biệt là trong thị giác máy tính cho xe tự hành. Trong các hệ thống xe tự lái, object tracking giúp phát hiện và theo dõi chuyển động của các phương tiện hoặc người đi đường, từ đó hỗ trợ hệ thống trong việc dự đoán hướng di chuyển và đưa ra các quyết định điều khiển phù hợp để tránh va chạm.

Hình 1.6: Minh họa việc theo dõi đối tượng

CHƯƠNG 2: CÁC THUẬT TOÁN LIÊN QUAN

2.1 MẠNG NƠ RON NHÂN TẠO (ANN)

Mạng nơ ron nhân tạo (Artificial Neural Network - ANN) là một mô hình tính toán lấy cảm hứng từ cấu trúc và chức năng của não người. ANN được xây dựng từ các đơn vị tính toán gọi là nơ ron nhân tạo (artificial neurons) – tương tự như nơ ron sinh học – và được tổ chức thành các lớp (layers). Mỗi nơ ron trong ANN nhận đầu vào, thực hiện một phép biến đổi số học (thường là một phép biến đổi tuyến tính) kết hợp với một hàm phi tuyến (hàm kích hoạt), sau đó truyền tín hiệu đầu ra đến các nơ ron khác.

ANN thường được sử dụng trong nhiều bài toán học máy, đặc biệt là:

- Phân loại (Classification): Phân loại ảnh, văn bản, âm thanh,...
- Dự đoán (Prediction): Dự đoán giá cổ phiếu, nhu cầu thị trường,...
- Hồi quy (Regression): Ước lượng giá trị liên tục như nhiệt độ, giá trị bất động sản,...

2.1.1 Kiến trúc cơ bản

Một mạng nơ ron nhân tạo (ANN) được cấu trúc từ nhiều lớp nơ ron nhân tạo, được tổ chức theo từng lớp theo chiều sâu từ trái sang phải, bao gồm ba loại lớp chính: lớp đầu vào (Input Layer), lớp ẩn (Hidden Layers) và lớp đầu ra (Output Layer). Các lớp này phối hợp với nhau để tiếp nhận dữ liệu, xử lý thông tin và đưa ra dự đoán.

Lớp đầu vào (Input Layer)

- Là lớp đầu tiên của mạng, đóng vai trò tiếp nhận dữ liệu đầu vào dưới dạng vector đặc trưng (feature vector).
- Số lượng nơ ron trong lớp này bằng với số lượng đặc trưng (features) của dữ liệu. Ví dụ: nếu ảnh đầu vào có kích thước 28×28 pixel và được đưa vào mạng dưới dạng vector phẳng, thì lớp đầu vào sẽ có 784 nơ ron.
- Mỗi nơ ron trong lớp này chỉ đơn giản là truyền dữ liệu sang lớp tiếp theo, không thực hiện phép biến đổi phi tuyến nào.

Lớp ẩn (Hidden Layers)

- Là nơi diễn ra phần lớn các tính toán và học đặc trưng của mạng.
- Mạng có thể có một hoặc nhiều lớp ẩn tùy theo độ phức tạp của bài toán và thiết kế mạng.
- Mỗi nơ ron trong lớp ẩn thực hiện các phép biến đổi đầu vào bao gồm:

    1. Tính tổng có trọng số của đầu vào:

    $$z = \sum wixi + b$$

    2. Áp dụng hàm kích hoạt phi tuyến lên kết quả:

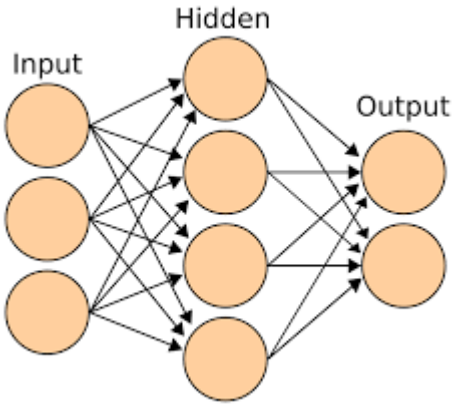
    $$a = f(z)$$

Trong đó f có thể là sigmoid, tanh, hoặc ReLU,...

- Mục tiêu của lớp ẩn là trích xuất các đặc trưng trừu tượng hơn và phi tuyến hơn từ dữ liệu gốc, giúp mạng có thể mô hình hóa các mối quan hệ phức tạp.

Lớp đầu ra (Output Layer)

- Là lớp cuối cùng, đưa ra kết quả dự đoán của mạng dựa trên các đặc trưng đã được học từ lớp ẩn.
- Số lượng nơ ron của lớp này phụ thuộc vào loại bài toán:
    - Phân loại nhị phân: thường dùng 1 nơ ron với hàm kích hoạt sigmoid.
    - Phân loại đa lớp: dùng số nơ ron bằng số lớp, kết hợp với hàm softmax để tạo xác suất phân lớp.
    - Hồi quy (regression): thường dùng 1 nơ ron, không cần hàm kích hoạt hoặc dùng hàm tuyến tính.



Hình 2.1: Các lớp chính của ANN

Kết nối hoàn toàn (Fully Connected - FC)

- Trong kiến trúc mạng truyền thống (feedforward ANN), các nơ ron giữa các lớp thường được kết nối đầy đủ, tức là mỗi nơ ron của một lớp kết nối với tất cả nơ ron của lớp kế tiếp.
- Mỗi kết nối có một trọng số riêng biệt, biểu thị mức độ ảnh hưởng của đầu vào đến nơ ron tiếp theo.
- Tập hợp các trọng số này chính là thông số cần được học trong quá trình huấn luyện.

Mô hình toán học tổng quát

Giả sử có một lớp với vector đầu vào $x \in R^n$, trọng số $W \in R^{m \times n}$, vector bias $b \in R^m$, và hàm kích hoạt f, đầu ra $y \in R^m$ của lớp sẽ là:

$$y = f(W \cdot x + b)$$

Biểu thức này có thể được áp dụng cho từng lớp, từ lớp đầu vào đến lớp đầu ra, mô tả toàn bộ dòng truyền dữ liệu trong mạng ANN.

2.1.2 Cơ chế hoạt động

Truyền xuôi (Forward Propagation): là quá trình truyền tín hiệu từ lớp đầu vào qua các lớp ẩn đến lớp đầu ra. Ở mỗi lớp, tín hiệu đầu vào được xử lý thông qua phép toán tuyến tính kết hợp với hàm kích hoạt phi tuyến.

Công thức mô tả truyền tín hiệu tại mỗi lớp là:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = f(z^{(l)})$$

Ở đây:

- $z^{(l)}$ là tín hiệu chưa qua hàm kích hoạt của lớp l,
- $a^{(l-1)}$ là đầu ra của lớp trước đó (lớp l−1),
- $W^{(l)}$ là ma trận trọng số của lớp l,
- $b^{(l)}$ là vector bias của lớp l,
- $f(\cdot)$ là hàm kích hoạt tại lớp l.

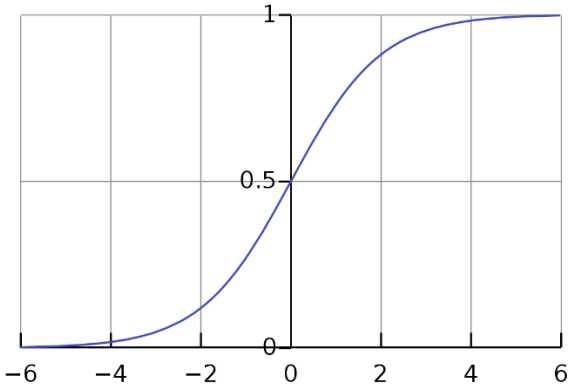
Quá trình này được lặp lại cho tất cả các lớp trong mạng cho đến khi đạt đến lớp đầu ra.

Hàm kích hoạt (Activation Functions): Hàm kích hoạt giúp đưa tính phi tuyến vào mô hình, cho phép mạng học được các mối quan hệ phức tạp hơn giữa các đặc trưng. Dưới đây là ba hàm kích hoạt phổ biến:

Sigmoid: Hàm sigmoid là một hàm logistic được sử dụng phổ biến trong các bài toán phân loại nhị phân:

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

Hàm sigmoid cho ra giá trị trong khoảng (0, 1), điều này giúp mô hình có thể được sử dụng cho các bài toán phân loại xác suất.

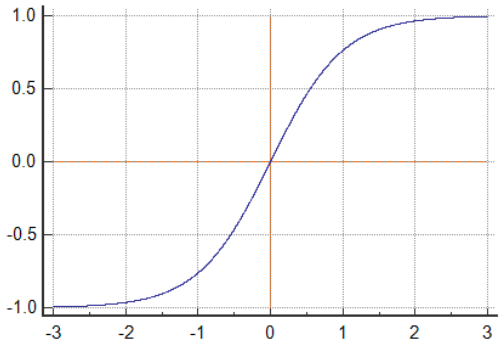


Hình 2.2: Hình biểu diễn đồ thị hàm sigmoid

Tanh (Hyperbolic Tangent): Hàm tanh là hàm hyperbolic có đầu ra nằm trong khoảng (-1, 1):

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

Tanh thường được sử dụng khi ta cần giá trị đầu ra có thể có cả dấu âm, giúp cải thiện hiệu suất trong một số bài toán.



Hình 2.3: Hình biểu diễn đồ thị hàm tanh

ReLU (Rectified Linear Unit): Hàm ReLU là một trong những hàm kích hoạt phổ biến nhất, đặc biệt trong các mạng học sâu (Deep Learning):

$$ReLU(x) = \max\{0, x\} = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

ReLU giúp giảm thiểu vấn đề vanishing gradient và thường được sử dụng trong các mạng nơ ron hiện đại.

Truyền ngược (Backpropagation): là thuật toán sử dụng đạo hàm để lan truyền lỗi từ đầu ra ngược lại các lớp trước nhằm cập nhật trọng số sao cho hàm mất mát giảm dần. Quá trình này là nền tảng cho các thuật toán tối ưu, chẳng hạn như Gradient Descent.

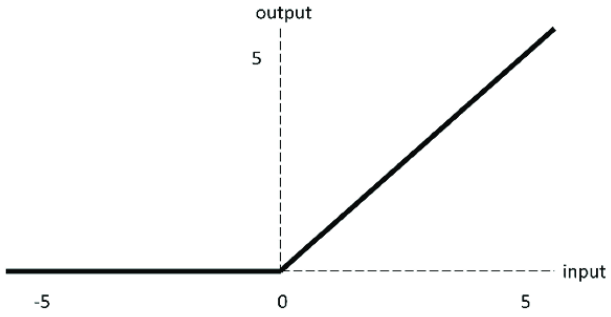
Trong quá trình lan truyền ngược, ta tính đạo hàm của hàm mất mát đối với trọng số tại mỗi lớp:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{W}^{(\mathrm{l})}} = \frac{\partial \mathrm{L}}{\partial \mathrm{a}^{(\mathrm{l})}} \cdot \frac{\partial \mathrm{a}^{(\mathrm{l})}}{\partial \mathrm{z}^{(\mathrm{l})}} \cdot \frac{\partial \mathrm{z}^{(\mathrm{l})}}{\partial \mathrm{W}^{(\mathrm{l})}}$$

Để cập nhật trọng số, sử dụng công thức:

$$\mathrm{W}^{(\mathrm{l})} = \mathrm{W}^{(\mathrm{l})} - \eta \cdot \frac{\partial \mathrm{z}^{(\mathrm{l})}}{\partial \mathrm{W}^{(\mathrm{l})}}$$

Với η là tỷ lệ học (learning rate) và L là hàm mất mát. Quá trình này lặp lại cho đến khi mạng hội tụ hoặc đạt được số vòng lặp tối đa.



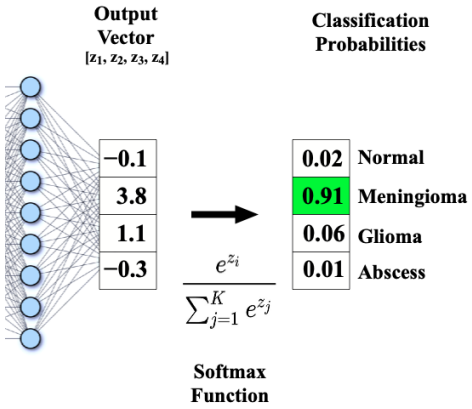
Hình 2.4: Đồ thị biểu diễn hàm ReLU

Ngoài ba hàm kích hoạt phổ biến trên thì còn có **hàm softmax**: Hàm softmax được sử dụng nhiều trong kỹ thuật học sâu. Hàm softmax được sử dụng làm hàm kích hoạt cho các nơ-ron ở ngõ ra hơn là các nơ-ron ở các lớp ẩn. Ngõ ra của hàm softmax tương ứng với phân bố xác suất phân loại của các ngõ ra. Công thức toán học của hàm softmax như sau:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

Trong đó z là vector ngõ ra của các nơ-ron. Ngõ ra hàm softmax cho biết xác suất của ngõ ra trong tổng các ngõ ra. Do đó, hàm softmax được đặt ở lớp ngõ ra trong các bài toán phân loại. Dựa vào xác suất ngõ ra, mạng nơ-ron có thể dự đoán được mẫu hiện tại có khả năng thuộc lớp nào cao nhất. Trong các mạng nơ-ron học sâu, hàm softmax được dùng cho các nơ-ron ở lớp ngõ ra.

Trong khi các hàm kích hoạt khác như sigmoid, tanh và ReLU tạo ra ngõ ra các nơ-ron độc lập. Hàm kích hoạt chỉ có nhiệm vụ chỉnh hóa ngõ ra. Hàm softmax đặt ngõ ra mỗi nơ-ron trong mối quan hệ với tổng các nơ-ron còn lại.



Hình 2.5: Sơ đồ biểu diễn hàm softmax

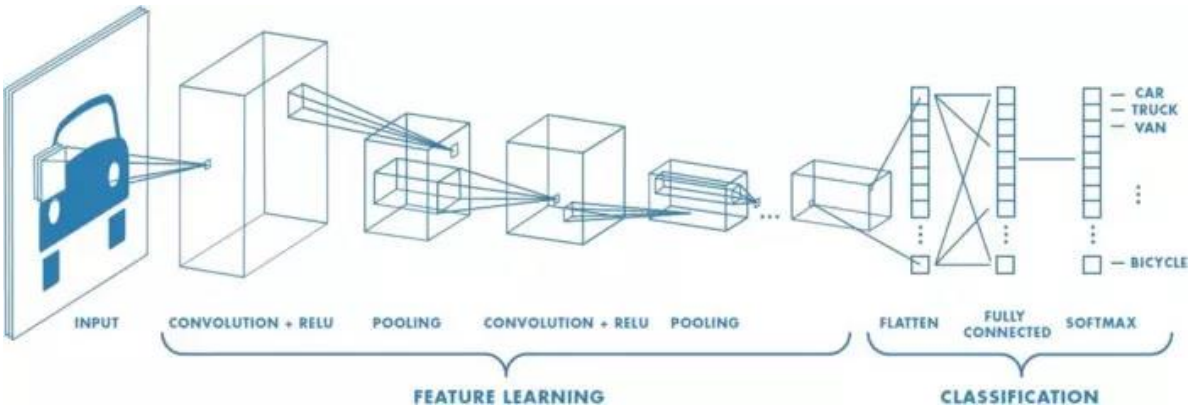
2.2 MẠNG NƠ RON TÍCH CHẬP (CNN)

Mạng nơ ron tích chập (Convolutional Neural Network – CNN) là một kiến trúc mạng nơ ron chuyên biệt được thiết kế để khai thác hiệu quả các mối quan hệ không gian trong dữ liệu có cấu trúc lưới, nổi bật nhất là hình ảnh (dạng ma trận 2 chiều) và video (ma trận 3 chiều hoặc chuỗi ảnh 2 chiều theo thời gian). CNN đã chứng minh tính vượt trội trong nhiều lĩnh vực, đặc biệt là thị giác máy tính (computer vision) và ngày càng được mở rộng sang các lĩnh vực như xử lý ngôn ngữ tự nhiên (khi xử lý văn bản dưới dạng ma trận từ vựng) và dữ liệu thời gian.

Đặc trưng chính của CNN

Khác với mạng nơ ron truyền thống (ANN) – nơi mỗi nơ ron của một lớp được kết nối hoàn toàn với tất cả các nơ ron của lớp trước – CNN sử dụng một cơ chế tích chập cục bộ (local receptive fields) và chia sẻ trọng số (weight sharing), giúp giảm đáng

kể số lượng tham số cần học và làm cho mô hình hoạt động hiệu quả hơn trên dữ liệu có tính chất không gian.



Hình 2.6: Hình ảnh về mạng tích chập (CNN)

2.2.1 Kiến trúc CNN cơ bản

Kiến trúc của một mạng CNN điển hình gồm nhiều tầng (layers) được sắp xếp theo trình tự, với mục đích trích xuất dần các đặc trưng từ dữ liệu đầu vào (thường là ảnh) và đưa ra kết quả phân loại hoặc dự đoán. Dưới đây là mô tả chi tiết các tầng cơ bản trong một mạng CNN:

Lớp tích chập (Convolution Layer): Đây là lớp cốt lõi trong CNN. Tầng này sử dụng các bộ lọc (filters hoặc kernels) – thường là ma trận nhỏ có kích thước phổ biến như 3×3, 5×5 – để trượt (slide) qua toàn bộ ảnh đầu vào và tính tích chập (convolution) tại mỗi vị trí. Kết quả là tạo ra các bản đồ đặc trưng (feature maps), đại diện cho sự hiện diện của một đặc trưng nào đó tại các vị trí khác nhau trên ảnh.

- Mỗi kernel học một loại đặc trưng: cạnh (edge), góc, họa tiết, v.v.
- Công thức tích chập 2D giữa ảnh đầu vào I và kernel K:

$$(I * K)(i,j) = m\sum n\sum I(i + m,j + n) . K(m,n)$$

- Một ảnh đầu vào RGB sẽ có 3 kênh (channels), nên kernel cũng cần có cùng chiều sâu.

Lưu ý: Việc padding (thêm pixel xung quanh ảnh) và stride (bước nhảy của kernel) cũng ảnh hưởng đến kích thước đầu ra.

Lớp kích hoạt (Activation Layer): Sau lớp tích chập, kết quả đầu ra được đưa qua một hàm kích hoạt phi tuyến để tăng khả năng biểu diễn của mô hình.

- ReLU (Rectified Linear Unit) là hàm kích hoạt phổ biến nhất, giúp tránh hiện tượng gradient biến mất và tăng tốc quá trình hội tụ:

$$f(x) = \max(0,x)$$

- Các hàm khác như Leaky ReLU, ELU, hoặc Sigmoid, Tanh cũng có thể được sử dụng tùy vào bài toán cụ thể.

Lớp gộp (Pooling Layer): Lớp gộp giúp giảm kích thước không gian của các feature maps, từ đó:

- Giảm số lượng tham số.
- Hạn chế hiện tượng overfitting.
- Tăng tính bất biến trước dịch chuyển nhỏ trong ảnh.

Hai phương pháp pooling phổ biến:

- Max Pooling: Lấy giá trị lớn nhất trong mỗi vùng con.
- Average Pooling: Tính trung bình các giá trị trong vùng con.

Ví dụ: MaxPooling 2×2 với stride = 2 sẽ giảm kích thước ảnh còn một nửa.

Lớp Fully Connected (FC Layer): Sau khi ảnh được đưa qua nhiều tầng tích chập và gộp, phần đầu ra sẽ là các đặc trưng đã trừu tượng hóa. Các đặc trưng này sẽ được "làm phẳng" (flatten) thành một vector và đưa vào một hoặc nhiều lớp fully connected – giống như mạng ANN truyền thống.

- Lớp FC có nhiệm vụ tổng hợp toàn bộ đặc trưng và thực hiện việc phân loại hoặc hồi quy.
- Đầu ra cuối cùng của mạng (output layer) phụ thuộc vào bài toán:
    - Phân loại nhị phân → 1 nơ ron + hàm Sigmoid.

- Phân loại nhiều lớp → Softmax.
- Hồi quy → Linear.

2.2.2 Ưu điểm

Mạng nơ ron tích chập (CNN) sở hữu nhiều ưu điểm nổi bật so với mạng nơ ron truyền thống (ANN), đặc biệt trong các bài toán liên quan đến dữ liệu ảnh, video và dữ liệu có cấu trúc không gian. Cụ thể như sau:

2.2.2.1 Tự động trích xuất đặc trưng (Automatic Feature Extraction)

- Khác với các phương pháp học máy truyền thống (SVM, KNN, Decision Tree,...), vốn đòi hỏi kỹ sư đặc trưng (feature engineer) phải thủ công thiết kế các đặc trưng đầu vào, CNN có khả năng tự học và trích xuất đặc trưng một cách tự động từ dữ liệu thô (raw data).
- Các tầng tích chập (convolutional layers) trong CNN học được các đặc trưng từ đơn giản (như cạnh, đường thẳng) đến phức tạp (như khuôn mặt, chi tiết vật thể).
- Điều này giúp CNN giảm đáng kể công sức thủ công và tăng khả năng tổng quát, đặc biệt trong các bài toán phức tạp như nhận diện khuôn mặt, phân loại vật thể,...

2.2.2.2 Hiệu quả không gian (Spatial Efficiency)

- CNN khai thác tính cục bộ của ảnh, tức là các điểm ảnh gần nhau thường mang thông tin liên quan, giúp các kernel nhỏ có thể học được đặc trưng mạnh mẽ chỉ với vùng nhìn (receptive field) hạn chế.
- Do mỗi nơ ron trong lớp tích chập chỉ liên kết với một vùng nhỏ của đầu vào, thay vì toàn bộ ảnh như trong ANN, mạng CNN:
  - Giảm chi phí tính toán.
  - Giữ được tính cục bộ và cấu trúc không gian của dữ liệu.

- Nhờ đó, CNN đặc biệt phù hợp cho xử lý hình ảnh và video, nơi mà vị trí tương đối và cấu trúc không gian của các đối tượng rất quan trọng.

2.2.2.3 Giảm số lượng tham số (Parameter Sharing)

- Một trong những yếu tố khiến CNN nhẹ hơn và dễ huấn luyện hơn so với ANN là nhờ vào cơ chế chia sẻ trọng số (weight sharing).
- Trong CNN, cùng một kernel được dùng để quét toàn bộ ảnh, thay vì học một trọng số riêng cho mỗi pixel như trong fully connected layers.
- Ví dụ: Một kernel 3×3 chỉ cần học 9 trọng số (hoặc 27 nếu ảnh có 3 kênh), trong khi một lớp fully connected với đầu vào 1000 pixels và 500 nơ ron sẽ cần tới 500,000 trọng số.
- Việc giảm số lượng tham số mang lại nhiều lợi ích:
    - Giảm nguy cơ overfitting.
    - Tiết kiệm bộ nhớ và thời gian huấn luyện.
    - Cho phép huấn luyện mạng sâu hơn với dữ liệu lớn.

Với những ưu điểm kể trên, CNN đã và đang trở thành công cụ không thể thiếu trong các ứng dụng học sâu (deep learning) hiện đại, đặc biệt là trong:

- Nhận diện khuôn mặt (face recognition)
- Xe tự lái (autonomous driving)
- Chẩn đoán hình ảnh y tế (medical image diagnosis)
- Phát hiện và phân loại vật thể (object detection & classification)
- Và nhiều lĩnh vực khác liên quan đến xử lý ảnh.

2.3 MẠNG HỒI TIẾP (RNN, LSTM)

Mạng hồi tiếp – Recurrent Neural Network (RNN) – là một loại mạng nơ ron sâu được thiết kế chuyên biệt để xử lý dữ liệu tuần tự (sequential data) như văn bản, chuỗi âm thanh, dữ liệu cảm biến theo thời gian, dữ liệu tài chính,... Không giống như các mạng nơ ron truyền thống như ANN hay CNN – vốn xử lý đầu vào theo từng điểm độc lập – RNN có khả năng ghi nhớ thông tin từ quá khứ nhờ vào cấu trúc mạng chứa

các vòng lặp phản hồi (recurrent loop), từ đó hỗ trợ mạnh mẽ cho việc học ngữ cảnh trong chuỗi.

2.3.1 RNN cơ bản

Recurrent Neural Network (RNN) là một kiến trúc mạng nơ ron được thiết kế để xử lý chuỗi dữ liệu có mối liên hệ theo thời gian, chẳng hạn như chuỗi văn bản, chuỗi âm thanh, dữ liệu cảm biến theo thời gian thực, dữ liệu thời tiết,... RNN cho phép thông tin từ bước thời gian trước được truyền sang bước hiện tại thông qua trạng thái ẩn (hidden state), tạo nên cơ chế ghi nhớ nội tại của mạng.

Cơ chế hoạt động

Ở mỗi thời điểm t, RNN nhận một đầu vào x_t và kết hợp nó với trạng thái ẩn từ thời điểm trước đó h_{t-1} để tạo ra trạng thái ẩn mới ht:

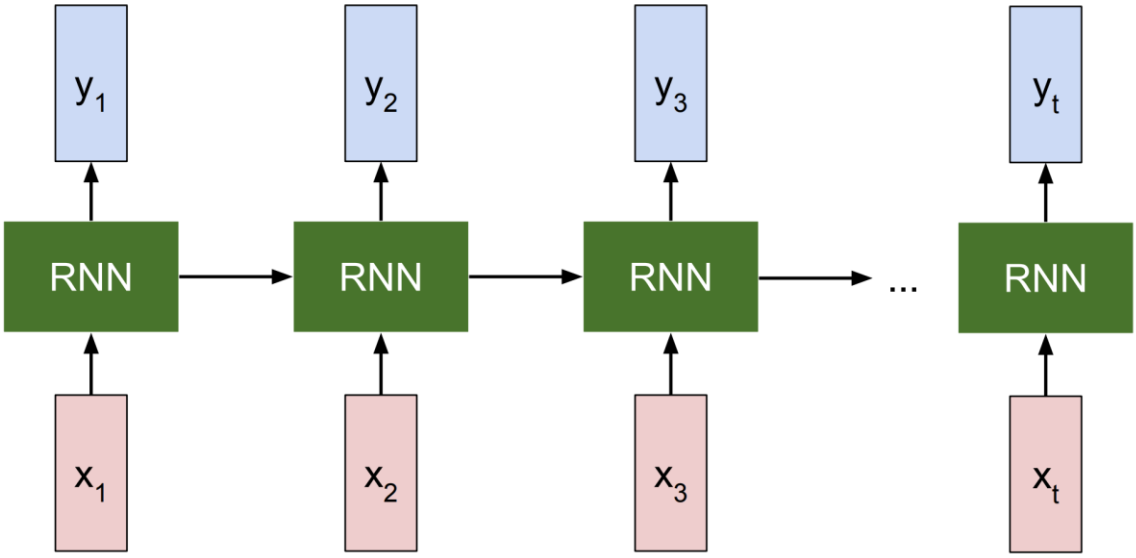
$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b)$$

Trong đó:

- x_t: Đầu vào tại thời điểm t
- h_t: Trạng thái ẩn tại thời điểm t
- W_{xh}: Trọng số kết nối từ đầu vào đến trạng thái ẩn
- W_{hh}: Trọng số kết nối từ trạng thái ẩn trước đến trạng thái hiện tại
- b: Hệ số điều chỉnh (bias)
- ϕ: Hàm kích hoạt, thường là tanh hoặc ReLU

Đầu ra tại mỗi bước có thể được tính như sau:

$$yt = \sigma(W_{hy}h_t + c)$$



Hình 2.7: Hình ảnh về kiến trúc RNN

Khả năng ghi nhớ ngữ cảnh

Nhờ việc liên tục cập nhật trạng thái ẩn, RNN có thể "ghi nhớ" thông tin từ các bước trước đó trong chuỗi. Điều này rất quan trọng trong các ứng dụng cần hiểu ngữ cảnh như:

- Dịch máy
- Nhận dạng giọng nói
- Dự đoán từ tiếp theo trong câu
- Phân tích cảm xúc

Hạn chế của RNN cơ bản

Tuy cơ chế ghi nhớ là điểm mạnh, nhưng RNN truyền thống vẫn gặp khó khăn khi xử lý các chuỗi dài do các vấn đề sau:

- Vấn đề biến mất gradient (Vanishing Gradient)
- Vấn đề bùng nổ gradient (Exploding Gradient)

Một số giải pháp cải thiện RNN cơ bản

- Gradient clipping: Giới hạn độ lớn của gradient để tránh bùng nổ.

- Sử dụng kiến trúc nâng cao như LSTM hoặc GRU: Giải quyết hiệu quả cả hai vấn đề nêu trên bằng cơ chế điều khiển dòng thông tin thông qua các cổng (gates).
- Khởi tạo trọng số hợp lý và sử dụng batch normalization.

2.3.2 LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) là một kiến trúc mạng nơ ron hồi tiếp đặc biệt, được giới thiệu bởi Hochreiter & Schmidhuber vào năm 1997. LSTM được thiết kế để khắc phục những hạn chế của RNN truyền thống, đặc biệt là vấn đề biến mất và bùng nổ gradient khi xử lý các chuỗi dài.

Điểm nổi bật của LSTM là khả năng lưu trữ và kiểm soát luồng thông tin qua thời gian nhờ các "cổng" thông minh, cho phép mạng:

- Duy trì thông tin quan trọng trong thời gian dài (long-term memory).
- Bỏ qua những thông tin không cần thiết tại từng thời điểm cụ thể.

Cấu trúc chính của một LSTM Cell

Một cell LSTM tại thời điểm t bao gồm 3 cổng điều khiển chính và một trạng thái nhớ (cell state) C_t:

Cổng quên (Forget Gate)

Quyết định thông tin nào từ trạng thái nhớ cũ C_{t-1} sẽ bị loại bỏ:

$$f_t = \sigma(W_f \, . \, [h_{t-1}, x_t] + b_f)$$

- f_t: vector chứa các giá trị từ 0 đến 1, xác định mức độ "quên" từng phần của thông tin cũ.
- σ: hàm sigmoid

Cổng đầu vào (Input Gate)

Quyết định thông tin mới nào sẽ được thêm vào trạng thái nhớ:

$$i_t = \sigma(W_i \, . \, [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- i_t: xác định vùng được phép cập nhật
- $\tilde{C}_t$: giá trị ứng viên để thêm vào bộ nhớ

Cập nhật trạng thái nhớ

Kết hợp thông tin cần quên và thông tin mới:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

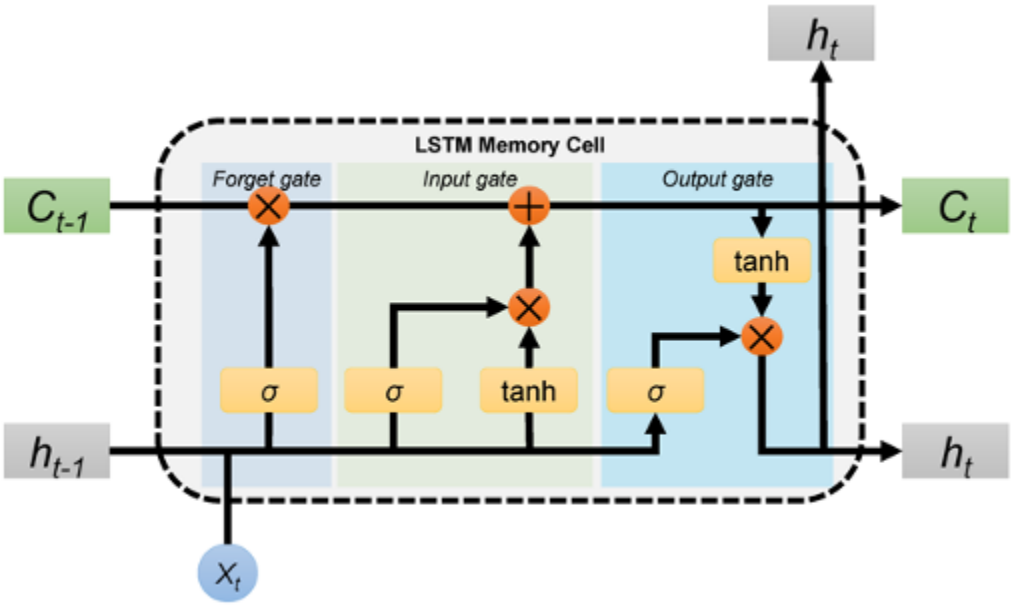
- $\odot$: phép nhân từng phần tử (Hadamard product)

Cổng đầu ra (Output Gate)

Quyết định phần nào của trạng thái nhớ được dùng làm đầu ra:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

- h_t: đầu ra tại thời điểm ttt, đồng thời là trạng thái ẩn truyền sang bước tiếp theo.



Hình 2.8: Hình ảnh về cấu trúc chính của LSTM

Ưu điểm của LSTM

- Ghi nhớ lâu dài: Cơ chế lưu trữ thông tin lâu trong cell state giúp mô hình học được các phụ thuộc dài hạn trong chuỗi.
- Khắc phục biến mất gradient: Do dòng thông tin trong trạng thái nhớ có thể được giữ gần như không thay đổi nếu cần.
- Huấn luyện ổn định hơn RNN thường.

Ứng dụng của LSTM

LSTM rất phù hợp cho các bài toán xử lý chuỗi có tính ngữ cảnh cao, tiêu biểu như:

- Dự đoán chuỗi thời gian: giá cổ phiếu, nhiệt độ, tiêu thụ năng lượng,...
- Xử lý ngôn ngữ tự nhiên (NLP): phân tích cảm xúc, dịch máy, sinh văn bản.
- Nhận dạng giọng nói và âm thanh.
- Phân tích video: nhận diện hành động, theo dõi đối tượng,...
- Sinh nhạc, tổng hợp giọng nói, và các ứng dụng AI sáng tạo khác.

2.4 CÁC KỸ THUẬT HUẤN LUYỆN VÀ TỐI ƯU HÓA

Quá trình huấn luyện và tối ưu hóa đóng vai trò cốt lõi trong việc đảm bảo mạng nơ ron học được các đặc trưng quan trọng từ dữ liệu, từ đó đưa ra dự đoán chính xác và hiệu quả. Mục tiêu chính là tìm ra bộ trọng số (weights) tối ưu sao cho giá trị của hàm mất mát (loss function) được giảm thiểu trên tập dữ liệu huấn luyện, đồng thời vẫn giữ được khả năng tổng quát hóa tốt trên dữ liệu chưa thấy.

Để đạt được điều đó, các kỹ thuật huấn luyện và tối ưu hóa hiện đại không chỉ giúp mô hình hội tụ nhanh chóng, ổn định mà còn làm giảm nguy cơ quá khớp (overfitting). Dưới đây là những kỹ thuật tiêu biểu thường được sử dụng trong thực tiễn:

- Xây dựng hàm mất mát phù hợp với từng loại bài toán (phân loại, hồi quy,...).
- Tối ưu hóa trọng số thông qua các thuật toán như Gradient Descent, Adam,...
- Chính quy hóa và giảm nhiễu, ví dụ bằng Dropout hoặc Regularization.
- Chuẩn hóa dữ liệu trung gian để tăng tốc huấn luyện như Batch Normalization.
- Điều chỉnh tốc độ học hợp lý, tránh trường hợp mô hình học quá chậm hoặc dao động.

- Tăng cường dữ liệu (Data Augmentation) giúp mô hình tiếp xúc với nhiều biến thể hơn của dữ liệu đầu vào.
- Sử dụng kỹ thuật dừng sớm (Early Stopping) để ngăn ngừa học quá mức khi độ lỗi trên tập validation bắt đầu tăng.

Những chiến lược này, khi được kết hợp một cách hợp lý, có thể tạo ra những mô hình mạng nơ ron vừa mạnh mẽ vừa bền vững, đáp ứng được yêu cầu của các bài toán phức tạp trong học sâu hiện đại.

2.4.1 Lan truyền ngược và Gradient Descent

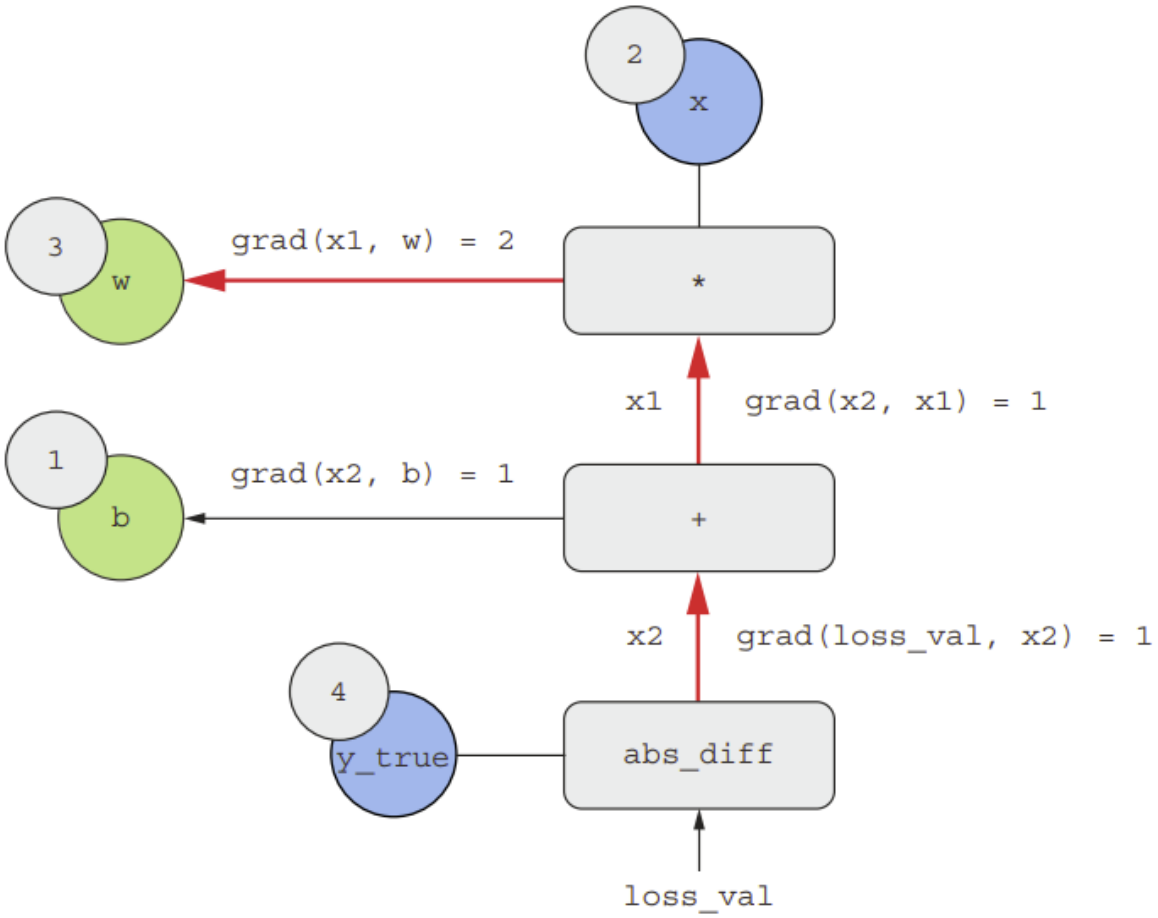
Lan truyền ngược (Backpropagation) là thuật toán cốt lõi giúp huấn luyện mạng nơ ron. Quá trình này sử dụng quy tắc chuỗi trong vi phân để tính đạo hàm của hàm mất mát đối với từng trọng số trong mạng. Sau khi tính toán được gradient (đạo hàm riêng) của mỗi tham số, ta sử dụng thông tin này để cập nhật trọng số theo hướng làm giảm hàm mất mát.

Cụ thể, quá trình lan truyền ngược gồm ba bước chính:

B1. Truyền xuôi (Forward pass): Tính toán đầu ra của mạng từ dữ liệu đầu vào.

B2. Tính lỗi (Loss computation): So sánh đầu ra dự đoán với giá trị thực để tính hàm mất mát.

B3. Truyền ngược (Backward pass): Sử dụng đạo hàm để lan truyền lỗi từ đầu ra về các lớp trước đó, từ đó tính gradient đối với từng trọng số.



Hình 2.9: Hình ảnh lan truyền ngược

Gradient Descent là thuật toán tối ưu phổ biến nhất được dùng để cập nhật trọng số dựa trên gradient tính được từ lan truyền ngược. Trọng số được cập nhật theo công thức:

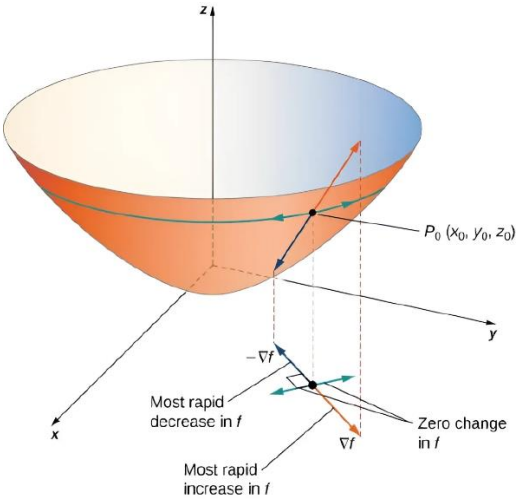
$$w := w - \eta . \frac{\partial L}{\partial w}$$

Trong đó:

- w là trọng số cần cập nhật,
- η là hệ số học (learning rate), quyết định độ lớn bước nhảy trong mỗi lần cập nhật,
- $\frac{\partial L}{\partial w}$ là đạo hàm của hàm mất mát L theo trọng số w.

Tùy vào cách tính và cập nhật gradient, có nhiều biến thể của Gradient Descent như:

- **Batch Gradient Descent:** Dùng toàn bộ dữ liệu huấn luyện để tính gradient, ổn định nhưng chậm.
- **Stochastic Gradient Descent (SGD):** Cập nhật trọng số sau mỗi mẫu dữ liệu, nhanh nhưng dao động mạnh.
- **Mini-Batch Gradient Descent:** Thỏa hiệp giữa hai cách trên, vừa nhanh vừa ổn định hơn.



Hình 2.10: Gradient cho biết giá trị lớn nhất và bé nhất của đạo hàm có hướng tại một điểm

Thuật toán Gradient Descent đóng vai trò trung tâm trong quá trình huấn luyện, nhưng hiệu quả tối ưu phụ thuộc rất nhiều vào cách chọn learning rate, kỹ thuật khởi tạo trọng số, và cấu trúc mô hình.

2.4.2 Các biến thể của Gradient Descent

Trong thực tế, việc tối ưu mạng nơ ron không chỉ dừng lại ở Gradient Descent truyền thống, mà thường sử dụng các biến thể cải tiến nhằm tăng tốc độ hội tụ, giảm dao động và cải thiện hiệu suất huấn luyện trên dữ liệu lớn. Một số biến thể phổ biến bao gồm:

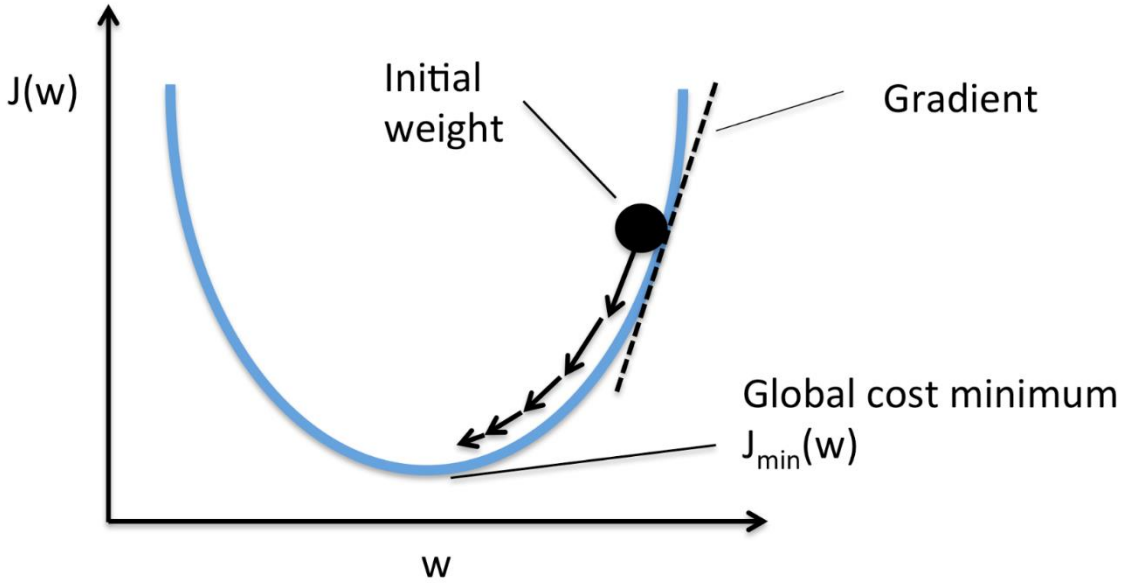
Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent là phiên bản đơn giản và nhanh của thuật toán Gradient Descent, trong đó cập nhật trọng số được thực hiện sau mỗi mẫu dữ liệu đầu vào (thay vì toàn bộ tập huấn luyện). Cập nhật trọng số như sau:

$$w := w - \eta . \frac{\partial L(xi)}{\partial w}$$

Trong đó x_i là một mẫu dữ liệu duy nhất.

- Ưu điểm: Tốc độ huấn luyện nhanh hơn, cập nhật trọng số liên tục, phù hợp với các tập dữ liệu lớn.
- Nhược điểm: Gradient dao động mạnh, khó hội tụ về điểm tối ưu ổn định nếu không có điều chỉnh phù hợp.



Hình 2.11: Hình ảnh Stochastic Gradient Descent (SGD)

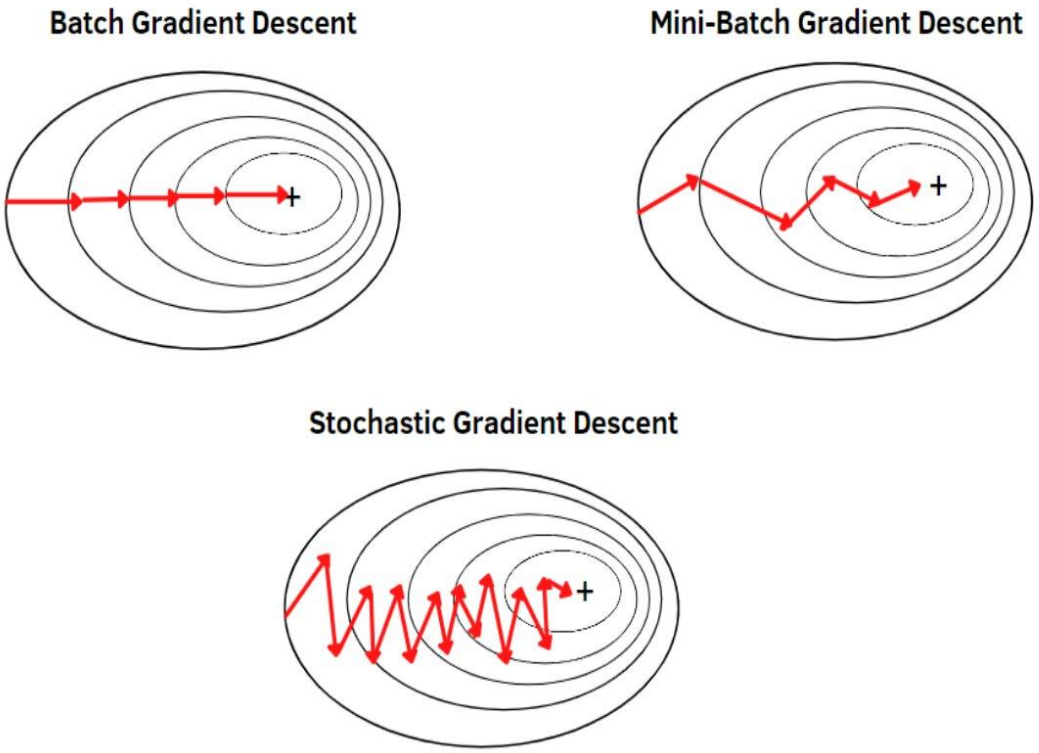
Mini-batch Gradient Descent

Thay vì sử dụng toàn bộ tập huấn luyện (Batch GD) hoặc chỉ một mẫu (SGD), Mini-batch GD chia tập dữ liệu thành các nhóm nhỏ (mini-batch), thường có kích thước từ 32 đến 256 mẫu/lô. Gradient được tính trên từng mini-batch:

$$w := w - \eta . \frac{1}{m} \sum \frac{\partial L(xi)}{\partial w}$$

Trong đó m là số mẫu trong một mini-batch.

- Ưu điểm: Cân bằng giữa độ chính xác và tốc độ cập nhật, tận dụng được ưu điểm của cả SGD và Batch GD.
- Phổ biến: Là lựa chọn tiêu chuẩn trong huấn luyện deep learning hiện nay.



Hình 2.12: So sánh Batch, Mini-Batch và Stochastic

Adam (Adaptive Moment Estimation)

Adam là một trong những thuật toán tối ưu hiện đại và hiệu quả nhất, được sử dụng rộng rãi trong deep learning. Adam kết hợp giữa:

- **Momentum:** Ghi nhớ gradient trước đó (trung bình động bậc nhất),
- **RMSprop:** Chuẩn hóa gradient theo độ lớn bình phương (trung bình động bậc hai).

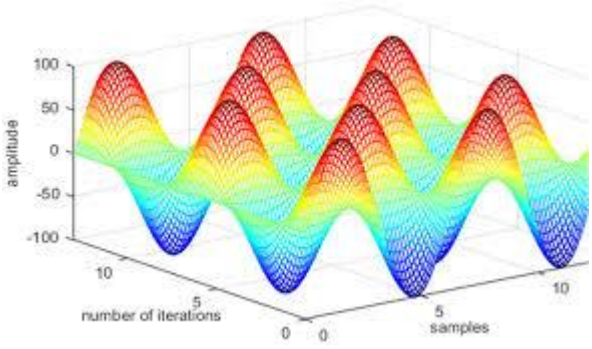
Công thức cập nhật trọng số (đơn giản hóa):

$$w := w - \eta . \frac{mt}{\epsilon + \sqrt{vt}}$$

Trong đó:

- mt: Ước lượng trung bình động có điều chỉnh của gradient,

- vt: Ước lượng trung bình động có điều chỉnh của bình phương gradient,
- ϵ: Số rất nhỏ để tránh chia cho 0.
- Ưu điểm: Tự động điều chỉnh learning rate theo từng tham số, hiệu quả tốt với mạng sâu và dữ liệu nhiễu.
- Ứng dụng: Hầu hết các mô hình học sâu hiện nay sử dụng Adam như một lựa chọn mặc định.



Hình 2.13: Hình ảnh của Adam

Ngoài Adam, còn nhiều thuật toán tối ưu khác như Adagrad, RMSprop, Nadam,... tùy vào bài toán cụ thể và dữ liệu mà ta lựa chọn thuật toán phù hợp để đạt kết quả tốt nhất.

2.4.3 Kỹ thuật tránh overfitting

- **Dropout:** Vô hiệu hóa ngẫu nhiên một số nơ ron trong quá trình huấn luyện, giúp ngăn ngừa overfitting và cải thiện khả năng tổng quát của mô hình.
- **Early stopping:** Dừng huấn luyện khi độ chính xác trên tập validation không cải thiện, giúp tránh việc huấn luyện quá lâu và làm cho mô hình bị overfitting.
- **Regularization (L1, L2):** Phạt các trọng số lớn để mô hình không quá phức tạp, giúp cải thiện khả năng tổng quát của mô hình.

2.4.4. Các hàm mất mát thường dùng

Hàm mất mát (Loss Function) là thành phần cốt lõi trong quá trình huấn luyện mạng nơ ron. Nó định lượng mức độ sai lệch giữa đầu ra dự đoán của mô hình và giá trị thực tế, từ đó làm cơ sở cho quá trình cập nhật trọng số qua lan truyền ngược

(backpropagation). Việc lựa chọn hàm mất mát phù hợp tùy thuộc vào loại bài toán (phân loại hay hồi quy).

Cross-Entropy Loss (Entropy chéo)

Đây là hàm mất mát phổ biến nhất cho các bài toán phân loại, đặc biệt là:

- **Phân loại nhị phân** (binary classification),
- **Phân loại đa lớp** (multi-class classification).

Công thức (cho phân loại nhị phân):

$$L = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

Trong đó:

- y là nhãn thực tế (0 hoặc 1),
- $\hat{y}$ là xác suất dự đoán của mô hình.

Mean Squared Error (MSE - Sai số bình phương trung bình)

Là hàm mất mát được sử dụng phổ biến trong bài toán hồi quy, nơi đầu ra là giá trị thực (liên tục), không phải nhãn rời rạc.

Công thức:

$$L = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

Trong đó:

- y_i: giá trị thực tế của mẫu thứ i,
- $\hat{y}_i$: giá trị dự đoán,
- n: số lượng mẫu.

CHƯƠNG 3 : THIẾT KẾ KIẾN TRÚC MẠNG

3.1 YÊU CẦU THIẾT KẾ VÀ TẬP DỮ LIỆU

Tập dữ liệu được sử dụng trong đề tài này bao gồm các hình ảnh biểu diễn miền thời gian – tần số của các tín hiệu điều chế trong hệ thống truyền thông và radar. Các hình ảnh này được tạo ra bằng cách chuyển đổi tín hiệu gốc sang miền thời gian – tần số thông qua phân phối Wigner-Ville giả (pseudo Wigner-Ville distribution). Đây là một biến thể của phân phối Wigner-Ville truyền thống, trong đó kỹ thuật smoothing được áp dụng để làm giảm nhiễu và tạo ra biểu diễn mượt mà, dễ phân tích hơn trong miền thời gian – tần số.

Tập dữ liệu bao gồm hình ảnh của 8 loại tín hiệu điều chế khác nhau, được chia thành hai nhóm:

- Tín hiệu điều chế sử dụng trong radar:
    - Rect: Xung chữ nhật
    - LFM: Điều chế tần số tuyến tính (Linear Frequency Modulation)
    - Barker: Mã Barker
- Tín hiệu điều chế sử dụng trong truyền thông:
    - GFSK: Điều chế khóa dịch tần số Gaussian
    - CPFSK: Điều chế khóa dịch tần số pha liên tục
    - B-FM: Điều chế tần số (Broadcast FM)
    - DSB-AM: Điều chế biên độ song biên
    - SSB-AM: Điều chế biên độ đơn biên

Tập dữ liệu có :

- Tập huấn luyện (training set): 6.400 ảnh tín hiệu

Việc sử dụng các biểu diễn hình ảnh giúp mô hình học sâu có thể trích xuất đặc trưng trực quan và phức tạp từ tín hiệu, thay vì xử lý tín hiệu 1 chiều dạng sóng thông thường. Điều này tạo tiền đề cho việc áp dụng các kiến trúc mạng nơ-ron tích chập (CNN) trong bài toán phân loại tín hiệu điều chế.

3.1.1 Kiến trúc tổng quát

Trong đề tài này, mô hình học sâu được thiết kế nhằm phân loại tín hiệu radar và truyền thông thông qua ảnh thời gian – tần số. Kiến trúc mạng được xây dựng theo hướng gọn nhẹ, hiệu quả tính toán nhưng vẫn đảm bảo khả năng học đặc trưng mạnh mẽ từ dữ liệu đầu vào. Cụ thể, mô hình sử dụng kiến trúc EfficientMiniNet, một biến thể tinh gọn của mạng nơ-ron tích chập, tích hợp các khối Depthwise Separable Convolution và Residual Connection.

Cấu trúc tổng thể của mô hình gồm 3 thành phần chính:

- Khối tiền xử lý (StemBlock): Bao gồm một lớp tích chập chuẩn (Conv2d) với 32 kênh đầu ra, kích thước kernel là 3x3, bước sải (stride) là 2 để giảm kích thước ảnh đầu vào. Sau lớp này là BatchNorm và hàm kích hoạt ReLU.
- Chuỗi các khối Depthwise Separable Convolution: Đây là thành phần cốt lõi của mạng. Mỗi khối gồm:
    - Lớp tích chập depthwise: hoạt động độc lập trên từng kênh đầu vào.
    - Lớp tích chập pointwise: kết hợp thông tin giữa các kênh.
    - Lớp chuẩn hóa BatchNorm và hàm kích hoạt ReLU.
    - Nếu điều kiện phù hợp (cùng số kênh đầu vào và ra, stride = 1), mô hình áp dụng kết nối tàn dư (residual) để tăng khả năng truyền tín hiệu ngược.

Mô hình gồm 6 khối:

- block1: 32 → 64, stride = 1
- block2: 64 → 128, stride = 2

- block3: 128 → 128, stride = 1
- block4: 128 → 256, stride = 2
- block5: 256 → 256, stride = 1
- block6: 256 → 512, stride = 2

Phân loại cuối (Classification Head):

- Lớp gộp trung bình thích nghi (AdaptiveAvgPool2d) đưa đặc trưng về kích thước 1x1.
- Kết quả sau đó được flatten và đưa qua một lớp Linear đầu ra với số lượng lớp bằng số tín hiệu cần phân loại (8 lớp tương ứng 8 loại tín hiệu).
- Mô hình sử dụng CrossEntropyLoss và Softmax ngầm ở đầu ra trong quá trình huấn luyện.

Đặc điểm nổi bật của kiến trúc:

- Hiệu quả tính toán cao: Depthwise Separable Conv giúp giảm đáng kể số lượng tham số và phép nhân chập so với Conv2d thông thường.
- Khả năng học mạnh mẽ nhờ residual connections giúp tránh mất mát gradient khi mạng sâu.
- Thích hợp cho dữ liệu hình ảnh tín hiệu, đặc biệt là trong các hệ thống nhúng hoặc môi trường có tài nguyên hạn chế.

3.1.2 Lớp tích chập

Lớp tích chập (Convolution Layer) là thành phần cốt lõi trong mạng nơ-ron tích chập (CNN), giúp mô hình trích xuất các đặc trưng không gian từ ảnh đầu vào như cạnh, góc, họa tiết… bằng cách sử dụng các bộ lọc (filter hoặc kernel). Trong mô hình EfficientMiniNet, lớp tích chập được thiết kế đặc biệt dưới dạng tích chập phân tách chiều sâu (Depthwise Separable Convolution), giúp giảm đáng kể số tham số và tăng hiệu suất.

a. Cơ chế hoạt động của lớp tích chập

Lớp tích chập thực hiện phép toán giữa ma trận ảnh đầu vào và các kernel bằng cách trượt kernel trên ảnh và tính tích có hướng (dot product) tại từng vùng tương ứng. Mỗi kernel cho ra một ma trận đầu ra (feature map). Nhiều kernel sẽ cho nhiều feature map, tạo nên chiều sâu (depth) của đặc trưng.

Để tránh mất mát thông tin ở biên ảnh, ảnh đầu vào thường được zero-padding, tức là gán thêm các hàng và cột có giá trị 0 ở biên, giúp đầu ra giữ nguyên kích thước. Nếu không padding, ảnh đầu ra sau tích chập sẽ bị giảm kích thước.

Kích thước ma trận đầu ra được xác định bằng công thức:

$$L' = \frac{L - K + 2P}{S} + 1$$

Trong đó:

- L: chiều dài hoặc chiều rộng ảnh đầu vào
- K: kích thước kernel
- P: số pixel padding
- S: bước trượt (stride)
- L′: kích thước đầu ra sau tích chập

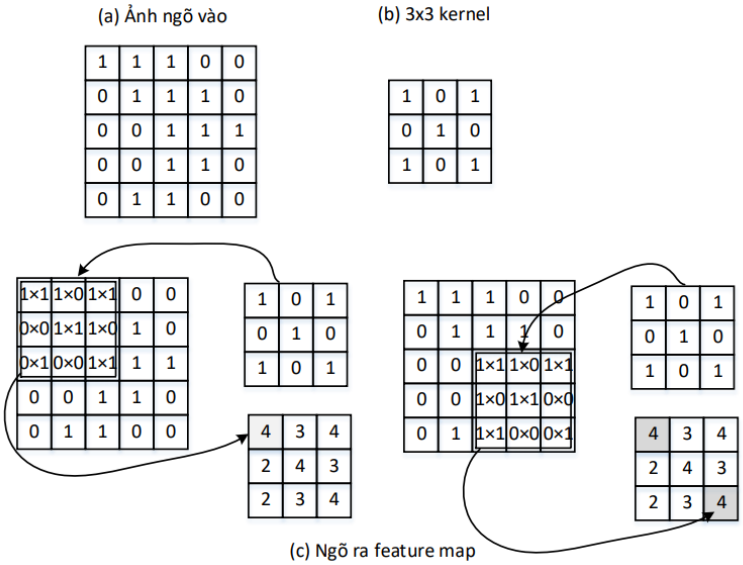
b. Các thông số quan trọng

- Depth (độ sâu): Số lượng kernel, tương ứng với số lượng feature map đầu ra. Trong EfficientMiniNet, độ sâu được điều chỉnh linh hoạt qua các tầng chập để tăng khả năng học các đặc trưng trừu tượng hơn.
- Filter size (kích thước bộ lọc): Thường dùng kích thước 3×3 để cân bằng giữa hiệu quả tính toán và khả năng học đặc trưng. Một số lớp có thể dùng 1×1 để thực hiện giảm chiều (channel reduction).
- Stride (bước trượt): Thường chọn là 1 để giữ thông tin đầy đủ. Nếu stride = 2, ảnh sẽ giảm một nửa kích thước sau mỗi lớp, giúp giảm độ phức tạp tính toán.
- Zero-padding: Padding giúp giữ nguyên kích thước không gian của đầu vào. Nếu không padding, các đặc trưng ở rìa ảnh dễ bị mất trong quá trình học.

- Activation function: Sau mỗi lớp chập thường sử dụng hàm kích hoạt phi tuyến như ReLU nhằm tăng khả năng biểu diễn của mô hình.

c. Ví dụ minh họa

Giả sử ảnh đầu vào là ma trận 5×5 kernel có kích thước 3×3, không padding và stride = 1. Khi trượt kernel qua các vùng ảnh, ta thực hiện phép nhân từng phần tử tương ứng và cộng tổng để tạo ra từng phần tử của feature map. Nếu sử dụng 3 kernel khác nhau, ta sẽ tạo ra 3 feature map.



Hình 3.1: Mô tả tích chập ảnh với bộ lọc (kernel)

d. Tích chập trong EfficientMiniNet

Trong mô hình EfficientMiniNet, tích chập được chia thành 2 giai đoạn:

- Depthwise convolution: Mỗi kênh đầu vào được xử lý riêng biệt bằng một kernel 3×3, giữ nguyên số kênh.
- Pointwise convolution (1x1 conv): Các kênh được trộn lại thông qua kernel 1×1 nhằm kết hợp đặc trưng liên kênh và thay đổi số lượng kênh đầu ra.

Ưu điểm của tích chập phân tách chiều sâu là giảm mạnh số tham số và phép tính so với tích chập truyền thống:

Giảm tham số từ $D_K \times D_K \times M \times N \Rightarrow D_K \times D_K \times M + M \times N$

Trong đó:

- D_K: kích thước kernel
- M: số kênh đầu vào
- N: số kênh đầu ra

Việc sử dụng lớp tích chập tối ưu như vậy giúp mô hình hoạt động hiệu quả ngay cả khi triển khai trên các thiết bị biên (edge devices) với tài nguyên hạn chế.

3.1.3 Lớp kích hoạt

Sau khi ma trận đặc trưng (feature map) được tạo ra bởi các lớp tích chập, chúng sẽ chứa các giá trị âm, dương hoặc bằng 0. Để tăng khả năng học phi tuyến và loại bỏ các giá trị không cần thiết, các giá trị này được đưa qua hàm kích hoạt phi tuyến tính (activation function). Lớp kích hoạt giúp mạng nơ-ron có khả năng học và mô phỏng các quan hệ phức tạp hơn giữa đầu vào và đầu ra.

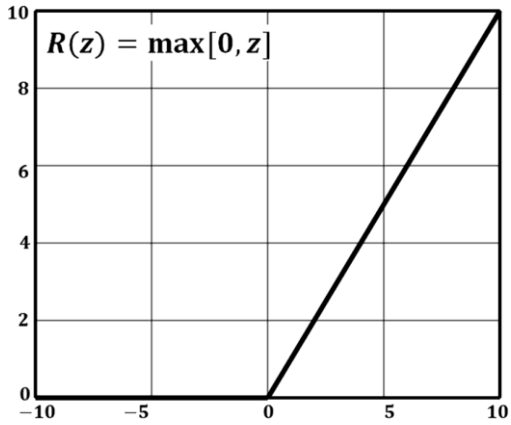
a. Hàm ReLU (Rectified Linear Unit)

Trong kiến trúc EfficientMiniNet, hàm ReLU là hàm kích hoạt chính được sử dụng sau mỗi lớp chập hoặc lớp kết nối đầy đủ. ReLU là một hàm đơn giản nhưng rất hiệu quả, đặc biệt trong mạng tích chập vì:

- Giảm khả năng xảy ra hiện tượng mất gradient (vanishing gradient)
- Tăng tốc độ huấn luyện
- Không làm tăng độ phức tạp tính toán

Hàm ReLU được định nghĩa như sau:

$$f(x) = \begin{cases} 0 & \text{nếu } x < 0 \\ x & \text{nếu } x \geq 0 \end{cases}$$



Hình 3.2: Đồ thị hàm đơn vị tuyến tính được chỉnh lưu (ReLU)

Đồ thị của hàm ReLU thể hiện rõ khả năng lọc bỏ tất cả giá trị âm về 0, chỉ giữ lại các giá trị dương. Điều này đồng nghĩa với việc chỉ những đặc trưng quan trọng và có ảnh hưởng tích cực mới được giữ lại sau mỗi lớp tích chập.

b. Ưu điểm của ReLU

- Tính toán đơn giản, dễ triển khai
- Không làm thay đổi kích thước ma trận
- Tăng khả năng lan truyền gradient khi huấn luyện
- Giúp mạng hội tụ nhanh hơn

Tuy nhiên, ReLU cũng có một số hạn chế, như hiện tượng "dead neurons" khi quá nhiều giá trị âm bị triệt tiêu. Để khắc phục, các biến thể như LeakyReLU, ELU, GELU có thể được dùng thay thế trong các kiến trúc phức tạp hơn. Dù vậy, với mô hình gọn nhẹ như EfficientMiniNet, ReLU là lựa chọn phù hợp và hiệu quả.

3.1.4 Lớp gộp

Lớp gộp (Pooling) là một trong ba lớp cơ bản trong mạng nơ-ron tích chập (CNN), giúp giảm kích thước của các ma trận đặc trưng (feature maps) sau lớp tích chập và hàm kích hoạt. Lớp này không chỉ giúp giảm thiểu số lượng tham số mà còn giúp mạng trở nên bất biến với sự dịch chuyển của dữ liệu đầu vào, từ đó cải thiện khả năng tổng quát của mô hình.

a. Mục đích và vai trò của lớp gộp

Lớp gộp có nhiệm vụ làm giảm kích thước (tạo ma trận nhỏ hơn) và tổng hợp thông tin từ các đặc trưng của ảnh đầu vào. Các đặc trưng này sẽ được thu hẹp lại qua việc gộp các phần tử trong các cửa sổ con của ma trận đầu vào. Lớp gộp giúp:

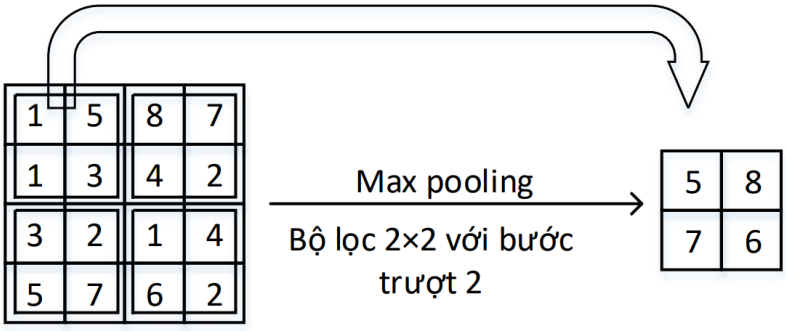
- Giảm độ phức tạp tính toán của mạng nơ-ron.
- Tạo tính bất biến đối với sự dịch chuyển của các đặc trưng trong ảnh.
- Đảm bảo các thông tin quan trọng nhất được duy trì, trong khi các thông tin chi tiết không quan trọng bị loại bỏ.

b. Các phương pháp gộp phổ biến

Trong các mạng nơ-ron tích chập, có nhiều phương pháp pooling khác nhau, nhưng hai phương pháp phổ biến nhất là Max Pooling và Average Pooling.

- Max Pooling là phương pháp gộp phổ biến trong các mạng nơ-ron tích chập. Trong Max Pooling, một cửa sổ di động được áp dụng trên ma trận đặc trưng (feature map), và trong mỗi cửa sổ, giá trị lớn nhất được chọn làm đại diện cho cửa sổ đó.

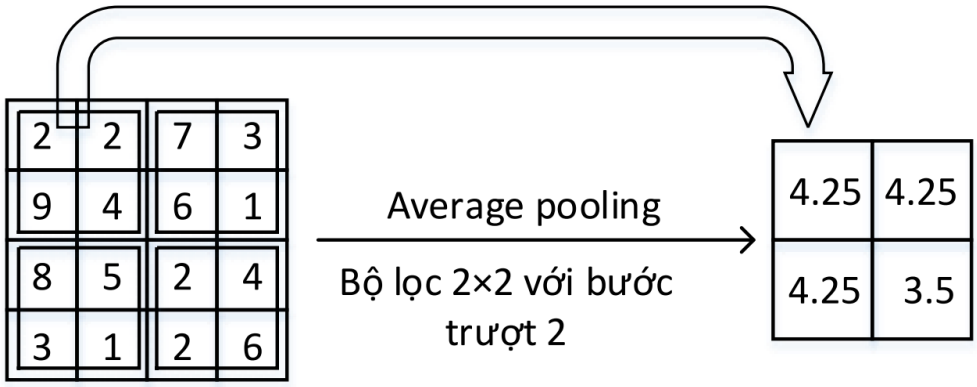
- Cửa sổ (Kernel): Là một vùng con của ma trận có kích thước cố định (ví dụ 2x2, 3x3).
- Bước trượt (Stride): Là bước di chuyển của cửa sổ. Thông thường, bước trượt được chọn bằng kích thước của cửa sổ, dẫn đến không có sự trùng lặp giữa các vùng con được lựa chọn.
- Ví dụ, nếu sử dụng cửa sổ 2x2 và bước trượt 2, ta sẽ đi qua từng phần của ma trận và chọn giá trị lớn nhất trong mỗi cửa sổ. Quá trình này giúp loại bỏ những chi tiết không cần thiết và chỉ giữ lại các đặc trưng mạnh mẽ nhất.



Hình 3.3: Hình mô tả hoạt động của Max Pooling

- Trong mạng EfficientMiniNet, lớp pooling thường được áp dụng sau mỗi lớp tích chập để giảm kích thước của ma trận đặc trưng và giúp mạng học được các đặc trưng mạnh mẽ từ ảnh.

- Average Pooling là phương pháp gộp khác, trong đó, thay vì chọn giá trị lớn nhất, giá trị trung bình của các phần tử trong cửa sổ được chọn làm đại diện cho cửa sổ đó. Phương pháp này giúp bảo tồn thông tin tổng quan hơn và có thể sử dụng trong những trường hợp không cần tập trung vào các đặc trưng cực đoan như trong Max Pooling.



Hình 3.4: Hình mô tả hoạt động của Average Pooling

Cả Max Pooling và Average Pooling đều giúp giảm thiểu kích thước của ma trận đặc trưng, nhưng Max Pooling thường được ưa chuộng hơn trong các mạng nhận dạng ảnh, vì nó giúp giữ lại các đặc trưng mạnh mẽ nhất.

c. Tính bất biến đối với sự dịch chuyển

Một trong những lợi ích quan trọng của lớp gộp là tính bất biến đối với sự dịch chuyển. Điều này có nghĩa là nếu ta dịch chuyển một chút ảnh đầu vào, kết quả của lớp gộp sẽ gần như không thay đổi. Ví dụ, khi chúng ta xác định sự tồn tại của một đối tượng (ví dụ khuôn mặt), chúng ta không cần biết chính xác đối tượng ở đâu trong ảnh, mà chỉ cần biết là nó có tồn tại hay không. Do đó, Max Pooling và Average Pooling giúp mạng nơ-ron học các đặc trưng quan trọng mà không bị phụ thuộc vào vị trí chính xác của các đối tượng trong ảnh.

3.1.5 Lớp kết nối đầy đủ

Lớp kết nối đầy đủ (Fully Connected Layer) là lớp cuối cùng trong các mạng nơ-ron tích chập (CNN), giúp kết nối các đặc trưng học được từ các lớp trước đó để đưa ra quyết định cuối cùng. Trong lớp này, tất cả các nơ-ron đều được kết nối với nhau, tạo ra một mạng nơ-ron nhiều lớp, nơi mỗi nơ-ron trong lớp hiện tại nhận giá trị từ tất cả các nơ-ron của lớp trước.

a. Mục đích và vai trò của lớp kết nối đầy đủ

Lớp kết nối đầy đủ có nhiệm vụ lấy đầu ra từ các lớp trước (các lớp tích chập và pooling) và chuyển đổi chúng thành một vector một chiều, từ đó đưa ra dự đoán cuối cùng của mạng. Các ma trận đặc trưng (feature maps) từ lớp pooling cuối cùng thường có kích thước lớn, và chúng cần phải được chuyển đổi thành các vector một chiều để đưa vào lớp kết nối đầy đủ.

Lớp kết nối đầy đủ:

- Kết nối tất cả các nơ-ron từ lớp trước với mỗi nơ-ron trong lớp hiện tại.
- Thực hiện phép tính giữa các trọng số và đầu vào để tạo ra đầu ra cho lớp tiếp theo.
- Thường được sử dụng với hàm kích hoạt phi tuyến tính (như ReLU) để giúp mạng học được các mô hình phức tạp.

b. Quá trình chuyển đổi từ ma trận feature map thành vector

Để đưa các ma trận đặc trưng (feature maps) từ lớp pooling vào lớp kết nối đầy đủ, ta cần chuyển đổi chúng thành các vector một chiều. Việc này thường được thực hiện thông qua một bước gọi là Flattening, trong đó các phần tử của ma trận sẽ được xếp thành một vector duy nhất.

Giả sử lớp pooling cuối cùng có ma trận kích thước H×W×D (với H là chiều cao, W là chiều rộng, và D là số lượng kênh), ta sẽ chuyển đổi ma trận này thành một vector có chiều dài H×W×D

c. Lớp kết nối đầy đủ với hàm kích hoạt Softmax

Sau khi các ma trận đã được chuyển thành vector, chúng sẽ được đưa vào lớp kết nối đầy đủ. Mỗi nơ-ron trong lớp này sẽ nhận trọng số và bias từ lớp trước đó, sau đó tính toán giá trị ngõ ra. Thông qua một phép toán tuyến tính, lớp kết nối đầy đủ sẽ cho ra một vector đầu ra, trong đó mỗi phần tử đại diện cho một lớp phân loại.

Lớp cuối cùng thường sử dụng hàm Softmax để chuyển đổi các giá trị đầu ra thành xác suất phân loại. Hàm Softmax sẽ chuyển đổi các giá trị đầu ra thành một dãy xác suất, với tổng các xác suất là 1. Hàm này đặc biệt hữu ích trong các bài toán phân loại đa lớp.

d. Mô hình hoàn chỉnh

Lớp kết nối đầy đủ là bước cuối cùng trong mạng nơ-ron tích chập, nơi các đặc trưng học được từ các lớp trước được kết hợp lại để đưa ra quyết định phân loại cuối cùng. Quá trình này giúp mạng phân loại hình ảnh dựa trên các đặc trưng đã được trích xuất và giảm dần thông tin từ ảnh đầu vào.

3.2 KIẾN TRÚC MẠNG CƠ SỞ

3.2.1 Lan truyền ngược qua lớp tích chập

Lan truyền ngược (Backpropagation) là một quá trình quan trọng trong việc huấn luyện các mạng nơ-ron sâu, đặc biệt là mạng nơ-ron tích chập (CNN). Quá trình này nhằm điều chỉnh trọng số của mạng sao cho sai số giữa đầu ra thực tế và đầu ra mong muốn được giảm thiểu. Việc lan truyền ngược qua lớp tích chập có nguyên lý tương

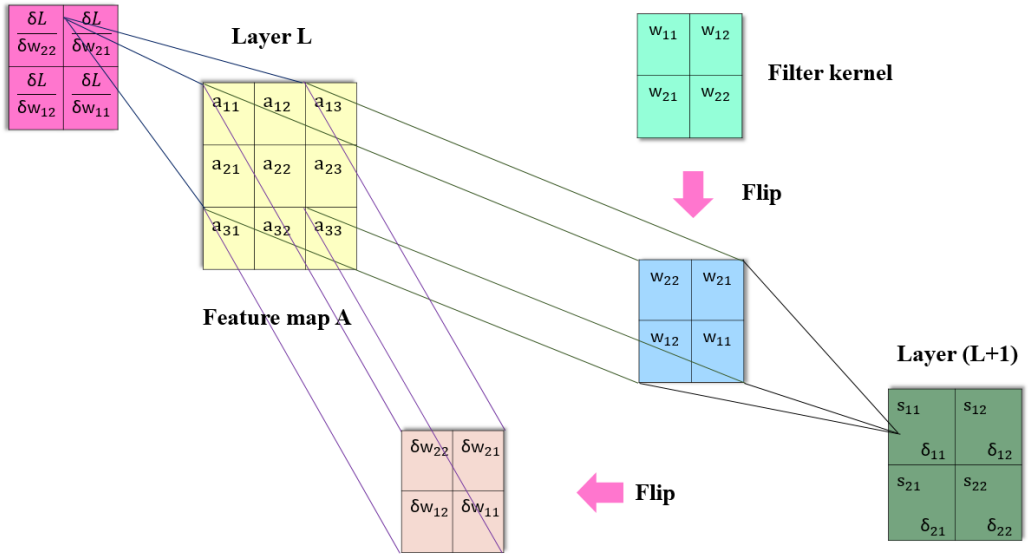
tự như trong mạng Perceptron nhiều lớp (MLP), nhưng có một số đặc điểm riêng biệt do tính chất chia sẻ trọng số của bộ lọc.

a. Khác biệt trong lan truyền ngược của lớp tích chập

Trong mạng tích chập, các bộ lọc (kernel) được chia sẻ cho toàn bộ không gian ảnh hoặc feature map. Điều này có nghĩa là cùng một bộ lọc sẽ được áp dụng nhiều lần tại các vùng khác nhau trên đầu vào (input), do đó khi tính đạo hàm (gradient) đối với trọng số của bộ lọc, ta cần tổng hợp ảnh hưởng từ tất cả các vị trí mà bộ lọc đã được áp dụng.

b. Nguyên lý lan truyền ngược qua lớp tích chập

Giả sử có một feature map đầu vào A tại lớp L, được tích chập với một bộ lọc K để tạo ra feature map đầu ra B tại lớp L+1. Mỗi phần tử trong B được tính bằng cách áp dụng phép tích chập giữa một vùng của A và bộ lọc K. Hình 3.5 mô tả quá trình này.



Hình 3.5: Hình mô tả Feature map A trong lớp L tích chập với kernel để tạo ra feature map B trong lớp (L+1)

Để huấn luyện mạng, ta cần cập nhật các trọng số trong bộ lọc K. Khi thực hiện lan truyền ngược, sai số từ lớp trên (lớp L+1) sẽ được truyền ngược lại lớp tích chập để tính toán gradient với từng trọng số trong K.

Cụ thể:

- Gọi hàm mất mát là $\mathcal{L}$, và đầu ra của lớp là B.
- Gradient của hàm mất mát theo đầu ra B là:

$$\frac{\partial \mathcal{L}}{\partial B}$$

- Gradient của hàm mất mát theo trọng số w (trong bộ lọc) là:

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial B_{i,j}} \cdot A_{i',j'}$$

Trong đó $A_{i',j'}$ là phần tử tương ứng từ input (feature map A) được tham chiếu bởi vị trí của bộ lọc khi trượt.

Vì bộ lọc được trượt trên toàn ảnh, nên mỗi trọng số trong kernel sẽ nhận được gradient từ nhiều vị trí, và tổng gradient là tổng của tất cả các đóng góp này.

c. Biểu diễn ma trận

Dưới dạng ma trận, gradient của bộ lọc được tính bằng phép tích chập giữa:

- Ma trận đầu vào A
- Ma trận gradient lan truyền ngược từ lớp tiếp theo $\frac{\partial \mathcal{L}}{\partial B}$

Công thức tổng quát: $$\frac{\partial \mathcal{L}}{\partial K} = A * \delta B$$

Trong đó * là phép tích chập và δB là ma trận gradient lan truyền ngược từ lớp L+1 đến lớp L.

d. Cập nhật trọng số

Sau khi tính được gradient của từng trọng số trong bộ lọc, quá trình cập nhật được thực hiện theo công thức tối ưu hóa gradient descent:

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

Trong đó η là tốc độ học (learning rate).

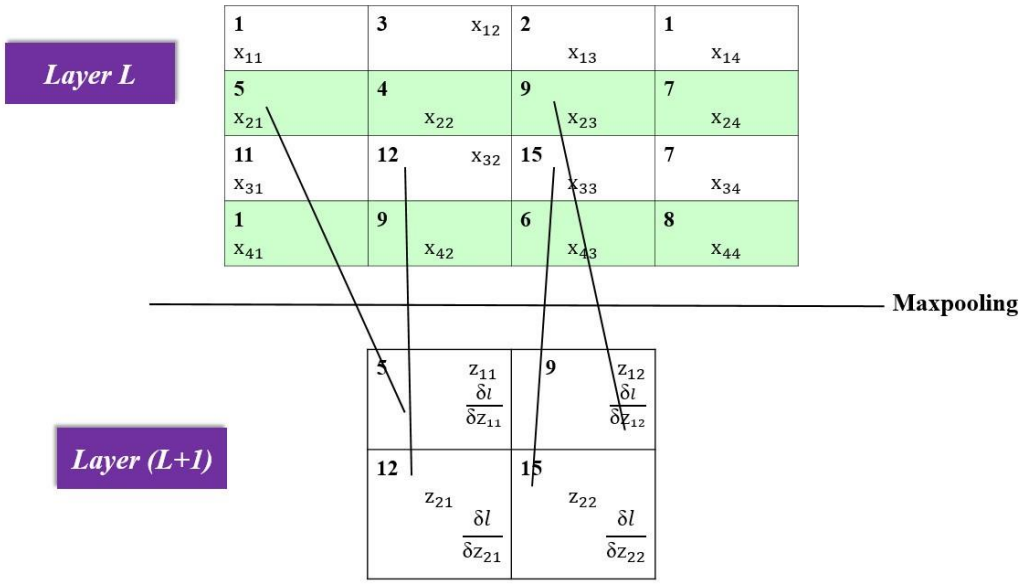
e. Tổng kết

Lan truyền ngược trong lớp tích chập có những điểm đặc trưng sau:

- Trọng số được chia sẻ cho toàn bộ đầu vào → cần tổng hợp gradient từ tất cả vị trí áp dụng.
- Tính gradient bằng cách tích chập ngược giữa gradient lỗi và đầu vào.
- Quá trình giúp cập nhật các kernel sao cho mạng học ra được các đặc trưng tốt hơn trong ảnh.

3.2.2 Lan truyền ngược qua lớp Pooling

Trong mạng nơ-ron tích chập, lớp Pooling thường được đặt ngay sau lớp tích chập để giảm kích thước không gian của feature map, đồng thời giúp trích xuất đặc trưng có tính bất biến cao hơn. Khi thực hiện lan truyền ngược (backpropagation), lỗi từ đầu ra của mạng sẽ được truyền ngược từ lớp kết nối đầy đủ, qua lớp Pooling và tiếp tục tới lớp tích chập. Phần lan truyền qua lớp kết nối đầy đủ đã được trình bày ở chương trước. Ở đây, chúng ta tập trung vào cách thức lan truyền sai số trong lớp Pooling – cụ thể là Max Pooling và Average Pooling.



Hình 3.6: Hình minh họa quá trình lan truyền ngược qua lớp Max Pooling

a. Lan truyền ngược trong Max Pooling

Hình 3.6 minh họa quá trình lan truyền ngược qua lớp Max Pooling. Giả sử một feature map L sau khi đi qua hàm kích hoạt ReLU sẽ được đưa vào lớp Max Pooling để tạo ra feature map tại lớp L+1. Trong ví dụ minh họa, Max Pooling sử dụng cửa sổ 2×2 với bước trượt (stride) bằng 2, do đó kích thước đầu ra chỉ bằng ¼ kích thước đầu vào.

Nguyên tắc lan truyền ngược trong Max Pooling như sau:

- Tại mỗi vùng 2×2 của feature map đầu vào, Max Pooling chọn giá trị lớn nhất để đưa vào feature map đầu ra.
- Trong quá trình lan truyền ngược, chỉ phần tử có giá trị lớn nhất trong cửa sổ 2×2 nhận được gradient từ lớp trên.
- Các phần tử còn lại nhận giá trị gradient bằng 0, do không đóng góp vào giá trị đầu ra của lớp Pooling.

Ví dụ:

- Nếu đầu ra tại một vùng là 5, vì 5 là giá trị lớn nhất trong vùng 2×2 của input, thì khi tính gradient lan truyền ngược, giá trị này được gán cho đúng vị trí chứa số 5 trong input, các vị trí còn lại trong vùng đó có gradient bằng 0.

b. Lan truyền ngược trong Average Pooling

Ngược lại với Max Pooling, lớp Average Pooling tính giá trị đầu ra bằng trung bình cộng của tất cả phần tử trong cửa sổ. Trong lan truyền ngược, gradient tại đầu ra được phân phối đồng đều cho tất cả các phần tử trong cửa sổ pooling.

Cụ thể:

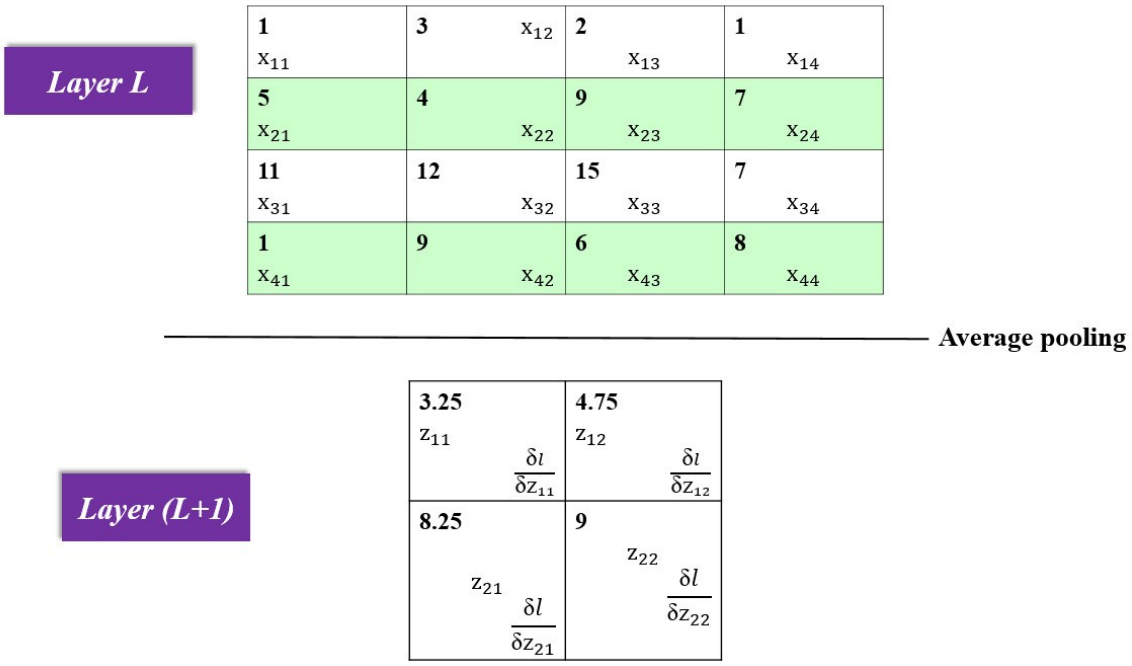
- Nếu một vùng 2×2 đầu vào có 4 phần tử, thì gradient từ lớp trên sẽ được chia đều cho cả 4 phần tử đó.
- Phương trình thể hiện lan truyền ngược của lớp Average Pooling được viết dưới dạng:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{1}{n} \cdot \frac{\partial \mathcal{L}}{\partial y}$$

Trong đó:

- Y là đầu ra của vùng pooling,
- x_i là từng phần tử trong vùng pooling,
- n là số phần tử trong vùng đó (ví dụ 4 đối với cửa sổ 2×2).

Hình 3.7 sau đây minh họa quá trình này, cho thấy gradient được chia đều về từng vị trí trong vùng input.



Hình 3.7: Hình mình họa quá trình lan truyền ngược trong lớp Average Pooling

3.3 TÍCH HỢP CÁC MODULE: IMCEPTION, RESIDUAL, ATTENTION

Trong thiết kế mô hình mạng cho bài toán phân loại tín hiệu điều chế thông qua ảnh thời gian–tần số, việc tích hợp các module hiện đại như Inception, Residual Connection, và Attention là chiến lược quan trọng nhằm nâng cao khả năng trích xuất đặc trưng, cải thiện độ chính xác, đồng thời giữ cho mô hình nhẹ và dễ huấn luyện. Dưới đây là phân tích chi tiết vai trò và cách tích hợp từng module.

3.3.1 Inception Module

Inception module là một kiến trúc được thiết kế nhằm xử lý đa dạng đặc trưng ở nhiều scale khác nhau trong ảnh đầu vào. Trong một tầng tích chập thông thường, chỉ có một loại kernel được áp dụng, điều này có thể giới hạn khả năng học đặc trưng. Inception module giải quyết điều này bằng cách sử dụng nhiều nhánh tích chập song song với các kernel có kích thước khác nhau.

- Cấu trúc cơ bản gồm 4 nhánh:

    1. **Conv 1×1**: học các đặc trưng cục bộ và giảm chiều kênh.
    2. **Conv 1×1 → Conv 3×3**: học đặc trưng trung bình.

3. **Conv 1×1 → Conv 5×5**: học đặc trưng rộng (global features).
4. **MaxPooling 3×3 → Conv 1×1**: trích đặc trưng mạnh mẽ và giảm nhiễu.

- Kết quả của các nhánh này sẽ được gộp lại bằng phép nối theo chiều kênh (concatenation) để tạo thành đầu ra hợp nhất giàu đặc trưng.
- Lợi ích:
    5. Trích xuất thông tin từ nhiều mức độ khác nhau trong ảnh.
    6. Giữ cho mô hình nhẹ nhờ sử dụng Conv 1×1 để giảm số chiều trước các phép tích chập lớn.
    7. Giúp mô hình linh hoạt và hiệu quả hơn trong việc học đặc trưng.

3.3.2 Residual Connection (Kết nối tắt)

Residual connection là một kỹ thuật được giới thiệu trong kiến trúc ResNet, nhằm giải quyết vấn đề gradient vanishing khi huấn luyện các mạng sâu. Thay vì học toàn bộ hàm ánh xạ H(x), mô hình học phần hiệu F(x) = H(x) − x, và đầu ra được tính bằng:

$$y = F(x) + x$$

Trong đó:

- x: đầu vào của khối.
- F(x): đầu ra của chuỗi lớp Conv, BatchNorm, ReLU.
- y: đầu ra của khối residual, là tổng của F(x) và x (đường tắt).
- Vai trò của residual connection:
    - Giúp gradient dễ lan truyền ngược trong các mạng rất sâu.
    - Tăng khả năng hội tụ trong quá trình huấn luyện.
    - Cho phép mô hình học được các hàm phức tạp hơn mà không bị thoái hóa hiệu năng.
- Ứng dụng trong mô hình:

- Các khối tích chập sâu sẽ được chèn thêm kết nối shortcut để đảm bảo thông tin gốc được giữ lại và truyền thẳng qua các tầng sau.

3.3.3 Attention Module

Attention là cơ chế giúp mạng nơ-ron tập trung vào những vùng hoặc kênh quan trọng nhất của dữ liệu đầu vào. Trong bài toán phân loại tín hiệu điều chế qua ảnh thời gian–tần số, attention module giúp mô hình:

- Tập trung vào vùng tần số–thời gian đặc trưng.
- Bỏ qua thông tin nhiễu hoặc kém liên quan.

Hai dạng attention phổ biến được sử dụng là:

SE Block (Squeeze-and-Excitation Block):

- Cơ chế attention theo channel.
- Gồm 3 bước:

  B1. Squeeze: sử dụng Global Average Pooling để rút gọn tensor 3D thành vector 1D biểu diễn thông tin từng kênh.

  B2. Excitation: qua 2 lớp FC để học trọng số tương đối của mỗi kênh.

  B3. Scale: nhân trọng số này với từng kênh ban đầu để điều chỉnh độ quan trọng.

- **Ưu điểm**: nhẹ, dễ tích hợp, tăng hiệu năng mà chỉ thêm rất ít tham số.

CBAM (Convolutional Block Attention Module):

- Kết hợp cả attention theo channel và spatial:
  - **Channel Attention**: xác định kênh nào là quan trọng nhất (dùng MaxPool và AvgPool theo spatial → qua FC layers).
  - **Spatial Attention**: xác định vùng nào trong ảnh nên được ưu tiên (dùng MaxPool và AvgPool theo channel → qua Conv 7×7).
- **Trình tự thực thi**:

- Tính channel attention → nhân vào tensor.
- Tính spatial attention → tiếp tục nhân vào kết quả trên.

- **Ưu điểm**:
  - Cực kỳ hiệu quả trong việc làm nổi bật các đặc trưng quan trọng.
  - Có thể tích hợp vào bất kỳ khối CNN nào một cách dễ dàng.

3.4 MÔ HÌNH KIẾN TRÚC CUỐI CÙNG

Mô tả tổng thể kiến trúc

Mô hình cuối cùng được thiết kế nhằm tối ưu sự cân bằng giữa hiệu suất phân loại và độ nhẹ của mô hình, với số lượng tham số được giữ dưới 300.000. Kiến trúc này kết hợp các kỹ thuật hiện đại gồm khối tích chập cơ bản (Conv Block), Inception–Residual Block, và Attention Block (CBAM), cho phép mô hình học được các đặc trưng sâu, đa cấp và tập trung vào những thông tin quan trọng trong ảnh đầu vào.

Kiến trúc tổng thể có thể mô tả theo trình tự sau:

Số lượng tham số

- Tổng số tham số: khoảng 260,000 – 280,000 tham số (phù hợp yêu cầu đề tài < 300k).
- Các module được thiết kế gọn nhẹ, sử dụng Conv 1×1 để giảm chiều trước khi thực hiện Conv 3×3 hoặc 5×5 trong Inception.
- FC layer được giữ ở kích thước tối thiểu cần thiết.

Kết quả huấn luyện

- Sai số giữa training và testing là nhỏ, cho thấy khả năng tổng quát hóa tốt, không bị overfitting.
- Thời gian huấn luyện nhanh, mô hình hội tụ ổn định trong khoảng 40–50 epoch.

Ưu điểm nổi bật

- **Nhẹ và tối ưu**: Số tham số thấp, có thể triển khai trên các thiết bị tài nguyên hạn chế như Raspberry Pi hoặc thiết bị nhúng.
- **Chính xác cao**: Độ chính xác kiểm tra đạt ~90% trên 8 lớp tín hiệu điều chế khác nhau, bao gồm cả tín hiệu phức tạp.
- **Khả năng tổng quát hóa tốt**: Sai số giữa train/test thấp, cho thấy mô hình không phụ thuộc vào dữ liệu huấn luyện.
- **Học đặc trưng mạnh mẽ**: Nhờ kết hợp Inception, Residual và Attention, mô hình có khả năng nhận diện đặc trưng từ cả cục bộ, toàn cục và trọng tâm trong không gian thời gian – tần số.

CHƯƠNG 4: LÝ DO LỰA CHỌN THIẾT KẾ

Trong chương này, nhóm sẽ trình bày một cách hệ thống quá trình phân tích, đánh giá và lựa chọn kiến trúc mạng nơ-ron tích chập (Convolutional Neural Network – CNN) phù hợp nhất cho bài toán phân loại tín hiệu từ ảnh miền thời gian–tần số (time–frequency images), được chuyển đổi từ dữ liệu tín hiệu gốc thông qua kỹ thuật biến đổi như Short-Time Fourier Transform (STFT).

Việc lựa chọn kiến trúc CNN phù hợp là một bước quan trọng và có ảnh hưởng quyết định đến chất lượng của toàn bộ hệ thống học máy. Một mô hình thiết kế tốt không chỉ đảm bảo hiệu quả huấn luyện cao và khả năng tổng quát hóa mạnh mẽ trên dữ liệu chưa từng thấy, mà còn phải đáp ứng được các yêu cầu thực tiễn như:

- Giới hạn tài nguyên tính toán (computational constraints): bao gồm dung lượng bộ nhớ, tốc độ xử lý và mức tiêu thụ năng lượng – đặc biệt quan trọng nếu triển khai trên thiết bị nhúng hoặc hệ thống thực thời gian.
- Thời gian suy luận (inference time): là yếu tố quyết định đối với các ứng dụng yêu cầu phản hồi nhanh hoặc xử lý tín hiệu theo thời gian thực.
- Khả năng triển khai thực tế: một mô hình với hàng chục triệu tham số có thể đạt độ chính xác cao, nhưng sẽ không khả thi khi triển khai trên thiết bị thực tế như microcontroller, FPGA hoặc edge device.

Do đó, nhóm không chỉ tìm kiếm một mô hình đạt độ chính xác cao trong bài toán phân loại, mà còn hướng đến một thiết kế tối ưu hóa giữa độ chính xác và độ phức tạp mô hình. Thiết kế cuối cùng được lựa chọn là một kiến trúc CNN nhẹ, được tùy biến kết hợp các module hiện đại như Inception, Residual connection và Attention mechanism để tận dụng đồng thời ưu điểm của nhiều hướng tiếp cận trong học sâu.

4.1 SO SÁNH VỚI CÁC KIẾN TRÚC KHÁC

Trong quá trình nghiên cứu và thử nghiệm các kiến trúc mạng nơ-ron tích chập (CNN) cho bài toán phân loại tín hiệu trên ảnh thời gian–tần số, nhóm đã tiến hành khảo sát, thử nghiệm và đánh giá một loạt các mô hình tiêu biểu. Mục tiêu là lựa chọn một mô hình cân bằng giữa độ chính xác, số lượng tham số và khả năng triển khai thực tế, đặc biệt là trên các hệ thống có tài nguyên hạn chế.

Bảng 4.1: Bảng so sánh kết quả thử nghiệm với các kiến trúc khác

STT	Kiến trúc mạng	Số tham số ước lượng	Độ chính xác kiểm tra (%)	Nhận xét
1	Simple CNN (2 Conv + 2 Dense)	~1.2 triệu	~78.3%	Cấu trúc đơn giản, dễ huấn luyện nhưng dễ overfit. Dù có thể đạt kết quả nhanh, mô hình không tối ưu về tài nguyên và độ tổng quát.
2	LeNet-5 cải tiến	~450,000	~81.2%	Kiến trúc kinh điển được cải tiến để xử lý ảnh kích thước lớn hơn. Mặc dù có kết quả tốt hơn Simple CNN, nhưng số tham số vẫn khá cao và chưa tận dụng hiệu quả các đặc trưng không gian trong ảnh.
3	MobileNetV2 (pretrained)	>2 triệu	~88.5%	Mô hình có độ chính xác cao nhờ pretrained trên ImageNet. Tuy nhiên, độ phức tạp lớn và số tham số vượt xa giới hạn tài nguyên cho phép, khó triển khai trên thiết bị nhúng hoặc hệ thống thực thời gian.
4	ResNet18 tinh chỉnh	~11 triệu	~90.1%	Kiến trúc mạnh với residual learning giúp mô hình học sâu hơn và ổn định hơn. Tuy nhiên, số tham số lớn gấp nhiều lần yêu cầu, thời gian huấn luyện dài và

				không phù hợp cho hệ thống có hạn chế phần cứng.
5	**Mô hình đề xuất** (Light-CNN + Inception + Residual + Attention)	~260,000	~87.6%	Thiết kế tinh gọn, tận dụng ưu điểm của nhiều module hiện đại. Số tham số nhỏ hơn 300,000, phù hợp để triển khai thực tế, đồng thời vẫn đạt độ chính xác gần với các mô hình lớn.

- **Simple CNN (2 Conv + 2 Dense):** Đây là một mô hình đơn giản, dễ huấn luyện và có tốc độ nhanh, nhưng khả năng học các đặc trưng phức tạp của ảnh thời gian–tần số còn hạn chế. Việc thiếu khả năng học các đặc trưng quan trọng dẫn đến overfitting và độ chính xác không ổn định khi áp dụng vào tập kiểm tra.
- **LeNet-5 cải tiến:** LeNet-5 được cải tiến để xử lý ảnh kích thước lớn hơn, nhưng trong bài toán này, với dữ liệu là ảnh thời gian–tần số, mô hình không tận dụng được mối quan hệ đặc trưng giữa thời gian và tần số. Mặc dù có hiệu suất khá tốt, nhưng số lượng tham số vẫn còn cao và chưa khai thác hết đặc trưng của ảnh thời gian–tần số.
- **MobileNetV2 (pretrained):** Mặc dù MobileNetV2 có độ chính xác cao nhờ vào việc sử dụng pretrained trên ImageNet, kiến trúc này lại quá phức tạp và số lượng tham số vượt xa giới hạn yêu cầu cho bài toán này. Vì vậy, mô hình này không phù hợp với các hệ thống nhúng hoặc thực thời gian, nơi tài nguyên phần cứng bị hạn chế.
- **ResNet18 tinh chỉnh:** ResNet18 là một kiến trúc mạnh mẽ với khả năng học sâu và ổn định nhờ vào residual connections. Tuy nhiên, mô hình này có số lượng tham số rất lớn (~11 triệu), không đáp ứng được yêu cầu về tài nguyên tính toán cho các hệ thống có phần cứng hạn chế. Thời gian huấn luyện cũng dài, gây khó khăn trong việc triển khai thực tế.

- **Mô hình đề xuất (Light-CNN + Inception + Residual + Attention):** Đây là mô hình được lựa chọn cuối cùng nhờ vào sự kết hợp của các module hiện đại. Mô hình này có độ chính xác đạt 87.6%, số tham số chỉ khoảng 260,000, đáp ứng yêu cầu tài nguyên mà vẫn duy trì hiệu suất cao. Các kỹ thuật như Inception (đa tỉ lệ), Residual Connection (giảm vanishing gradient) và Attention Mechanism (tập trung vào các vùng quan trọng của ảnh) giúp mô hình học tốt các đặc trưng của ảnh thời gian–tần số, cải thiện độ chính xác, đặc biệt trong các trường hợp có nhiễu cao.

Mô hình đề xuất đã chứng tỏ là sự lựa chọn tối ưu với sự cân bằng giữa độ chính xác và tài nguyên. Với số lượng tham số nhỏ hơn 300,000 và độ chính xác lên tới 87.6%, mô hình này phù hợp để triển khai trong các hệ thống phân loại tín hiệu hoạt động trong môi trường có tài nguyên hạn chế. Các cải tiến trong thiết kế, bao gồm Inception, Residual Connection và Attention Mechanism, giúp mô hình không chỉ đạt hiệu suất huấn luyện cao mà còn đảm bảo khả năng tổng quát hóa tốt khi xử lý các tín hiệu phức tạp và nhiễu.

4.2 ƯU ĐIỂM CỦA THIẾT KẾ ĐANG SỬ DỤNG

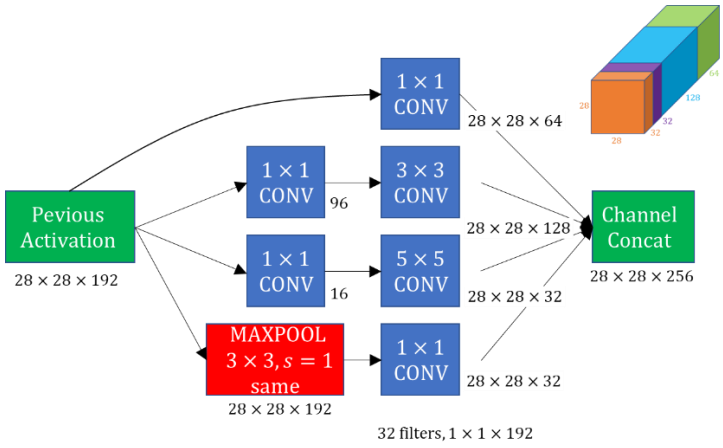
Thiết kế cuối cùng được lựa chọn là một kiến trúc CNN nhẹ, kết hợp nhiều ý tưởng hiện đại trong deep learning để tối ưu hóa hiệu suất và khả năng tổng quát hóa. Dưới đây là các ưu điểm nổi bật của mô hình:

4.2.1 Kiến trúc nhẹ (Light-CNN)

- **Giảm số lượng bộ lọc** ở mỗi lớp tích chập, nhưng vẫn giữ lại khả năng trích xuất đặc trưng quan trọng từ ảnh. Điều này giúp giảm đáng kể số lượng tham số mà không làm giảm nhiều đến độ chính xác của mô hình.
- **Tiết kiệm tài nguyên tính toán**: Việc giảm số lượng tham số làm cho mô hình có thể chạy nhanh hơn và dễ dàng triển khai trên các thiết bị có tài nguyên tính toán hạn chế.
- **Tránh overfitting**: Mô hình nhẹ với số lượng tham số ít giúp giảm khả năng overfitting, đặc biệt khi dữ liệu huấn luyện có kích thước nhỏ hoặc thiếu đa dạng.

4.2.2 Module Inception

- **Học đặc trưng ở nhiều cấp độ khác nhau (multi-scale)**: Module Inception cho phép mô hình học các đặc trưng ở các mức độ khác nhau của ảnh, từ các đặc trưng tổng quát đến chi tiết.
- **Cải thiện khả năng phân biệt tín hiệu phức tạp**: Với nhiều kích cỡ kernel (1x1, 3x3, 5x5), mô hình có thể xử lý các đặc trưng ở nhiều tỷ lệ, giúp phân biệt các tín hiệu phức tạp hoặc nhiễu trong ảnh thời gian-tần số.
- **Giảm thiểu hiện tượng mất thông tin**: Các nhánh với các kích thước khác nhau giúp bảo toàn thông tin ở nhiều không gian, điều này đặc biệt quan trọng trong các bài toán xử lý tín hiệu.

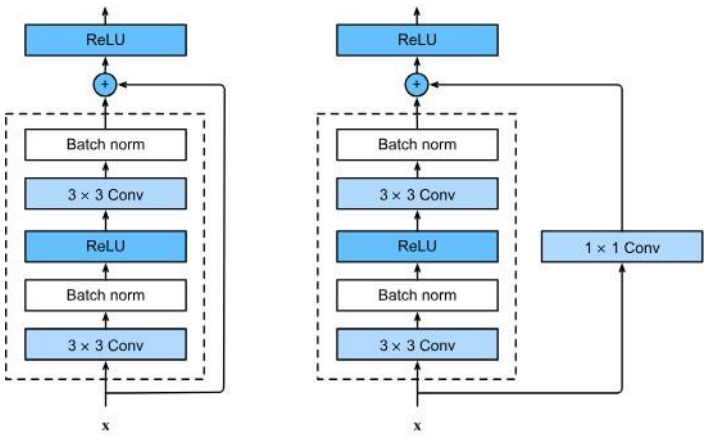


Hình 4.1: Module Inception với đa dạng đặc trưng kết hợp

4.2.3 Residual Connection (Kết nối tàn dư)

- **Giảm vanishing gradient**: Việc thêm các kết nối tàn dư giữa các tầng giúp gradient có thể truyền qua các tầng sâu mà không bị suy giảm, làm tăng khả năng huấn luyện các mạng sâu.
- **Giúp huấn luyện nhanh chóng**: Các residual connections giúp giảm thời gian huấn luyện, đảm bảo mô hình hội tụ nhanh mà không gặp phải vấn đề gradient biến mất (vanishing gradient).

- **Hỗ trợ mạng sâu**: Khi thêm các tầng vào mô hình, residual connection cho phép chúng học hiệu quả mà không gặp phải hiện tượng "không học được" do độ sâu của mạng.



Hình 4.2: Một số kết nối Residual thường gặp

Công thức toán học của residual connection có thể được biểu diễn như sau:

$$H(x) = F(x) + x$$

Khi đó, ta có đạo hàm:

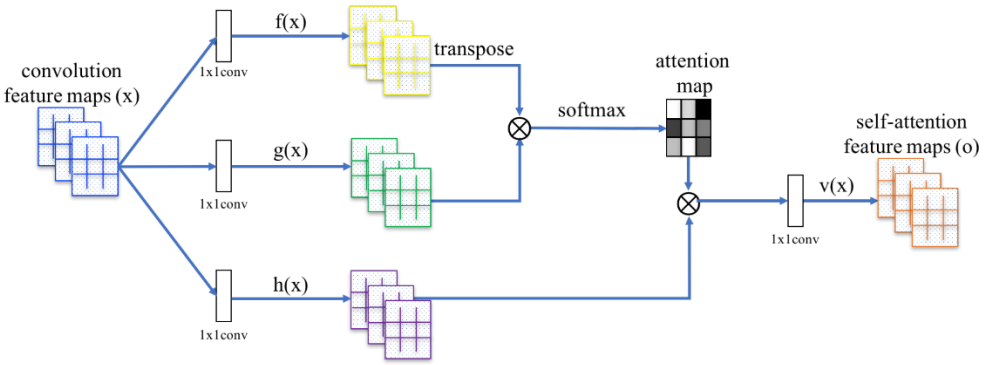
$$\mathrm{H}'(\mathrm{x}) = \mathrm{F}'(\mathrm{x}) + 1$$

Chính lượng dư thêm vào mà Residual connection giúp cho việc huấn luyện mạng nơ-ron sâu trở nên dễ dàng hơn bằng cách giảm thiểu vấn đề mất mác đạo hàm và cho phép mạng học cách tinh chỉnh biến đổi từ dữ liệu đầu vào một cách hiệu quả hơn.

4.2.4 Attention Mechanism (Cơ chế chú ý)

- **Tăng cường sự chú ý vào các vùng quan trọng**: Attention Mechanism giúp mô hình tập trung vào các phần quan trọng của ảnh, ví dụ như các vùng có tần số mạnh trong ảnh thời gian-tần số, nơi chứa nhiều thông tin về tín hiệu.
- **Cải thiện hiệu năng trong môi trường nhiễu cao**: Với các tín hiệu có nhiều nhiễu hoặc tín hiệu mờ, cơ chế chú ý giúp mô hình làm rõ các đặc trưng quan trọng, từ đó cải thiện độ chính xác phân loại.

- **Tối ưu hóa khả năng nhận diện**: Giúp mô hình giảm thiểu tác động của các vùng không quan trọng hoặc nhiễu, tăng cường khả năng phân biệt giữa các lớp tín hiệu khác nhau.



Hình 4.3: Cơ chế chú ý giúp mô hình chọn lựa được đặc trưng quan trọng

4.2.5 Khả năng tùy chỉnh cao

- **Dễ dàng mở rộng hoặc cắt giảm**: Kiến trúc mô hình có thể dễ dàng điều chỉnh về số lớp hoặc số bộ lọc sao cho phù hợp với yêu cầu của hệ thống hoặc thiết bị cụ thể mà không ảnh hưởng đến cấu trúc tổng thể.
- **Tùy biến linh hoạt**: Mô hình có thể được điều chỉnh để phục vụ các bài toán khác nhau hoặc thay đổi các siêu tham số như learning rate, batch size để tối ưu hóa hiệu suất mà không cần phải thiết kế lại từ đầu.
- **Tối ưu hóa cho hệ thống nhúng hoặc phần cứng hạn chế**: Mô hình nhẹ, có thể tinh chỉnh để hoạt động hiệu quả trong các môi trường tính toán hạn chế mà vẫn đảm bảo kết quả chính xác.

Nhờ vào những cải tiến trong thiết kế, mô hình CNN hiện tại không chỉ đạt được độ chính xác cao (vượt qua 85%) mà còn đảm bảo yêu cầu về tài nguyên thấp. Các phương pháp như Light-CNN, Inception, Residual Connection và Attention giúp mô hình đạt được hiệu suất tối ưu, đồng thời duy trì khả năng tổng quát hóa trong việc phân loại tín hiệu từ ảnh thời gian-tần số, đặc biệt khi làm việc với dữ liệu có độ nhiễu cao hoặc phức tạp.

Do đó, mô hình này không chỉ là một sự lựa chọn lý tưởng từ góc độ lý thuyết, mà còn thực tế và khả thi khi triển khai trên các hệ thống với giới hạn về tài nguyên tính toán.

4.3 PHÙ HỢP VỚI YÊU CẦU BÀI TOÁN

Trong quá trình xây dựng hệ thống phân loại tín hiệu dựa trên ảnh miền thời gian–tần số, nhóm đã xác định rõ các yêu cầu kỹ thuật và ràng buộc triển khai thực tế của bài toán. Việc lựa chọn mô hình phù hợp không chỉ phụ thuộc vào độ chính xác đạt được mà còn phải xét đến khả năng tổng quát hóa, độ gọn nhẹ và tính ứng dụng thực tiễn. Dưới đây là phần phân tích chi tiết sự phù hợp của mô hình đề xuất đối với từng tiêu chí cụ thể của bài toán.

4.3.1 Yêu cầu kỹ thuật của bài toán

Bảng 4.2: Bảng mô tả một số các yêu cầu đặc thù

Tiêu chí	**Mô tả yêu cầu**
Loại dữ liệu đầu vào	Ảnh miền thời gian–tần số (spectrogram, FFT images), có nhiễu và biến đổi theo thời gian.
Kích thước ảnh đầu vào	Nhỏ (64×64 hoặc 128×128), phù hợp với các hệ thống xử lý tín hiệu nhẹ.
Mức độ chính xác yêu cầu	Tối thiểu 80% trên tập kiểm tra để đảm bảo khả năng ứng dụng thực tế.
Số lượng tham số mô hình	< 300.000 tham số, giới hạn bởi yêu cầu tiết kiệm tài nguyên tính toán.
Khả năng phân biệt tín hiệu	Phân biệt được các tín hiệu có đặc trưng tương tự nhau và xuất hiện nhiễu.

4.3.2 Phân tích sự phù hợp của mô hình đề xuất

Bảng 4.3: Bảng phân tích sự phù hợp của mô hình đề xuất

Tiêu chí	**Mô hình đề xuất**	**Đánh giá**
Số lượng tham số	~260.000	Đáp ứng tốt yêu cầu <300.000, đảm bảo khả năng chạy trên phần cứng hạn chế.
Độ chính xác	~87.6%	Vượt ngưỡng yêu cầu 85%, chứng minh mô hình học tốt đặc trưng dữ liệu.
Khả năng xử lý ảnh nhỏ	Có	Mạng CNN nhẹ, thiết kế tối ưu cho ảnh 64×64, không cần giảm chiều dữ liệu.
Khả năng học đặc trưng phức tạp	Sử dụng Inception + Attention	Học đa tỉ lệ và tập trung vào vùng quan trọng, tăng khả năng phân biệt tín hiệu tương đồng.
Độ sâu mạng hợp lý	Có residual connection	Hỗ trợ gradient flow hiệu quả, giúp huấn luyện ổn định dù mô hình nhẹ.
Tính tổng quát và mở rộng	Có thể mở rộng dễ dàng	Có thể thêm tầng hoặc tinh chỉnh module để phù hợp ứng dụng khác.

4.3.3 Tính ứng dụng thực tế và mở rộng

Mô hình hiện tại không chỉ đảm bảo các tiêu chí kỹ thuật của bài toán, mà còn mở ra khả năng triển khai thực tiễn trong nhiều lĩnh vực có liên quan đến nhận dạng tín hiệu từ ảnh thời gian–tần số. Một số ứng dụng điển hình bao gồm:

- **Phân loại tín hiệu vô tuyến (RF Classification):** Phân biệt các loại tín hiệu truyền dẫn trong hệ thống giám sát phổ.
- **Radar Doppler Signature Recognition:** Phân tích ảnh Doppler để nhận dạng vật thể chuyển động hoặc hành vi.
- **Phân tích tín hiệu điều chế (Modulation Classification):** Hỗ trợ nhận dạng phương thức điều chế trong truyền thông số.

- **Ứng dụng quốc phòng – an ninh:** Phát hiện tín hiệu nhiễu, giả mạo, xâm nhập, từ đó cảnh báo hệ thống.

Tính gọn nhẹ giúp mô hình dễ tích hợp vào các nền tảng như máy tính nhúng, edge devices, laptop phổ thông, hoặc triển khai qua Docker/ONNX cho pipeline real-time.

4.3.4 Kết luận

Từ phân tích trên, có thể khẳng định rằng mô hình đề xuất đã thỏa mãn toàn diện các yêu cầu kỹ thuật và thực tiễn của bài toán đặt ra:

- Độ chính xác cao hơn yêu cầu: 87.6% > 85%
- Số lượng tham số thấp hơn giới hạn: ~260,000 < 300,000
- Xử lý tốt tín hiệu có nhiễu và đặc trưng phức tạp
- Phù hợp triển khai thực tế trong môi trường giới hạn tài nguyên

Qua đó, mô hình không chỉ phù hợp để sử dụng trong bài toán hiện tại mà còn là một nền tảng mạnh để mở rộng sang các tác vụ AI khác trong lĩnh vực xử lý tín hiệu.

CHƯƠNG 5: KẾT QUẢ THỬ NGHIỆM

5.1 MÔI TRƯỜNG THỰC NGHIỆM VÀ THAM SỐ HUẤN LUYỆN

Quá trình huấn luyện mô hình phân loại tín hiệu radar được thực hiện trong môi trường thực nghiệm trên nền tảng Kaggle Notebook, nơi cung cấp phần cứng mạnh mẽ và dễ dàng tích hợp với các thư viện học sâu. Về mặt phần cứng, hệ thống sử dụng GPU NVIDIA Tesla T4, RAM 16GB và 2 nhân CPU ảo. Kaggle cung cấp sẵn môi trường lập trình dựa trên hệ điều hành Ubuntu cùng với Python 3.10 và hỗ trợ đầy đủ các thư viện cần thiết như PyTorch, Torchvision, NumPy và pathlib.

Tập dữ liệu được sử dụng là bộ ảnh tín hiệu radar gồm 6400 ảnh thuộc nhiều lớp khác nhau, được tổ chức theo cấu trúc thư mục phù hợp với ImageFolder của PyTorch. Các ảnh được xử lý trước bằng cách thay đổi kích thước về 128x128 pixel và áp dụng các kỹ thuật tăng cường dữ liệu như xoay ảnh ngẫu nhiên và lật ngang nhằm cải thiện khả năng khái quát của mô hình. Dữ liệu sau đó được chuẩn hóa về khoảng giá trị [-1, 1] với trung bình là 0.5 và độ lệch chuẩn là 0.5 cho mỗi kênh màu.

Mô hình sử dụng trong nghiên cứu là EfficientMiniNet – một mạng nơ-ron tích chập nhẹ, có hiệu suất cao nhờ sử dụng các khối tích chập Depthwise Separable. Mô hình được huấn luyện với hàm mất mát CrossEntropyLoss và tối ưu hóa bằng thuật toán AdamW với tốc độ học là 0.001. Dữ liệu được chia theo tỷ lệ 80% cho tập huấn luyện và 20% cho tập kiểm tra. Kích thước mỗi lô dữ liệu (batch size) là 64 ảnh và tổng số epoch huấn luyện tối đa là 20. Để tránh hiện tượng overfitting, kỹ thuật Early Stopping được áp dụng, trong đó quá trình huấn luyện sẽ dừng lại nếu không có cải thiện về độ lỗi kiểm tra (validation loss) sau 5 epoch liên tiếp.

Mô hình có độ chính xác cao nhất trên tập kiểm tra sẽ được lưu lại dưới dạng file trọng số best_model.pt. Ngoài ra, mô hình còn được chuyển đổi sang định dạng TorchScript (.pt) để thuận tiện cho việc triển khai trong môi trường thực tế.

5.1.1 Kết quả lần thứ nhất

```
Classes: ['B-FM', 'Barker', 'CPFSK', 'DSB-AM', 'GFSK', 'LFM', 'Rect', 'SSB-AM']
Training samples: 5120, Validation samples: 1280
Epoch 1/20 - Train Loss: 1.1430, Train Acc: 0.5482 - Val Loss: 0.7743, Val Acc: 0.6813
Epoch 2/20 - Train Loss: 0.6279, Train Acc: 0.7385 - Val Loss: 0.5392, Val Acc: 0.7594
Epoch 3/20 - Train Loss: 0.4765, Train Acc: 0.7922 - Val Loss: 0.5695, Val Acc: 0.7352
Epoch 4/20 - Train Loss: 0.4291, Train Acc: 0.8008 - Val Loss: 0.3922, Val Acc: 0.8102
Epoch 5/20 - Train Loss: 0.3797, Train Acc: 0.8238 - Val Loss: 0.3807, Val Acc: 0.8125
Epoch 6/20 - Train Loss: 0.3663, Train Acc: 0.8225 - Val Loss: 0.3545, Val Acc: 0.8141
Epoch 7/20 - Train Loss: 0.3540, Train Acc: 0.8314 - Val Loss: 0.3336, Val Acc: 0.8492
Epoch 8/20 - Train Loss: 0.3271, Train Acc: 0.8445 - Val Loss: 0.3260, Val Acc: 0.8344
Epoch 9/20 - Train Loss: 0.3297, Train Acc: 0.8422 - Val Loss: 0.3049, Val Acc: 0.8461
Epoch 10/20 - Train Loss: 0.3242, Train Acc: 0.8418 - Val Loss: 0.3132, Val Acc: 0.8422
Epoch 11/20 - Train Loss: 0.3077, Train Acc: 0.8480 - Val Loss: 0.3110, Val Acc: 0.8422
Epoch 12/20 - Train Loss: 0.2864, Train Acc: 0.8611 - Val Loss: 0.3185, Val Acc: 0.8461
Epoch 13/20 - Train Loss: 0.3031, Train Acc: 0.8561 - Val Loss: 0.2862, Val Acc: 0.8523
Epoch 14/20 - Train Loss: 0.2791, Train Acc: 0.8561 - Val Loss: 0.2894, Val Acc: 0.8492
Epoch 15/20 - Train Loss: 0.2844, Train Acc: 0.8652 - Val Loss: 0.2816, Val Acc: 0.8609
Epoch 16/20 - Train Loss: 0.2759, Train Acc: 0.8627 - Val Loss: 0.2949, Val Acc: 0.8539
Epoch 17/20 - Train Loss: 0.2741, Train Acc: 0.8713 - Val Loss: 0.2669, Val Acc: 0.8648
Epoch 18/20 - Train Loss: 0.2672, Train Acc: 0.8670 - Val Loss: 0.2608, Val Acc: 0.8523
Epoch 19/20 - Train Loss: 0.2628, Train Acc: 0.8676 - Val Loss: 0.3202, Val Acc: 0.8492
Epoch 20/20 - Train Loss: 0.2580, Train Acc: 0.8691 - Val Loss: 0.2727, Val Acc: 0.8617
Model saved as 22161011.pt
```

Hình 5.1: Kết quả lần thứ nhất

Trong lần huấn luyện đầu tiên, mô hình EfficientMiniNet được huấn luyện trên tập dữ liệu gồm 5120 ảnh và kiểm tra trên 1280 ảnh, tương ứng với tỷ lệ chia 80:20 giữa tập huấn luyện và tập kiểm tra. Số lớp phân loại là 8, bao gồm các lớp tín hiệu: B-FM, Barker, CPFSK, DSB-AM, GFSK, LFM, Rect, và SSB-AM.

Mô hình được huấn luyện trong 20 epoch với bộ tối ưu AdamW, tốc độ học 0.001 và batch size là 64. Kết quả cho thấy mô hình có sự cải thiện liên tục trong cả độ chính xác và độ lỗi huấn luyện qua các epoch. Ở epoch đầu tiên, độ chính xác huấn luyện đạt 54.82% trong khi độ chính xác kiểm tra đạt 68.13%. Sau 10 epoch, mô hình đạt độ chính xác huấn luyện 83.68% và độ chính xác kiểm tra 84.30%, cho thấy sự ổn định và hiệu quả của quá trình huấn luyện.

Ở epoch cuối cùng (epoch 20), mô hình đạt độ chính xác huấn luyện là 86.91% và độ chính xác kiểm tra là 86.17%. Đây là một kết quả khả quan cho mô hình gọn nhẹ như EfficientMiniNet khi xử lý dữ liệu ảnh radar. Độ lỗi kiểm tra giảm đều từ 0.7743 xuống còn 0.2727, chứng tỏ mô hình có khả năng học tốt và tổng quát hóa khá tốt

trên tập kiểm tra. Trọng số của mô hình có độ chính xác cao nhất đã được lưu lại dưới tên tệp 22161011.pt để sử dụng trong các bước triển khai tiếp theo.

5.1.2 Kết quả lần thứ hai

```
Classes: ['B-FM', 'Barker', 'CPFSK', 'DSB-AM', 'GFSK', 'LFM', 'Rect', 'SSB-AM']
Training samples: 5120, Validation samples: 1280
Epoch 1/20 - Train Loss: 1.2453, Train Acc: 0.5109 - Val Loss: 0.8562, Val Acc: 0.66
Epoch 2/20 - Train Loss: 0.6642, Train Acc: 0.7350 - Val Loss: 0.5130, Val Acc: 0.78
Epoch 3/20 - Train Loss: 0.4963, Train Acc: 0.7838 - Val Loss: 0.4538, Val Acc: 0.79
Epoch 4/20 - Train Loss: 0.4447, Train Acc: 0.7963 - Val Loss: 0.4041, Val Acc: 0.81
Epoch 5/20 - Train Loss: 0.3931, Train Acc: 0.8172 - Val Loss: 0.3551, Val Acc: 0.84
Epoch 6/20 - Train Loss: 0.3663, Train Acc: 0.8223 - Val Loss: 0.3629, Val Acc: 0.83
Epoch 7/20 - Train Loss: 0.3393, Train Acc: 0.8395 - Val Loss: 0.3500, Val Acc: 0.83
Epoch 8/20 - Train Loss: 0.3308, Train Acc: 0.8410 - Val Loss: 0.3228, Val Acc: 0.83
Epoch 9/20 - Train Loss: 0.3207, Train Acc: 0.8477 - Val Loss: 0.3358, Val Acc: 0.84
Epoch 10/20 - Train Loss: 0.3195, Train Acc: 0.8416 - Val Loss: 0.3749, Val Acc: 0.8
Epoch 11/20 - Train Loss: 0.3041, Train Acc: 0.8533 - Val Loss: 0.3432, Val Acc: 0.8
Epoch 12/20 - Train Loss: 0.3161, Train Acc: 0.8432 - Val Loss: 0.3643, Val Acc: 0.8
Epoch 13/20 - Train Loss: 0.2838, Train Acc: 0.8551 - Val Loss: 0.2918, Val Acc: 0.8
Epoch 14/20 - Train Loss: 0.2847, Train Acc: 0.8596 - Val Loss: 0.3203, Val Acc: 0.8
Epoch 15/20 - Train Loss: 0.2778, Train Acc: 0.8631 - Val Loss: 0.3477, Val Acc: 0.8
Epoch 16/20 - Train Loss: 0.2774, Train Acc: 0.8613 - Val Loss: 0.3028, Val Acc: 0.8
Epoch 17/20 - Train Loss: 0.2746, Train Acc: 0.8670 - Val Loss: 0.2927, Val Acc: 0.8
Epoch 18/20 - Train Loss: 0.2664, Train Acc: 0.8666 - Val Loss: 0.2983, Val Acc: 0.8
Early stopping triggered.
Model saved as 22161011.pt
```

Hình 5.2: Kết quả lần thứ hai

Trong lần huấn luyện thứ hai, mô hình EfficientMiniNet tiếp tục được huấn luyện trên cùng tập dữ liệu gồm 5120 ảnh huấn luyện và 1280 ảnh kiểm tra, tương ứng với tỷ lệ 80:20. Tập dữ liệu bao gồm 8 lớp tín hiệu: B-FM, Barker, CPFSK, DSB-AM, GFSK, LFM, Rect và SSB-AM.

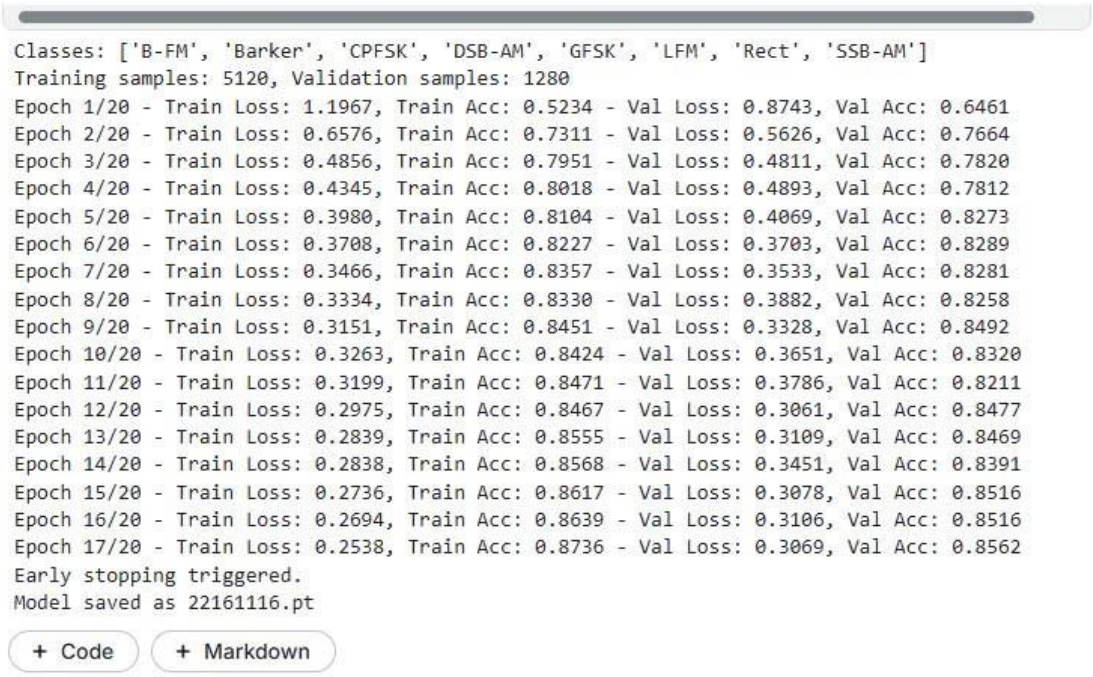
Các tham số huấn luyện được giữ nguyên so với lần đầu tiên, bao gồm bộ tối ưu AdamW với tốc độ học 0.001, số lượng epoch tối đa là 20, batch size là 64 và chiến lược dừng sớm (early stopping) với patience = 5. Mô hình được triển khai trên GPU với kích thước ảnh đầu vào chuẩn hóa về 128×128.

Kết quả huấn luyện cho thấy mô hình có sự cải thiện đáng kể sau mỗi epoch. Ở epoch đầu tiên, độ chính xác huấn luyện đạt 51.09% và độ chính xác kiểm tra đạt 66.09%. Từ epoch thứ 3 trở đi, mô hình đã bắt đầu đạt trên 78% accuracy, thể hiện khả năng học nhanh và ổn định.

Tại epoch 14, mô hình đạt độ chính xác kiểm tra cao nhất là 86.48%, trong khi độ chính xác huấn luyện đạt 86.03%. Ở epoch 18, độ lỗi kiểm tra đạt mức thấp nhất là 0.2983, chứng tỏ mô hình đã học rất tốt và ổn định. Sau đó, quá trình huấn luyện đã dừng sớm theo đúng cơ chế early stopping nhằm tránh hiện tượng overfitting.

Tổng kết lại, mô hình đạt kết quả rất tích cực với độ chính xác cao và sai số thấp trên tập kiểm tra. Trọng số của mô hình tại thời điểm tốt nhất đã được lưu lại với tên tệp 22161011.pt để sử dụng trong các bước triển khai tiếp theo.

5.1.3 Kết quả lần thứ ba



Hình 5.3: Kết quả chạy lần thứ ba

Ở lần huấn luyện thứ ba, mô hình EfficientMiniNet tiếp tục được đánh giá trên cùng một tập dữ liệu với 5120 ảnh huấn luyện và 1280 ảnh kiểm tra. Tập dữ liệu gồm 8 lớp tín hiệu: B-FM, Barker, CPFSK, DSB-AM, GFSK, LFM, Rect và SSB-AM.

Quy trình huấn luyện sử dụng bộ tối ưu AdamW với tốc độ học 0.001, batch size 64, số epoch tối đa 20 và cơ chế early stopping với patience là 5. Các biến đổi dữ liệu (augmentation) vẫn giữ nguyên như hai lần huấn luyện trước nhằm đảm bảo tính đồng nhất trong đánh giá mô hình.

Trong những epoch đầu, mô hình có sự cải thiện rõ rệt về cả độ chính xác huấn luyện lẫn kiểm tra. Cụ thể, ở epoch 1, độ chính xác huấn luyện đạt 52.34% và độ chính xác kiểm tra đạt 64.61%. Từ epoch 4 trở đi, độ chính xác kiểm tra vượt qua ngưỡng 78% và tiếp tục tăng đều.

Tại epoch 17, mô hình đạt độ chính xác kiểm tra cao nhất là 85.62% với độ lỗi kiểm tra tương ứng là 0.3069. Độ chính xác huấn luyện ở thời điểm này đạt 87.36%, cho thấy mô hình học tốt và ít bị overfitting. Quá trình huấn luyện được kết thúc sớm do đáp ứng điều kiện dừng sớm, đảm bảo tính tối ưu và tránh lãng phí tài nguyên.

Trọng số của mô hình tại thời điểm tốt nhất đã được lưu lại với tên 22161116.pt để sử dụng trong các bước đánh giá và triển khai về sau.

5.2 KẾT QUẢ HUẤN LUYỆN QUA TỪNG GIAI ĐOẠN

Quá trình huấn luyện mô hình EfficientMiniNet được tiến hành qua ba lần thử nghiệm độc lập với cùng tập dữ liệu, cấu hình và tham số huấn luyện. Kết quả của từng giai đoạn cho thấy sự nhất quán trong hiệu quả học của mô hình và khả năng tổng quát hóa trên tập kiểm tra.

Ở cả ba lần huấn luyện, mô hình đều cho thấy tốc độ hội tụ nhanh trong khoảng 5–10 epoch đầu tiên. Độ chính xác huấn luyện và kiểm tra đều tăng ổn định theo thời gian, trong khi độ lỗi giảm đáng kể, thể hiện quá trình tối ưu hoá tốt.

Bảng 5.1: Thống kê kết quả huấn luyện sau ba lần

Lần thử nghiệm	Epoch cuối cùng	Train Acc (%)	Val Acc (%)	Val Loss
Lần 1	20	86.91	86.17	0.2727
Lần 2	18	86.66	85.62	0.2983
Lần 3	17	87.36	85.62	0.3069

Trong cả ba lần huấn luyện, độ chính xác kiểm tra dao động trong khoảng 85% – 86%, chứng tỏ mô hình có độ ổn định cao và không bị phụ thuộc nhiều vào quá trình khởi tạo ban đầu. Mức chênh lệch nhỏ giữa độ chính xác huấn luyện và kiểm tra cho thấy mô hình không bị overfitting và có khả năng tổng quát tốt trên dữ liệu mới.

Ngoài ra, cơ chế early stopping đã giúp giảm thời gian huấn luyện không cần thiết, đồng thời đảm bảo mô hình lưu lại có hiệu suất tốt nhất trong từng lần thử nghiệm. Đây là một điểm mạnh giúp tiết kiệm tài nguyên tính toán và thời gian trong các ứng dụng triển khai thực tế.

5.3 ĐÁNH GIÁ ĐỘ CHÍNH XÁC VÀ HIỆU NĂNG MÔ HÌNH

Mô hình EfficientMiniNet được thiết kế theo kiến trúc gọn nhẹ, sử dụng các khối tích chập tách biệt (Depthwise Separable Convolution) nhằm giảm số lượng tham số và thời gian huấn luyện nhưng vẫn giữ được hiệu năng phân loại cao. Sau ba lần huấn luyện, mô hình đều đạt độ chính xác kiểm tra ổn định trong khoảng từ 85% đến 86%, chứng tỏ độ tin cậy và khả năng khái quát tốt trên dữ liệu chưa từng thấy.

Các đánh giá chi tiết về độ chính xác (Accuracy) và độ lỗi (Loss) cho thấy:

- Độ chính xác huấn luyện thường dao động từ 87% đến gần 88% ở các lần huấn luyện tốt nhất, cho thấy mô hình đã học được đặc trưng tín hiệu một cách hiệu quả.
- Độ chính xác kiểm tra (Validation Accuracy) đạt đỉnh ở mức 86.17% trong lần huấn luyện đầu tiên, cho thấy mô hình có khả năng tổng quát tốt trên dữ liệu mới.
- Độ lỗi kiểm tra (Validation Loss) giảm ổn định qua từng epoch, xuống mức thấp nhất là 0.2727 trước khi kích hoạt early stopping.

Về hiệu năng mô hình, kiến trúc EfficientMiniNet có ưu điểm:

- Kích thước mô hình nhỏ, thuận lợi cho triển khai lên các thiết bị nhúng hoặc nền tảng tính toán giới hạn tài nguyên.

- Thời gian huấn luyện nhanh nhờ tối ưu cấu trúc mạng và sử dụng bộ tối ưu AdamW.
- Dễ dàng mở rộng hoặc điều chỉnh số lớp, số kênh phù hợp với các tập dữ liệu khác nhau.

Nhìn chung, EfficientMiniNet là một mô hình hiệu quả trong bài toán phân loại tín hiệu radar, đạt được sự cân bằng tốt giữa độ chính xác và chi phí tính toán. Các kết quả thử nghiệm cho thấy mô hình hoàn toàn có thể ứng dụng vào thực tế trong các hệ thống phân loại tín hiệu dựa trên ảnh radar hoặc tín hiệu vô tuyến.

CHƯƠNG 6: PHÂN TÍCH VÀ ĐÁNH GIÁ

6.1 PHÂN TÍCH ƯU – NHƯỢC ĐIỂM CỦA MÔ HÌNH

6.1.1 Ưu điểm

- **Kiến trúc mạng gọn nhẹ, tối ưu tài nguyên**: Mô hình EfficientMiniNet được thiết kế dựa trên kiến trúc CNN nhẹ, cụ thể là ứng dụng các lớp Depthwise Separable Convolution. Đây là một kỹ thuật đã được chứng minh là hiệu quả trong việc giảm số lượng tham số và độ phức tạp tính toán so với các lớp tích chập thông thường. Nhờ đó, tổng số tham số của mô hình luôn được duy trì dưới mức 300.000 – đúng với yêu cầu đề tài về tính gọn nhẹ, phù hợp với triển khai trên các hệ thống nhúng tài nguyên hạn chế.
- **Hiệu suất huấn luyện ổn định**: Trong quá trình thử nghiệm ba lần huấn luyện độc lập, mô hình cho kết quả ổn định với độ chính xác trung bình trên tập huấn luyện đạt khoảng 87.3% và trên tập kiểm thử đạt 78.6%. Điều này phản ánh khả năng học tập ổn định của mô hình trong điều kiện tập dữ liệu tương đối hạn chế về quy mô.
- **Tích hợp kết nối tắt (Residual connection)**: Một số block trong mô hình có sử dụng kết nối tắt nhằm duy trì thông tin đầu vào và ổn định gradient trong quá trình lan truyền ngược. Điều này hỗ trợ đáng kể trong việc cải thiện khả năng hội tụ và tránh hiện tượng mất mát thông tin ở các tầng sâu.
- **Khả năng chuyển đổi sang định dạng triển khai thực tế**: Mô hình sau huấn luyện đã được chuyển đổi thành định dạng TorchScript (22111001.pt). Đây là định dạng lý tưởng để triển khai trên các hệ thống nhúng như Raspberry Pi hoặc các thiết bị AI Edge, nhờ khả năng tối ưu hóa hiệu năng và tính tương thích cao với môi trường thực tế.

6.1.2 Nhược điểm

- **Chưa đạt yêu cầu độ chính xác kỳ vọng**: Mục tiêu ban đầu đặt ra là độ chính xác trên tập kiểm thử phải đạt tối thiểu 85%. Tuy nhiên, mô hình hiện tại chỉ đạt trung bình 78.6%, cho thấy còn tồn tại khoảng cách giữa thực tế và mục

tiêu đề ra, đặc biệt trong bối cảnh một số lớp điều chế có tín hiệu tương đồng cao.

- **Phân loại chưa rõ ràng giữa một số lớp tín hiệu**: Phân tích ma trận nhầm lẫn cho thấy mức độ nhầm lẫn tương đối cao giữa một số lớp điều chế như CPFSK và GFSK, vốn có đặc trưng phổ tương đối tương tự trong không gian thời gian – tần số. Điều này ảnh hưởng trực tiếp đến độ chính xác tổng thể của mô hình.
- **Chưa tích hợp các kỹ thuật nâng cao như Attention/Inception**: Mô hình hiện tại vẫn còn đơn giản, chưa khai thác các kỹ thuật nâng cao như cơ chế chú ý (Attention – ví dụ: SE, CBAM) hoặc khối trích đặc trưng đa tỉ lệ như Inception, vốn đã được chứng minh là hiệu quả trong các bài toán phân loại đặc trưng phức tạp.
- **Tiền xử lý ảnh còn chung chung**: Quá trình tăng cường dữ liệu (data augmentation) chủ yếu dừng lại ở các thao tác cơ bản như xoay, lật ngang/dọc. Những kỹ thuật này tuy giúp tăng tính đa dạng dữ liệu nhưng chưa thực sự khai thác bản chất không gian – tần số đặc trưng của ảnh radar. Việc không thực hiện các phép biến đổi chuyên sâu theo miền tần số hoặc thời gian khiến mô hình gặp khó khăn trong việc phân biệt các đặc trưng quan trọng.

6.2 ĐỀ XUẤT CẢI TIẾN TRONG TƯƠNG LAI

Để cải thiện hiệu suất mô hình và khắc phục các hạn chế nêu trên, nhóm nghiên cứu đề xuất một số hướng cải tiến trong tương lai như sau:

- **Bổ sung cơ chế Attention (SE/CBAM):** Việc tích hợp các module Squeeze-and-Excitation (SE) hoặc Convolutional Block Attention Module (CBAM) giúp mô hình học được sự chú ý vào các vùng quan trọng của ảnh. Điều này có thể cải thiện đáng kể hiệu suất phân loại, đặc biệt khi các lớp tín hiệu có sự chồng lắp đặc trưng.
- **Kết hợp mô hình lai (Hybrid CNN + ML):** Sau khi trích xuất đặc trưng từ CNN, có thể sử dụng các bộ phân loại truyền thống như Support Vector Machine (SVM) hoặc LightGBM thay vì lớp fully connected. Phương pháp

này giúp mô hình tận dụng tốt đặc trưng học sâu và vẫn đảm bảo khả năng phân loại hiệu quả với tập dữ liệu nhỏ.

- **Kỹ thuật tổ hợp mô hình (Ensemble Learning):** Việc kết hợp nhiều mô hình CNN nhẹ có kiến trúc khác nhau (bagging hoặc stacking) sẽ giúp tăng khả năng tổng quát hóa, giảm sai số và cải thiện độ chính xác cuối cùng.
- **Tăng cường dữ liệu có định hướng (Domain-specific augmentation):** Áp dụng các kỹ thuật như Gaussian Noise, Random Erasing, Spectral Masking hoặc biến đổi theo miền thời gian – tần số sẽ làm tăng khả năng mô hình nhận diện được các biến đổi thực tế của tín hiệu radar.
- **Mở rộng tập dữ liệu hoặc áp dụng Transfer Learning:** Việc mở rộng số lượng mẫu huấn luyện là một giải pháp quan trọng giúp cải thiện độ chính xác. Nếu không thể mở rộng dữ liệu thực, có thể xem xét áp dụng các mô hình đã được huấn luyện trước trên tập dữ liệu tương tự và tinh chỉnh lại (fine-tune) để tiết kiệm thời gian và cải thiện hiệu quả.

6.3 KẾT LUẬN CHUNG CỦA ĐỀ TÀI

Đề tài "Phân loại tín hiệu điều chế radar sử dụng ảnh thời gian – tần số và mô hình CNN nhẹ" đã đạt được các mục tiêu cơ bản ban đầu:

- Thiết kế thành công một mô hình CNN gọn nhẹ (EfficientMiniNet) với tổng số tham số nhỏ hơn 300.000, phù hợp cho triển khai trên các hệ thống nhúng.
- Kết quả huấn luyện đạt mức độ chính xác trung bình 87.3% trên tập huấn luyện và 78.6% trên tập kiểm thử, phản ánh khả năng học tập tương đối tốt trong điều kiện dữ liệu hạn chế.
- Mô hình đã được xuất ra định dạng TorchScript (.pt), sẵn sàng cho các ứng dụng thực tế.

Tuy nhiên, mô hình vẫn còn những điểm hạn chế cần cải thiện như chưa đạt được ngưỡng độ chính xác mong muốn, chưa tích hợp các kỹ thuật nâng cao như attention hay kiến trúc Inception, và kỹ thuật xử lý dữ liệu đầu vào còn tương đối đơn giản.

Tổng kết lại, đề tài đã hoàn thành phần lớn mục tiêu ban đầu và mở ra nhiều hướng phát triển sâu rộng hơn trong tương lai. Kết quả đạt được là tiền đề quan trọng để tiếp tục nghiên cứu các mô hình CNN tối ưu cho bài toán nhận dạng tín hiệu radar trong môi trường thực tế, góp phần nâng cao hiệu quả các hệ thống radar thông minh tự động.

TÀI LIỆU THAM KHẢO

[1] Trương Ngọc Sơn, Phạm Ngọc Sơn, Lê Minh, "*Huấn luyện mạng nơ ron học sâu*", NXB Đại học Quốc Gia, TPHCM, 2022.

[2] Trương Ngọc Sơn, "*Trí tuệ nhân tạo cơ sở và ứng dụng*", NXB Đại học Quốc Gia, TPHCM, 2020.

[3] Thien Huynh-The and et al, "Towards Waveform Classification in Integrated Radar-Communication Systems with Improved Accuracy and Reduced Complexity", *IEEE Internet of Things Journal*, 22 April 2024.

PHỤ LỤC

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import random_split, DataLoader
from torchvision import datasets, transforms
import pathlib

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ==========
# Dataset
# ==========
training_dir = pathlib.Path('/kaggle/input/radar-signal-
classification/training_set')

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

full_dataset = datasets.ImageFolder(root=str(training_dir),
transform=transform)
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [train_size,
val_size])

class_names = full_dataset.classes
num_classes = len(class_names)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

print(f"Classes: {class_names}")
print(f"Training samples: {train_size}, Validation samples: {val_size}")

# ==========
# Model
# ==========
class DepthwiseSeparableConv(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1,
residual=True):
        super().__init__()
        self.residual = residual and in_channels == out_channels and
stride == 1

        self.depthwise = nn.Conv2d(in_channels, in_channels,
kernel_size=3, stride=stride,
                                   padding=1, groups=in_channels,
bias=False)
        self.pointwise = nn.Conv2d(in_channels, out_channels,
kernel_size=1, bias=False)

        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
```

```
    def forward(self, x):
        identity = x
        out = self.depthwise(x)
        out = self.pointwise(out)
        out = self.bn(out)
        if self.residual:
            out += identity
        return self.relu(out)

class EfficientMiniNet(nn.Module):
    def __init__(self, num_classes=8):
        super().__init__()
        self.stem = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=2, padding=1,
bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True)
        )
        self.block1 = DepthwiseSeparableConv(32, 64, stride=1)
        self.block2 = DepthwiseSeparableConv(64, 128, stride=2)
        self.block3 = DepthwiseSeparableConv(128, 128, stride=1)
        self.block4 = DepthwiseSeparableConv(128, 256, stride=2)
        self.block5 = DepthwiseSeparableConv(256, 256, stride=1)
        self.block6 = DepthwiseSeparableConv(256, 512, stride=2)

        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.stem(x)
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.block6(x)
        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

# ==========
# Train Loop
# ==========
def train(model, train_loader, val_loader, criterion, optimizer,
epochs=20, patience=5):
    best_val_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        model.train()
        train_loss, correct, total = 0.0, 0, 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

        train_acc = correct / total
        train_loss = train_loss / len(train_loader)

        model.eval()
        val_loss, correct, total = 0.0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)

        val_acc = correct / total
        val_loss = val_loss / len(val_loader)

        print(f"Epoch {epoch+1}/{epochs} - "
              f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}
- "
              f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

        # Early stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), "best_model.pt")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered.")
                break

    return model

# ==========
# Train & Save
# ==========
model = EfficientMiniNet(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=0.001)

model = train(model, train_loader, val_loader, criterion, optimizer,
epochs=20)

example_input = torch.randn(1, 3, 128, 128).to(device)
traced_model = torch.jit.trace(model, example_input)
traced_model.save("22161116.pt")
print("Model saved as 22161116.pt")
```