

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐIỆN - ĐIỆN TỬ



ĐỒ ÁN CUỐI KỲ
MÔN HỌC: CƠ SỞ VÀ ỨNG DỤNG AI

**THIẾT KẾ HỆ THỐNG PHÂN LOẠI ẢNH BẰNG
PYTHON, SỬ DỤNG MẠNG HỌC SÂU**

Ngành: KỸ THUẬT MÁY TÍNH

GVHD: TS. Huỳnh Thế Thiện
Sinh viên: Trần Nguyễn Thành Tài
MSSV: 22119129
Sinh viên: Nguyễn Thị Ngọc Hân
MSSV: 22119067

TP. HỒ CHÍ MINH - 04/2025

Mục lục

Tóm tắt nội dung	4
1 Tổng Quan Về Học Sâu	4
1.1 Định nghĩa và vị trí trong lĩnh vực AI	4
1.2 Cách hoạt động của học sâu	4
1.2.1 Nền tảng cơ bản: Mạng nơ-ron nhân tạo	4
1.2.2 Quá trình lan truyền tiến (Forward Propagation)	5
1.2.3 Học tập và tối ưu hóa	5
1.2.4 Kỹ thuật xử lý các thách thức	6
1.2.5 Quá trình huấn luyện mô hình học sâu	6
1.3 Kiến trúc chính của học sâu	7
1.3.1 Mạng nơ-ron tích chập (CNN - Convolutional Neural Networks)	7
1.4 Ứng dụng của học sâu	8
2 Các thuật toán liên quan	9
2.1 Phép tích chập (Convolution)	9
2.2 Gộp tối đa (Max Pooling)	10
2.3 Gộp trung bình thích ứng (Adaptive Average Pooling)	10
2.4 Chuẩn hóa theo batch (Batch Normalization)	11
2.5 Hàm kích hoạt ReLU (Rectified Linear Unit)	11
2.6 Hàm mất mát Cross-Entropy	11
2.7 Tối ưu hóa AdamW	12
2.8 Lịch trình học OneCycleLR	13
2.9 Dropout	13
2.10 Huấn luyện với độ chính xác hỗn hợp (Mixed Precision Training)	14
3 Thiết kế kiến trúc mạng	16
3.1 Phân tích chi tiết sơ đồ kiến trúc mạng RadarSignalCNN	17
3.1.1 Cột 1: Bắt đầu xử lý hình ảnh (Block 1 và Block 2)	17
3.1.2 Cột 2: Xử lý sâu hơn (Block 2 tiếp tục và Block 3)	18
3.1.3 Cột 3: Phân loại (Classifier)	19
4 Lý do chọn thiết kế kiến trúc	19
5 Kết quả thực nghiệm	21
5.1 Tính toán các tham số	21
5.2 Kết quả huấn luyện	23

6	Phân tích và đánh giá kết quả	23
6.1	Phân tích chi tiết về tham số	23
6.2	Phân tích chi tiết về kết quả huấn luyện	24
6.3	Đánh giá tổng quan	24

Tóm tắt nội dung

Tóm tắt nội dung

Báo cáo này trình bày tổng quan về học sâu (deep learning), một nhánh quan trọng của học máy, tập trung vào các mạng nơ-ron nhân tạo với nhiều lớp ẩn. Nội dung bao gồm các thuật toán và kiến trúc được sử dụng trong bài toán phân loại tín hiệu radar, dựa trên mã nguồn PyTorch. Các công thức toán học được trình bày để giải thích cơ chế lan truyền tiến, lan truyền ngược, và tối ưu hóa. Báo cáo cũng đề cập đến ứng dụng thực tiễn trong phân loại tín hiệu radar, cùng các kỹ thuật giải quyết thách thức như overfitting. Cuối cùng, báo cáo cung cấp danh sách tài liệu tham khảo để hỗ trợ nghiên cứu sâu hơn.

1 Tổng Quan Về Học Sâu

1.1 Định nghĩa và vị trí trong lĩnh vực AI

Học sâu (Deep Learning) là một phân nhánh quan trọng của học máy (Machine Learning). Học sâu là một lĩnh vực con của học máy, sử dụng mạng nơ-ron nhân tạo với nhiều lớp ẩn để phân tích dữ liệu. Học sâu đặc biệt hiệu quả trong các nhiệm vụ phức tạp như nhận dạng hình ảnh và xử lý tín hiệu, bao gồm bài toán phân loại tín hiệu radar được triển khai trong mã nguồn.

1.2 Cách hoạt động của học sâu

Học sâu hoạt động thông qua các mạng nơ-ron nhân tạo với nhiều lớp ẩn để học các biểu diễn phức tạp từ dữ liệu. Dưới đây là các bước chi tiết:

1.2.1 Nền tảng cơ bản: Mạng nơ-ron nhân tạo

- **Cấu trúc nơ-ron nhân tạo:**
 - Đầu vào (Inputs): Nhận tín hiệu từ dữ liệu hoặc nơ-ron trước.
 - Trọng số (Weights): Quyết định mức độ ảnh hưởng của đầu vào.
 - Hàm tổng hợp (Summation function): Tính tổng có trọng số.
 - Độ lệch (Bias): Điều chỉnh ngưỡng kích hoạt.
 - Hàm kích hoạt (Activation function): Biến đổi đầu ra theo cách phi tuyến.
- **Tính toán tại mỗi nơ-ron:**
 - Tính tổng có trọng số:

$$z = \sum (w_i \times x_i) + b$$

- * z : Giá trị tổng hợp tại nơ-ron.
- * w_i : Trọng số tương ứng với đầu vào i .
- * x_i : Giá trị đầu vào thứ i .
- * b : Độ lệch (bias).
- Áp dụng hàm kích hoạt:

$$a = f(z)$$
 - * a : Giá trị đầu ra của nơ-ron sau khi kích hoạt.
 - * f : Hàm kích hoạt (ví dụ: ReLU, được sử dụng trong mã nguồn).
 - * z : Giá trị tổng hợp từ công thức trên.
- Hàm kích hoạt được sử dụng trong mã nguồn: ReLU (ReLU).

1.2.2 Quá trình lan truyền tiến (Forward Propagation)

- Lớp đầu vào: Nhận dữ liệu thô (hình ảnh radar kích thước $128 \times 128 \times 3$).
- Lớp ẩn: Thực hiện biến đổi phi tuyến:

$$a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$$

- $a^{(l)}$: Đầu ra của lớp thứ l .
- f : Hàm kích hoạt (ReLU trong mã nguồn).
- $W^{(l)}$: Ma trận trọng số của lớp thứ l .
- $a^{(l-1)}$: Đầu ra của lớp trước (lớp $l - 1$).
- $b^{(l)}$: Vector độ lệch của lớp thứ l .
- Lớp đầu ra: Tạo dự đoán cuối cùng (logits thô, sau đó được xử lý bởi hàm `CrossEntropyLoss` trong mã nguồn).

1.2.3 Học tập và tối ưu hóa

- **Hàm mất mát (Loss Function):**
 - Cross-Entropy với `label_smoothing=0.1`: Dùng cho bài toán phân loại (được sử dụng trong mã nguồn).
- **Lan truyền ngược (Backpropagation):**
 - Tính gradient của hàm mất mát.

- Cập nhật trọng số:

$$W = W - \eta \frac{\partial L}{\partial W}$$

- * W : Ma trận trọng số cần cập nhật.
 - * η : Tốc độ học (learning rate, 0.001 trong mã nguồn).
 - * $\frac{\partial L}{\partial W}$: Đạo hàm của hàm mất mát L theo W .
- Cập nhật độ lệch:

$$b = b - \eta \frac{\partial L}{\partial b}$$

- * b : Vector độ lệch cần cập nhật.
 - * η : Tốc độ học (learning rate).
 - * $\frac{\partial L}{\partial b}$: Đạo hàm của hàm mất mát L theo b .
- **Thuật toán tối ưu (Optimizers):**
 - AdamW với `weight_decay=1e-4` (được sử dụng trong mã nguồn).

1.2.4 Kỹ thuật xử lý các thách thức

- **Gradient biến mất/bùng nổ:**
 - Batch Normalization (`BatchNorm2d`, `BatchNorm1d`): Chuẩn hóa đầu ra để ổn định huấn luyện.
 - Mixed Precision Training: Sử dụng `GradScaler` để điều chỉnh gradient, tránh mất thông tin khi tính toán với FP16.
- **Overfitting:**
 - Dropout: Ngẫu nhiên vô hiệu hóa nơ-ron trong huấn luyện (`Dropout` với xác suất 0.2, 0.3, 0.5 ở các khối khác nhau).
 - Weight Decay: Điều chuẩn hóa L2 thông qua tối ưu hóa AdamW.
 - Label Smoothing: Sử dụng `label_smoothing=0.1` trong `CrossEntropyLoss` để làm mềm nhãn, giảm nguy cơ overfitting.

1.2.5 Quá trình huấn luyện mô hình học sâu

- Khởi tạo trọng số: Sử dụng khởi tạo Kaiming Normal (`kaiming_normal_`) cho các tầng tích chập và khởi tạo hằng số cho `BatchNorm`.
- Mini-batch training: Sử dụng batch size 64 (`batch_size=64` trong `DataLoader`).

- Epoch và Validation: Huấn luyện 50 epoch (theo tham số `epochs=50`), đánh giá trên tập xác thực sau mỗi epoch.
- Điều chỉnh lịch trình học: Sử dụng `OneCycleLR` với chiến lược cosine annealing, `max_lr=0.001`, và `steps_per_epoch` bằng độ dài của `train_loader`.
- Testing trên tập dữ liệu xác thực: Độ chính xác và mất mát được tính trên tập xác thực sau mỗi epoch.

Học sâu xây dựng các biểu diễn phân cấp từ dữ liệu thô, tự động trích xuất đặc trưng phức tạp, giúp giải quyết hiệu quả các bài toán như phân loại tín hiệu radar mà phương pháp truyền thống gặp giới hạn.

1.3 Kiến trúc chính của học sâu

1.3.1 Mạng nơ-ron tích chập (CNN - Convolutional Neural Networks)

Mạng nơ-ron tích chập (CNN) là kiến trúc duy nhất được sử dụng trong mã nguồn, cụ thể là mô hình `RadarSignalCNN` cho bài toán phân loại tín hiệu radar. CNN được thiết kế để xử lý dữ liệu hình ảnh, trích xuất đặc trưng không gian thông qua các lớp tích chập, chuẩn hóa, kích hoạt, gộp, và kết nối đầy đủ.

- **Cấu trúc mô hình:** Mô hình bao gồm hai phần chính: khối trích xuất đặc trưng (`features`) và khối phân loại (`classifier`).

– Khối trích xuất đặc trưng (`features`):

* Gồm 3 khối, mỗi khối chứa:

- Lớp tích chập (`Conv2d`) với bộ lọc 3×3 , padding 1 để giữ kích thước không gian, không sử dụng bias (`bias=False`).
- Lớp chuẩn hóa theo batch (`BatchNorm2d`) để ổn định huấn luyện.
- Hàm kích hoạt ReLU (`ReLU`, `inplace=True`) để thêm tính phi tuyến.
- Lớp gộp tối đa (`MaxPool2d`) với kích thước 2×2 , bước nhảy 2, để giảm kích thước không gian (trừ khối cuối).
- Dropout (`Dropout`) với xác suất 0.2 (Block 1), 0.3 (Block 2) để chống overfitting.

* Cụ thể:

- **Block 1:** 2 tầng `Conv2d` ($3 \rightarrow 32$), `MaxPool2d`, đầu ra: $32 \times 64 \times 64$.
- **Block 2:** 2 tầng `Conv2d` ($32 \rightarrow 64$), `MaxPool2d`, đầu ra: $64 \times 32 \times 32$.

- **Block 3:** 2 tầng `Conv2d` ($64 \rightarrow 120$), không có `MaxPool2d`, sử dụng `AdaptiveAvgPool2d` để giảm đầu ra về $120 \times 1 \times 1$.
 - * Số kênh tăng dần: $3 \text{ (RGB)} \rightarrow 32 \rightarrow 64 \rightarrow 120$.
- **Khối phân loại (classifier):**
 - * Làm phẳng (`Flatten`) để chuyển bản đồ đặc trưng thành vector 120 chiều.
 - * Lớp tuyến tính (`Linear`): $120 \rightarrow 120$.
 - * Chuẩn hóa theo batch (`BatchNorm1d`).
 - * Hàm kích hoạt ReLU (`ReLU, inplace=True`).
 - * Dropout (`Dropout`) với xác suất 0.5 để chống overfitting.
 - * Lớp tuyến tính cuối: $120 \rightarrow$ số lớp (8 trong bài toán).
- **Đặc điểm nổi bật:**
 - Mô hình nhẹ, với số lượng tham số được tính bằng `sum(p.numel())` trong mã nguồn, phù hợp cho môi trường tài nguyên hạn chế như Kaggle.
 - Sử dụng ReLU thay vì SiLU, tối ưu hóa tính toán và hiệu quả huấn luyện.
 - Kết hợp `BatchNorm2d`, `BatchNorm1d`, và `Dropout` để ổn định huấn luyện và chống overfitting.
 - `AdaptiveAvgPool2d` giúp mô hình linh hoạt với các kích thước đầu vào.
 - Sử dụng `bias=False` trong `Conv2d`, giảm tham số không cần thiết nhờ có `BatchNorm2d`.
- **Lan truyền tiến:** Dữ liệu đầu vào (hình ảnh $128 \times 128 \times 3$) đi qua các khối tích chập, giảm kích thước không gian ($128 \times 128 \rightarrow 64 \times 64 \rightarrow 32 \times 32 \rightarrow 1 \times 1$) và tăng số kênh ($3 \rightarrow 32 \rightarrow 64 \rightarrow 120$). Đầu ra là vector logits cho 8 lớp, được xử lý bởi hàm `CrossEntropyLoss` trong quá trình huấn luyện.

1.4 Ứng dụng của học sâu

- **Phân loại tín hiệu radar:** Mã nguồn áp dụng học sâu để phân loại tín hiệu radar, một bài toán thuộc lĩnh vực thị giác máy tính và xử lý tín hiệu. Dữ liệu đầu vào là hình ảnh biểu diễn tín hiệu radar (định dạng RGB), được tải từ thư mục `/kaggle/input/radar-signal-classification` bằng `ImageFolder`. Mô hình `RadarSignalCNN` phân loại hình ảnh vào 8 lớp tương ứng với các loại tín hiệu radar.
 - **Chuẩn bị dữ liệu:**

- * Tiền xử lý hình ảnh: Thay đổi kích thước về 128×128 , lật ngang ngẫu nhiên (`RandomHorizontalFlip`), xoay ngẫu nhiên 10 độ (`RandomRotation(10)`), điều chỉnh màu (`ColorJitter`), chuẩn hóa với trung bình $[0.5, 0.5, 0.5]$ và độ lệch chuẩn $[0.5, 0.5, 0.5]$.
- * Chia dữ liệu: 80% cho huấn luyện, 20% cho xác thực (`random_split`).
- * Sử dụng `DataLoader` với `batch_size=64`, `shuffle=True` cho huấn luyện, `pin_memory=True` và `num_workers=4` để tối ưu hóa chuyển dữ liệu sang GPU.
- **Huấn luyện:** Mô hình được huấn luyện trong 50 epoch, sử dụng hàm mất mát `CrossEntropyLoss` với `label_smoothing=0.1`, tối ưu hóa AdamW (`lr=0.001`, `weight_decay=1e-4`) và lịch trình học `OneCycleLR` với `max_lr=0.001`, `steps_per_epoch=len(train_loader)`, `epochs=50`, và chiến lược `anneal_strategy='cos'`. Độ chính xác và mất mát được đánh giá trên tập huấn luyện và xác thực sau mỗi epoch, với mô hình tốt nhất được lưu dưới dạng `best_model.pt`.
- **Kết quả:** Mô hình được lưu dưới dạng TorchScript (`22119129_22119067.pt`) để triển khai, phù hợp cho các ứng dụng thời gian thực như phát hiện và phân loại tín hiệu radar trong hệ thống quân sự, hàng không, hoặc giám sát giao thông.

2 Các thuật toán liên quan

Dưới đây là các thuật toán được sử dụng trong mã nguồn, kèm công thức, chú thích, và giải thích chi tiết.

2.1 Phép tích chập (Convolution)

Phép tích chập trích xuất đặc trưng không gian từ hình ảnh đầu vào bằng cách áp dụng các bộ lọc (kernel).

$$(I * K)(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(x+m, y+n) \cdot K(m, n)$$

- I : Ảnh đầu vào (ma trận pixel, kích thước $H \times W \times C$).
- K : Kernel (bộ lọc tích chập, kích thước $M \times N$).
- (x, y) : Tọa độ pixel trong ảnh đầu ra.
- M, N : Kích thước kernel (3×3 trong mã nguồn).
- $I(x+m, y+n)$: Giá trị pixel tại $(x+m, y+n)$.
- $K(m, n)$: Giá trị kernel tại (m, n) .

Giải thích: Trong mã nguồn, `Conv2d` áp dụng phép tích chập với kernel 3×3 , padding 1 để giữ kích thước không gian, và `bias=False`. Mỗi lớp tích chập tăng số kênh ($3 \rightarrow 32$, $32 \rightarrow 64$, $64 \rightarrow 120$), trích xuất các đặc trưng như cạnh, góc, hoặc mẫu phức tạp hơn. Padding đảm bảo đầu ra có cùng kích thước không gian với đầu vào, trừ khi có gộp.

2.2 Gộp tối đa (Max Pooling)

Gộp tối đa giảm kích thước không gian bằng cách lấy giá trị lớn nhất trong các vùng con.

$$\text{Pool}(I)(x, y) = \max_{0 \leq m, n < S} I(x \cdot S + m, y \cdot S + n)$$

- I : Ảnh đầu vào.
- (x, y) : Tọa độ pixel trong ảnh đầu ra.
- S : Kích thước vùng gộp (2×2 trong mã nguồn).
- m, n : Chỉ số trong vùng gộp.
- $I(x \cdot S + m, y \cdot S + n)$: Giá trị pixel trong vùng gộp.

Giải thích: Lớp `MaxPool2d` với kích thước 2×2 và bước nhảy 2 giảm kích thước không gian xuống một nửa (ví dụ, $128 \times 128 \rightarrow 64 \times 64$). Điều này giảm số lượng tham số, tăng trường tiếp nhận (receptive field), và giữ lại các đặc trưng nổi bật.

2.3 Gộp trung bình thích ứng (Adaptive Average Pooling)

Gộp trung bình thích ứng giảm kích thước đầu ra về một kích thước cố định (1×1).

$$\text{AdaptiveAvgPool}(I)(x, y) = \frac{1}{H' \cdot W'} \sum_{i=0}^{H'-1} \sum_{j=0}^{W'-1} I(i, j)$$

- I : Ảnh đầu vào (kích thước $H' \times W' \times C$).
- (x, y) : Tọa độ đầu ra (1×1 trong mã nguồn).
- H', W' : Kích thước không gian của ảnh đầu vào.
- $I(i, j)$: Giá trị pixel tại (i, j) .

Giải thích: Lớp `AdaptiveAvgPool2d` trong khối cuối giảm bản đồ đặc trưng về 1×1 , tạo vector 120 chiều (số kênh). Điều này chuẩn bị dữ liệu cho lớp phân loại và đảm bảo mô hình linh hoạt với các kích thước đầu vào.

2.4 Chuẩn hóa theo batch (Batch Normalization)

Chuẩn hóa theo batch chuẩn hóa đầu ra của lớp trước để ổn định huấn luyện.

$$y_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

- x_i : Giá trị đầu vào của mẫu i trong batch.
- μ_B : Trung bình của batch, $\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$.
- σ_B^2 : Phương sai của batch, $\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$.
- ϵ : Hằng số nhỏ để tránh chia cho 0 (thường 10^{-5}).
- γ : Tham số tỷ lệ (học được).
- β : Tham số dịch chuyển (học được).
- y_i : Giá trị đầu ra sau chuẩn hóa.
- m : Kích thước batch (64 trong mã nguồn).

Giải thích: BatchNorm2d và BatchNorm1d được áp dụng sau các lớp tích chập và tuyến tính, chuẩn hóa bản đồ đặc trưng hoặc vector theo từng kênh hoặc chiều. Điều này giảm hiện tượng lệch phân phối nội bộ (internal covariate shift), tăng tốc huấn luyện, và cải thiện độ ổn định.

2.5 Hàm kích hoạt ReLU (Rectified Linear Unit)

Hàm ReLU áp dụng tính phi tuyến bằng cách giữ nguyên các giá trị dương và đặt các giá trị âm về 0.

$$\text{ReLU}(x) = \max(0, x)$$

- x : Giá trị đầu vào.

Giải thích: ReLU được sử dụng sau mỗi lớp tích chập và trong lớp phân loại (ReLU, inplace=True). ReLU đơn giản, hiệu quả, và giúp giảm hiện tượng gradient biến mất so với các hàm như Sigmoid, cải thiện hiệu quả huấn luyện trong các mạng sâu.

2.6 Hàm mất mát Cross-Entropy

Hàm mất mát Cross-Entropy đo lường sự khác biệt giữa phân phối xác suất dự đoán và nhãn thực tế.

$$L = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \log(\hat{y}_{i,c})$$

- L : Giá trị mất mát.
- N : Số mẫu trong batch (64 trong mã nguồn).
- C : Số lớp (8 trong bài toán).
- $y_{i,c}$: Nhãn thực tế (1 nếu mẫu i thuộc lớp c , 0 nếu không).
- $\hat{y}_{i,c}$: Xác suất dự đoán cho mẫu i , lớp c , được tính bằng $\hat{y}_{i,c} = \text{softmax}(z_{i,c}) = \frac{e^{z_{i,c}}}{\sum_{k=1}^C e^{z_{i,k}}}$, trong đó $z_{i,c}$ là logit đầu ra của mô hình.

Giải thích: `CrossEntropyLoss` với `label_smoothing=0.1` được sử dụng để huấn luyện mô hình. `CrossEntropyLoss` tự động áp dụng softmax lên logits đầu ra và tính mất mát, kết hợp với label smoothing để làm mềm nhãn, giảm nguy cơ overfitting.

2.7 Tối ưu hóa AdamW

AdamW là biến thể của Adam, thêm weight decay để điều chuẩn hóa.

- Cập nhật trung bình động của gradient:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L$$

- Cập nhật trung bình động của bình phương gradient:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L)^2$$

- Sửa lệch thiên kiến:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Cập nhật trọng số:

$$w_t = w_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \eta \lambda w_{t-1}$$

- m_t : Trung bình động của gradient tại bước t .
- β_1 : Hệ số suy giảm (thường 0.9).
- ∇L : Gradient của hàm mất mát.
- v_t : Trung bình động của bình phương gradient.

- β_2 : Hệ số suy giảm (thường 0.999).
- \hat{m}_t, \hat{v}_t : Giá trị sau sửa lệch.
- w_t : Trọng số tại bước t .
- η : Tốc độ học (0.001 trong mã nguồn).
- ϵ : Hằng số nhỏ (10^{-8}).
- λ : Hệ số weight decay (1×10^{-4} trong mã nguồn).

Giải thích: AdamW được sử dụng với $\eta = 0.001$ và $\lambda = 10^{-4}$. Weight decay thêm điều chuẩn L2, giảm nguy cơ overfitting. AdamW cải thiện hiệu quả so với Adam bằng cách tách biệt weight decay khỏi cập nhật gradient. Trong huấn luyện với độ chính xác hỗn hợp, AdamW hoạt động cùng GradScaler để điều chỉnh gradient.

2.8 Lịch trình học OneCycleLR

OneCycleLR điều chỉnh tốc độ học theo chiến lược cosine annealing.

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos \left(\frac{t}{T_{\text{total}}} \pi \right) \right)$$

- η_t : Tốc độ học tại bước t .
- η_{\max} : Tốc độ học tối đa (0.001 trong mã nguồn).
- η_{\min} : Tốc độ học tối thiểu (0 trong mã nguồn).
- t : Bước hiện tại.
- T_{total} : Tổng số bước, được tính bằng `steps_per_epoch × epochs` (`len(train_loader) × 50`).

Giải thích: OneCycleLR điều chỉnh tốc độ học từ 0 lên `max_lr=0.001` và giảm dần về 0 theo hàm cosin. Chiến lược này giúp mô hình hội tụ nhanh hơn và đạt hiệu suất tốt hơn.

2.9 Dropout

Dropout ngẫu nhiên vô hiệu hóa các nơ-ron để chống overfitting.

$$y_i = \begin{cases} 0 & \text{với xác suất } p \\ \frac{x_i}{1-p} & \text{với xác suất } 1-p \end{cases}$$

- x_i : Giá trị đầu vào của nơ-ron i .
- y_i : Giá trị đầu ra sau dropout.

- p : Xác suất dropout (0.2, 0.3, 0.5 trong các khối khác nhau).
- $\frac{x_i}{1-p}$: Tỷ lệ để duy trì kỳ vọng đầu ra.

Giải thích: Dropout được áp dụng trong các khối trích xuất đặc trưng ($p=0.2$ ở Block 1, $p=0.3$ ở Block 2) và trong lớp phân loại ($p=0.5$), vô hiệu hóa các nơ-ron trong quá trình huấn luyện để tăng tính tổng quát hóa.

2.10 Huấn luyện với độ chính xác hỗn hợp (Mixed Precision Training)

Huấn luyện với độ chính xác hỗn hợp sử dụng cả số thực 16-bit (FP16) và 32-bit (FP32) để tăng tốc và giảm sử dụng bộ nhớ.

- Tính toán lan truyền tiến và ngược trong FP16:

$$\text{output}_{\text{FP16}} = \text{model}_{\text{FP16}}(x_{\text{FP16}})$$

- Tính mất mát:

$$L_{\text{FP16}} = \text{criterion}(\text{output}_{\text{FP16}}, y)$$

- Nhân gradient với hệ số tỷ lệ:

$$L_{\text{scaled}} = s \cdot L_{\text{FP16}}$$

- Cập nhật trọng số trong FP32:

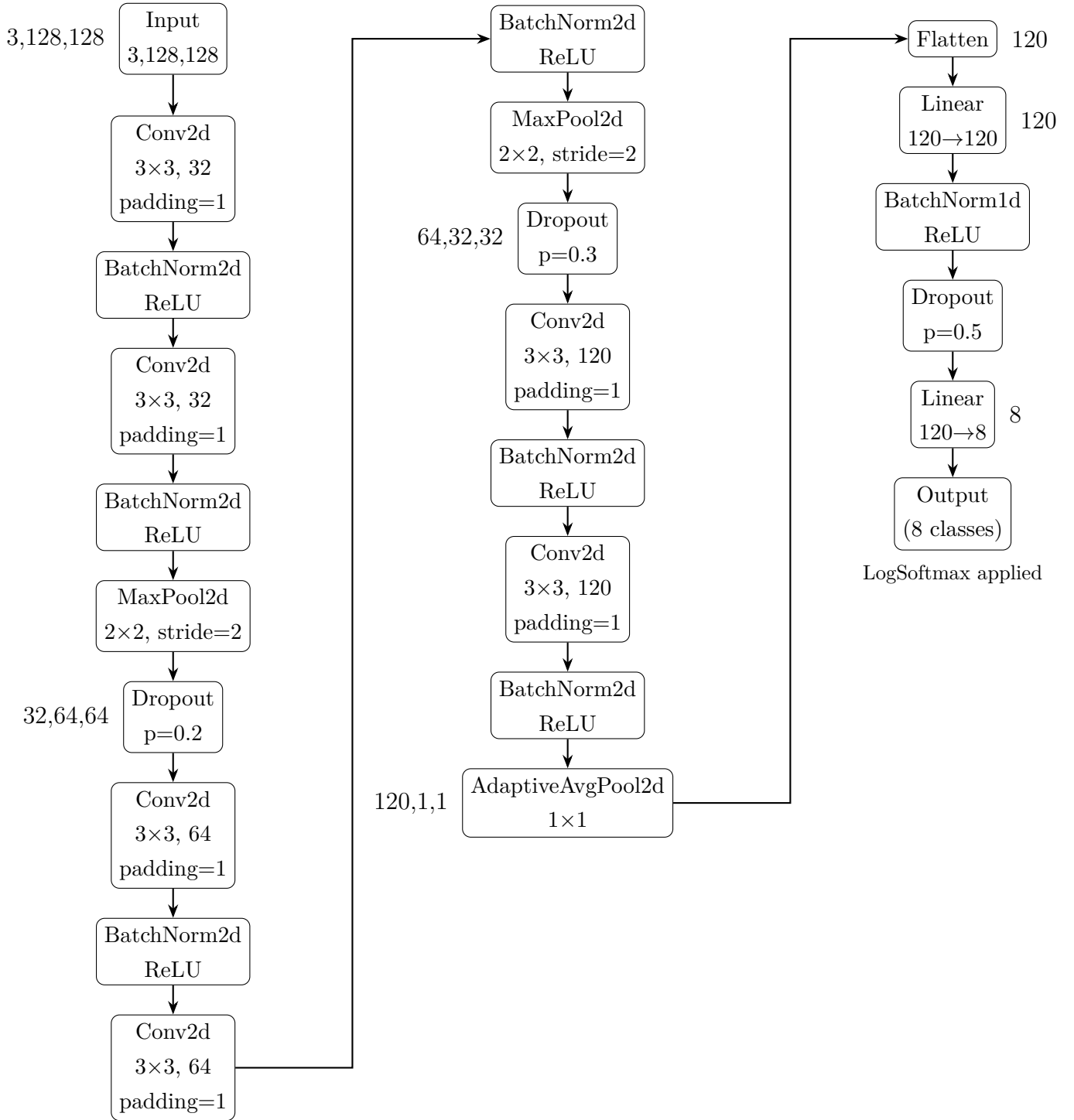
$$w_{\text{FP32}} = w_{\text{FP32}} - \eta \cdot \nabla L_{\text{FP32}}$$

- x_{FP16} : Dữ liệu đầu vào ở FP16.
- $\text{output}_{\text{FP16}}$: Đầu ra mô hình ở FP16.
- L_{FP16} : Mất mát ở FP16.
- s : Hệ số tỷ lệ (động, do `GradScaler` quản lý).
- w_{FP32} : Trọng số ở FP32.
- ∇L_{FP32} : Gradient chuyển về FP32.
- η : Tốc độ học.

Giải thích: `torch.cuda.amp` với `autocast` và `GradScaler` được sử dụng để huấn luyện với độ chính xác hỗn hợp. `autocast` tự động chọn FP16 cho các phép tính phù hợp, trong khi

GradScaler ngăn mất thông tin gradient bằng cách nhân với hệ số tỷ lệ trước khi lan truyền ngược. Điều này giảm sử dụng bộ nhớ GPU và tăng tốc huấn luyện, đặc biệt trong môi trường Kaggle.

3 Thiết kế kiến trúc mạng



Hình 1: Sơ đồ kiến trúc của mô hình RadarSignalCNN.

3.1 Phân tích chi tiết sơ đồ kiến trúc mạng RadarSignalCNN

Sơ đồ kiến trúc của mô hình RadarSignalCNN minh họa cách dữ liệu hình ảnh radar (kích thước 128×128 , 3 kênh màu RGB) được xử lý qua các tầng để phân loại thành 8 loại tín hiệu khác nhau. Sơ đồ được chia thành ba cột, đại diện cho ba giai đoạn xử lý: trích xuất đặc trưng (cột 1 và 2) và phân loại (cột 3). Mỗi cột chứa các tầng cụ thể, với kích thước đầu ra được ghi rõ để theo dõi sự thay đổi của dữ liệu. Dưới đây là phân tích chi tiết từng cột, kèm công thức tính toán kích thước đầu ra.

3.1.1 Cột 1: Bắt đầu xử lý hình ảnh (Block 1 và Block 2)

Cột đầu tiên mô tả hai khối xử lý đầu tiên trong phần **features** của mô hình, nơi hình ảnh radar được quét để tìm các đặc trưng cơ bản, như đường nét, vùng sáng tối, hoặc mẫu tín hiệu đơn giản. Đây là giai đoạn khởi đầu, biến đổi hình ảnh thô thành các bản đồ đặc trưng có ý nghĩa.

- **Input (Đầu vào):** Hình ảnh radar ban đầu có 3 kênh màu (RGB), kích thước 128×128 pixel. Trong ngữ cảnh tín hiệu radar, hình ảnh này có thể là spectrogram (biểu diễn thời gian-tần số) hoặc một dạng biểu đồ khác, chứa thông tin về biên độ, tần số, hoặc nhiễu. Kích thước được biểu diễn là $(C, H, W) = (3, 128, 128)$, trong đó C : số kênh (3 kênh màu), H : chiều cao (128 pixel), W : chiều rộng (128 pixel). Đầu vào cung cấp dữ liệu thô, là điểm khởi đầu để mô hình phân tích tín hiệu radar. Không có tính toán vì đây là đầu vào gốc, kích thước cố định $(3, 128, 128)$.

- **Block 1:**

- *Các tầng xử lý:*

- * **Conv2d (3x3, 32, padding=1)** (lần 1): Tầng tích chập sử dụng bộ lọc kích thước 3×3 , biến đổi 3 kênh màu đầu vào thành 32 bản đồ đặc trưng. **Padding=1** giữ nguyên kích thước không gian (128×128), và **bias=False** vì **BatchNorm2d** đã chuẩn hóa dữ liệu. Công thức tính kích thước đầu ra: $H_{\text{out}} = \left\lfloor \frac{128+2 \times 1-3}{1} + 1 \right\rfloor = 128$, $W_{\text{out}} = 128$. Kích thước: $(32, 128, 128)$.
 - * **BatchNorm2d (32) + ReLU**: Chuẩn hóa 32 bản đồ đặc trưng và áp dụng **ReLU** ($f(x) = \max(0, x)$) để chọn lọc đặc trưng. Kích thước không đổi: $(32, 128, 128)$.
 - * **Conv2d (3x3, 32, padding=1)** (lần 2): Tiếp tục quét 32 bản đồ đặc trưng để tìm các mẫu phức tạp hơn. Kích thước không đổi: $(32, 128, 128)$.
 - * **BatchNorm2d (32) + ReLU**: Chuẩn hóa và kích hoạt. Kích thước: $(32, 128, 128)$.
 - * **MaxPool2d (2x2, stride=2)**: Giảm kích thước từ 128×128 xuống 64×64 ($H_{\text{out}} = \left\lfloor \frac{128}{2} \right\rfloor = 64$). Kích thước: $(32, 64, 64)$.

- * Dropout ($p=0.2$): Ngẫu nhiên bỏ 20% kết nối giữa các nơ-ron. Kích thước: (32, 64, 64).
- Ý nghĩa tổng thể của Block 1: Từ (3, 128, 128), Block 1 tạo ra 32 bản đồ đặc trưng, giảm kích thước xuống (32, 64, 64), tập trung vào các đặc trưng cơ bản của tín hiệu radar, như biên độ hoặc mẫu tần số đơn giản.
- **Block 2:**
 - Các tầng xử lý:
 - * Conv2d (3x3, 64, padding=1) (lần 1): Quét 32 bản đồ đặc trưng, tạo ra 64 bản đồ mới. Padding=1, bias=False. Kích thước: (64, 64, 64).
 - * BatchNorm2d (64) + ReLU: Chuẩn hóa và kích hoạt. Kích thước: (64, 64, 64).
 - * Conv2d (3x3, 64, padding=1) (lần 2): Tiếp tục quét 64 bản đồ đặc trưng. Kích thước: (64, 64, 64).

3.1.2 Cột 2: Xử lý sâu hơn (Block 2 tiếp tục và Block 3)

Cột thứ hai tiếp tục phân features, tập trung vào việc trích xuất các đặc trưng cấp cao hơn và nén dữ liệu thành một vector nhỏ.

- **Block 2 (tiếp tục):**
 - Các tầng xử lý:
 - * BatchNorm2d (64) + ReLU: Chuẩn hóa và kích hoạt. Kích thước: (64, 64, 64).
 - * MaxPool2d (2x2, stride=2): Giảm kích thước từ 64×64 xuống 32×32 . Kích thước: (64, 32, 32).
 - * Dropout ($p=0.3$): Ngẫu nhiên bỏ 30% kết nối. Kích thước: (64, 32, 32).
 - Ý nghĩa tổng thể của Block 2: Block 2 tăng số lượng đặc trưng lên 64, tìm các mẫu tín hiệu phức tạp hơn, giảm kích thước xuống (64, 32, 32).
- **Block 3:**
 - Các tầng xử lý:
 - * Conv2d (3x3, 120, padding=1) (lần 1): Quét 64 bản đồ đặc trưng, tạo ra 120 bản đồ mới. Kích thước: (120, 32, 32).
 - * BatchNorm2d (120) + ReLU: Chuẩn hóa và kích hoạt. Kích thước: (120, 32, 32).
 - * Conv2d (3x3, 120, padding=1) (lần 2): Tiếp tục quét 120 bản đồ đặc trưng. Kích thước: (120, 32, 32).

- * **BatchNorm2d (120) + ReLU**: Chuẩn hóa và kích hoạt. Kích thước: (120, 32, 32).
- * **AdaptiveAvgPool2d (1x1)**: Nén mỗi bản đồ đặc trưng thành 1×1 . Kích thước: (120, 1, 1).
- *Ý nghĩa tổng thể của Block 3*: Block 3 tạo ra 120 bản đồ đặc trưng, nén thành vector 120 số, sẵn sàng cho phân loại.

3.1.3 Cột 3: Phân loại (Classifier)

Cột thứ ba mô tả phần **classifier**, nơi vector đặc trưng (120 số) được xử lý để dự đoán loại tín hiệu radar (1 trong 8 lớp).

- **Flatten**: Chuyển ma trận (120, 1, 1) thành vector 120 số. Kích thước: 120.
- **Linear 1**: Ánh xạ vector 120 chiều về chính nó. Kích thước: 120.
- **BatchNorm1d + ReLU + Dropout**: Chuẩn hóa, kích hoạt, và bỏ 50% kết nối. Kích thước: 120.
- **Linear 2**: Ánh xạ vector 120 chiều thành 8 chiều, tương ứng với 8 lớp tín hiệu radar. Kích thước: 8.
- **Output**: Vector 8 chiều (logits thô) được xử lý bởi **log_softmax** để tạo phân phối xác suất log, trước khi áp dụng **CrossEntropyLoss** để tính mất mát.

4 Lý do chọn thiết kế kiến trúc

Trong quá trình thiết kế mô hình cho bài toán phân loại hình ảnh, việc lựa chọn kiến trúc mạng phải đảm bảo cân đối nhiều yếu tố: khả năng học đặc trưng mạnh mẽ, tốc độ huấn luyện nhanh, khả năng tổng quát tốt, đồng thời giới hạn tài nguyên tính toán để thuận tiện cho việc triển khai thực tế.

Kiến trúc được đề xuất để đáp ứng hiệu quả những yêu cầu này thông qua việc kế thừa tinh thần thiết kế từ các mô hình CNN kinh điển (chẳng hạn như VGG), đồng thời có những cải tiến hợp lý nhằm tối ưu cho bài toán cụ thể.

Thứ nhất, mô hình sử dụng ý tưởng từ kiến trúc của mô hình VGGNet (VGG-16/VGG-19), một trong những mạng CNN nổi tiếng với cách sử dụng các khối convolution liên tiếp nhỏ (3x3) để học các đặc trưng ở nhiều cấp độ khác nhau.

Trong kiến trúc VGGNet, các khối convolution được chồng lên nhau để tăng độ sâu của mạng mà không làm tăng quá nhiều tham số, giúp mô hình có khả năng học được các đặc trưng phức tạp của dữ liệu.

Sự kế thừa này thể hiện rõ ở việc sử dụng hai lớp convolution nhỏ 3x3 liên tiếp (Conv2D + BatchNorm + ReLU) trong mỗi block, thay vì tăng kích thước kernel quá nhanh như một số

kiến trúc khác. Cách tiếp cận này không chỉ giảm số lượng tham số mà còn tạo ra các đặc trưng mạnh mẽ hơn nhờ khả năng học theo nhiều tầng. Đồng thời, việc sử dụng các khối MaxPooling sau mỗi cặp lớp convolution giúp giảm độ phân giải không gian của ảnh, đồng thời tăng cường khả năng nhận diện các đặc trưng ở các mức độ trừu tượng khác nhau, giống như cách làm của VGGNet.

Thứ hai, việc sử dụng các kỹ thuật hiện đại giúp cho mô hình cải thiện hiệu suất tổng quát. Mô hình hiện tại không chỉ sử dụng kiến trúc đơn giản và hiệu quả mà còn ứng dụng các kỹ thuật tiên tiến trong huấn luyện nhằm nâng cao khả năng tổng quát của mạng. Một trong những kỹ thuật quan trọng được sử dụng đó là Batch Normalization. Kỹ thuật này giúp làm giảm những vấn đề như vanishing và exploding gradients. Đây không chỉ là kỹ thuật giúp ổn định cho mô hình mà còn cho phép sử dụng learning rate lớn hơn mà không gặp mất mát trong quá trình học của mô hình trên.

Mô hình còn sử dụng Dropout trong các khối convolution và fully connected nhằm giảm thiểu tình trạng overfitting, giúp mô hình học được những đặc trưng quan trọng nhưng không bị lệ thuộc quá nhiều vào một số đặc trưng nhất định. Ngoài ra, kỹ thuật Label Smoothing cũng được ứng dụng trong mô hình trên (hàm loss) nhằm để giúp tổng quát tốt trong quá trình học, giúp phân phối xác suất dự đoán tốt hơn.

Việc tích hợp những kỹ thuật trên mà mô hình không chỉ có khả năng học tốt ở các đặc trưng mà còn có thể tổng quát một cách tốt hơn, mang lại hiệu suất tốt hơn trên tập validation và giảm thiểu overfitting khi làm việc với dữ liệu có sự biến thiên lớn.

Thứ ba, một mô hình khi được tạo ra cũng phải đảm bảo được tính linh hoạt. Tính linh hoạt đó thể hiện ở chỗ khi đưa các ảnh có kích thước khác nhau vào thì mô hình sẽ có thể xử lý một cách trơn tru và ta không cần phải can thiệp vào cấu trúc bên trong của mạng. Đây chính là một điểm vô cùng quan trọng trong việc xây dựng mô hình phân loại hình ảnh, giúp xử lý một cách linh hoạt trong bất kỳ hoàn cảnh nào mà không gặp khó khăn. Bằng cách sử dụng Adaptive Average Pooling ở cuối các khối tích chập (convolution) đã làm cho mô hình không còn tình trạng cứng nhắc và thiếu linh động khi chỉ xử lý một kích thước cố định. Đây là điều cần phải làm, đặc biệt khi xây dựng mô hình ứng dụng cho phân loại hình ảnh mà ta đang thực hiện, ứng dụng được trong nhiều môi trường đa dạng khác nhau yêu cầu xử lý đa dạng các kích thước ảnh khác nhau, giảm gánh nặng tiền xử lý dữ liệu.

Thứ tư, việc đảm bảo tốt giữa độ phức tạp cũng như chi phí tính toán là một điểm vô cùng cân nhắc khi được yêu cầu phải thực hiện đối với tài nguyên hạn chế. Khi thiết kế, ta cần phải quan tâm tới việc lựa chọn số lượng convolution, số lượng kênh đầu ra cũng như tổng các tham số nhằm để giúp cho mô hình vừa được tối ưu hóa hiệu suất trong quá trình thực thi và tính toán, vừa tiết kiệm được bộ nhớ.

5 Kết quả thực nghiệm

5.1 Tính toán các tham số

Bảng 1: Chi tiết số tham số của mô hình RadarSignalCNN

Layer	Loại	Số tham số	Cách tính
Conv2d(3, 32, 3x3)	Convolution	864	$3 \times 32 \times 3 \times 3 = 864$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(32)	BatchNorm	64	$32 \times 2 = 64$ (num_features \times 2 cho weight và bias)
Conv2d(32, 32, 3x3)	Convolution	9216	$32 \times 32 \times 3 \times 3 = 9216$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(32)	BatchNorm	64	$32 \times 2 = 64$ (num_features \times 2 cho weight và bias)
MaxPool2d(2, 2)	Pooling	0	Không có tham số học được
Dropout(0.2)	Dropout	0	Không có tham số học được
Conv2d(32, 64, 3x3)	Convolution	18432	$32 \times 64 \times 3 \times 3 = 18432$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(64)	BatchNorm	128	$64 \times 2 = 128$ (num_features \times 2 cho weight và bias)
Conv2d(64, 64, 3x3)	Convolution	36864	$64 \times 64 \times 3 \times 3 = 36864$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(64)	BatchNorm	128	$64 \times 2 = 128$ (num_features \times 2 cho weight và bias)
MaxPool2d(2, 2)	Pooling	0	Không có tham số học được
Dropout(0.3)	Dropout	0	Không có tham số học được
Conv2d(64, 120, 3x3)	Convolution	69120	$64 \times 120 \times 3 \times 3 = 69120$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(120)	BatchNorm	240	$120 \times 2 = 240$ (num_features \times 2 cho weight và bias)
Conv2d(120, 120, 3x3)	Convolution	129600	$120 \times 120 \times 3 \times 3 = 129600$ (in_channels \times out_channels \times kernel_size ² , bias=False)
BatchNorm2d(120)	BatchNorm	240	$120 \times 2 = 240$ (num_features \times 2 cho weight và bias)
AdaptiveAvgPool2d((1, 1))	Pooling	0	Không có tham số học được
Flatten	Flatten	0	Không có tham số học được
Linear(120, 120)	Fully Connected	14520	$120 \times 120 + 120 = 14400 + 120 = 14520$ (in_features \times out_features + bias)
BatchNorm1d(120)	BatchNorm	240	$120 \times 2 = 240$ (num_features \times 2 cho weight và bias)
Dropout(0.5)	Dropout	0	Không có tham số học được
Linear(120, 8)	Fully Connected	968	$120 \times 8 + 8 = 960 + 8 = 968$ (in_features \times out_features + bias)
Tổng cộng		280688	Tổng của tất cả các tham số trên

Các công thức tính số tham số cho từng lớp như sau:

$$\text{Parameters} = K^2 \times C_{\text{in}} \times C_{\text{out}} + C_{\text{out}} \quad (\text{bias}) \quad (1)$$

Chú thích:

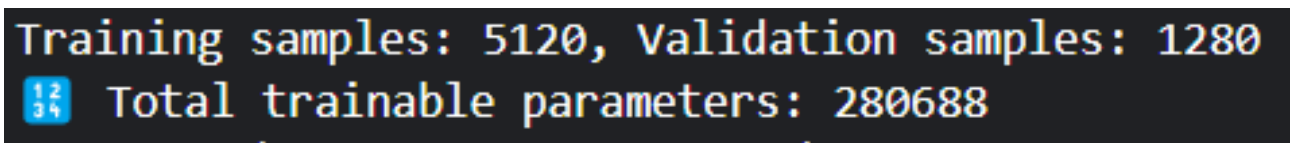
- K: Kích thước Kernel
- Cin : Số lượng kênh ngõ vào
- Cout : Số lượng kênh ngõ ra (số lượng bộ lọc)

$$\text{Số tham số BatchNorm} = 2 \times \text{Số lượng kênh} \quad (\gamma + \beta) \quad (2)$$

$$\text{Fully Connected (output vector)} = \text{Weight} \times \text{input vector} + \text{bias} \quad (3)$$

Chú thích:

- x : là vector ngõ vào (hay input features: từ flattening layer)
- W : là ma trận trọng số
- b : là bias
- y: là vector ngõ ra



```
Training samples: 5120, Validation samples: 1280
12 Total trainable parameters: 280688
34
```

Hình 2: Số lượng mẫu huấn luyện, mẫu xác thực và tổng số tham số của mô hình đã huấn luyện.

Kết quả tính từ bảng trên cho thấy đã trùng khớp so với khi ta tiến hành chạy thử và huấn luyện mô hình thực tế.

5.2 Kết quả huấn luyện

Epoch 19	Train Loss: 0.8043	Train Acc: 83.30%	Val Loss: 0.8005	Val Acc: 85.23%	Time: 7.94s	LR: 0.000968
Epoch 20	Train Loss: 0.7969	Train Acc: 83.40%	Val Loss: 0.7572	Val Acc: 86.88%	Time: 8.36s	LR: 0.000950
Epoch 21	Train Loss: 0.7965	Train Acc: 83.24%	Val Loss: 0.7734	Val Acc: 84.45%	Time: 7.90s	LR: 0.000929
Epoch 22	Train Loss: 0.7916	Train Acc: 83.91%	Val Loss: 0.8529	Val Acc: 81.56%	Time: 8.07s	LR: 0.000904
Epoch 23	Train Loss: 0.7799	Train Acc: 84.41%	Val Loss: 0.7519	Val Acc: 86.02%	Time: 8.05s	LR: 0.000876
Epoch 24	Train Loss: 0.7732	Train Acc: 84.12%	Val Loss: 0.7440	Val Acc: 85.55%	Time: 8.64s	LR: 0.000845
Epoch 25	Train Loss: 0.7705	Train Acc: 84.94%	Val Loss: 0.7767	Val Acc: 84.38%	Time: 8.06s	LR: 0.000811
Epoch 26	Train Loss: 0.7709	Train Acc: 85.00%	Val Loss: 0.7343	Val Acc: 85.70%	Time: 8.06s	LR: 0.000775
Epoch 27	Train Loss: 0.7668	Train Acc: 84.69%	Val Loss: 0.7614	Val Acc: 86.02%	Time: 7.97s	LR: 0.000736
Epoch 28	Train Loss: 0.7656	Train Acc: 84.92%	Val Loss: 0.7593	Val Acc: 86.95%	Time: 8.48s	LR: 0.000696
Epoch 29	Train Loss: 0.7674	Train Acc: 85.29%	Val Loss: 0.8174	Val Acc: 82.50%	Time: 8.11s	LR: 0.000654
Epoch 30	Train Loss: 0.7495	Train Acc: 86.19%	Val Loss: 0.7722	Val Acc: 83.05%	Time: 8.22s	LR: 0.000611
Epoch 31	Train Loss: 0.7535	Train Acc: 85.25%	Val Loss: 0.7619	Val Acc: 85.00%	Time: 8.15s	LR: 0.000567
Epoch 32	Train Loss: 0.7494	Train Acc: 85.74%	Val Loss: 0.7279	Val Acc: 88.44%	Time: 8.53s	LR: 0.000522
Epoch 33	Train Loss: 0.7465	Train Acc: 86.04%	Val Loss: 0.7473	Val Acc: 84.45%	Time: 7.90s	LR: 0.000477
Epoch 34	Train Loss: 0.7471	Train Acc: 86.19%	Val Loss: 0.7490	Val Acc: 85.94%	Time: 7.91s	LR: 0.000432
Epoch 35	Train Loss: 0.7394	Train Acc: 86.17%	Val Loss: 0.7509	Val Acc: 84.77%	Time: 7.97s	LR: 0.000388
Epoch 36	Train Loss: 0.7295	Train Acc: 86.48%	Val Loss: 0.7352	Val Acc: 85.62%	Time: 8.62s	LR: 0.000345
Epoch 37	Train Loss: 0.7329	Train Acc: 87.03%	Val Loss: 0.7218	Val Acc: 86.88%	Time: 8.21s	LR: 0.000303
Epoch 38	Train Loss: 0.7322	Train Acc: 87.07%	Val Loss: 0.7195	Val Acc: 86.95%	Time: 8.14s	LR: 0.000263
Epoch 39	Train Loss: 0.7222	Train Acc: 87.34%	Val Loss: 0.6987	Val Acc: 88.05%	Time: 8.09s	LR: 0.000224
Epoch 40	Train Loss: 0.7221	Train Acc: 87.25%	Val Loss: 0.6990	Val Acc: 88.12%	Time: 8.36s	LR: 0.000188
Epoch 41	Train Loss: 0.7258	Train Acc: 86.82%	Val Loss: 0.6993	Val Acc: 88.28%	Time: 7.91s	LR: 0.000154
Epoch 42	Train Loss: 0.7189	Train Acc: 87.11%	Val Loss: 0.7101	Val Acc: 86.56%	Time: 7.89s	LR: 0.000123
Epoch 43	Train Loss: 0.7255	Train Acc: 87.29%	Val Loss: 0.7031	Val Acc: 86.95%	Time: 7.98s	LR: 0.000095
Epoch 44	Train Loss: 0.7168	Train Acc: 87.66%	Val Loss: 0.6960	Val Acc: 87.81%	Time: 8.60s	LR: 0.000070
Epoch 45	Train Loss: 0.7127	Train Acc: 87.48%	Val Loss: 0.7002	Val Acc: 88.20%	Time: 7.99s	LR: 0.000049
Epoch 46	Train Loss: 0.7100	Train Acc: 87.58%	Val Loss: 0.7021	Val Acc: 88.28%	Time: 7.99s	LR: 0.000032
Epoch 47	Train Loss: 0.7089	Train Acc: 88.18%	Val Loss: 0.6969	Val Acc: 87.97%	Time: 7.98s	LR: 0.000018
Epoch 48	Train Loss: 0.7139	Train Acc: 88.24%	Val Loss: 0.6970	Val Acc: 89.38%	Time: 8.37s	LR: 0.000008
Epoch 49	Train Loss: 0.7077	Train Acc: 88.40%	Val Loss: 0.6964	Val Acc: 88.59%	Time: 7.94s	LR: 0.000002
Epoch 50	Train Loss: 0.7069	Train Acc: 87.91%	Val Loss: 0.6952	Val Acc: 88.28%	Time: 7.84s	LR: 0.000000
✔ Model saved as 22119129_22119067.pt						

Hình 3: Kết quả khi chạy mô hình huấn luyện sau 50 lần epoch.

6 Phân tích và đánh giá kết quả

6.1 Phân tích chi tiết về tham số

Từ bảng tính toán các tham số, ta thấy rằng:

- Các lớp Conv2D chiếm 94% (264096 tham số), tăng dần từ 864 đến 129600 khi số kênh tăng (3 -> 32 -> 64 -> 120), tập trung đặc trưng phức tạp.
- Các lớp BatchNorm chỉ có tổng cộng là 1104 tham số (chiếm tỷ lệ 0.4%) giúp ổn định trong việc huấn luyện mà không làm tăng độ phức tạp cho mô hình.
- Các lớp Pooling, Dropout, Flatten: các lớp như MaxPool2d, AdaptiveAvgPool2d,... thường sẽ không có tham số học được (0 tham số). Bởi chúng chỉ thực hiện các phép biến đổi không gian hay điều chỉnh cấu trúc dữ liệu. Cho nên các lớp này chủ yếu không đóng góp vào tổng số các tham số trong mô hình huấn luyện. Tuy nhiên, việc sử dụng các lớp này sẽ chống overfitting (Dropout) mà không làm tăng độ phức tạp của mô hình.
- Các lớp Fully Connected (Linear) chiếm 15488 tham số (tức khoảng 5.5% tổng số tham

số).

Nhận xét: Điều này cho thấy mô hình tập trung phần lớn tham số vào các lớp tích chập (Conv) để trích xuất đặc trưng, trong khi giai đoạn phân loại cuối thì được tối ưu nhằm cho mục đích tối ưu, giảm độ phức tạp.

6.2 Phân tích chi tiết về kết quả huấn luyện

Dựa theo kết quả trên, ta có thể đưa ra một số nhận xét như sau:

Train Accuracy:

- Train Accuracy thể hiện mức độ chính xác của mô hình trên tập dữ liệu.
- Qua 8 epoch đầu tiên, train acc tăng mạnh từ 27.75% đến 69%.
- Sau 15 epoch thì lúc này, train acc đã đến hơn 80%.
- Từ epoch 16 trở đi, train acc ngày càng tăng dần và đạt đỉnh điểm ở giá trị từ 87% đến 88%. Điều này cho thấy, mô hình có dấu hiệu học tốt từ tập dữ liệu.

Validation Accuracy

- Validation Accuracy phản ánh khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy.
- Bắt đầu từ epoch 15, lúc này giá trị Val acc đã đạt tới 80.62%.
- Từ epoch 16 đến 50, lúc này Val acc dần tăng mạnh và duy trì ổn định ở mức 88% đến khoảng 89%.
- Tuy nhiên, về độ chênh lệch giữa các epoch gần cuối thường nằm trong khoảng (0.3 đến 0.6). Đây là con số nhỏ, cho thấy mô hình đang hội tụ tốt.

Loss

- Từ bảng kết quả, ta thấy rằng cả train loss và val loss đều có xu hướng giảm ổn định, không có dấu hiệu tăng đột biến.
- > Điều này cho thấy rằng mô hình hội tụ tốt.
- Giá trị của val loss và train loss sau 50 lần epoch đã giảm đều, chứng tỏ mô hình tránh được tình trạng overfitting không may xảy ra.

6.3 Đánh giá tổng quan

Kiến trúc mạng:

- Cấu trúc hợp lý và hiện đại (3 khối convolution (feature extraction) và một khối fully connected(classifier)) cùng với classifier tuy đơn giản nhưng lại đem đến hiệu quả cao cho mô hình.
- Mô hình lựa chọn Batch Normalization, Dropout và AdaptiveAvgPool đúng chuẩn giúp cho việc hội tụ nhanh hơn, tăng tính ổn định.
- Kiến trúc trên cân bằng tốt giữa độ sâu cũng như khả năng tổng quát hóa.

Số lượng tham số:

- Số lượng tham số đạt tới khoảng 280k/300k, gần như tối đa. Điều này giúp mô hình có khả năng học các đặc trưng cần thiết cho bài toán phân loại hình ảnh 8 lớp.
- Đồng thời tránh được hiện tượng overfitting nhờ các kỹ thuật Dropout và BatchNorm.

Tính hội tụ của mô hình:

- Sau 50 epochs thì mô hình đạt hội tụ tốt và không có dấu hiệu bị overfitting nghiêm trọng.
- Với chênh lệch giữa Train Acc và Val Acc không nhiều (0.5 -> 1%), đây là điều vô cùng lý tưởng.

Độ chính xác:

- Với độ chính xác đạt tới 89.38% (cao nhất ở lần epoch 48) đã cho thấy kết quả vô cùng tốt cho mô hình phân loại hình ảnh. Bên cạnh đó, kết quả trên cũng vượt mức >80% yêu cầu đề bài đưa ra.
- Với độ chính xác cao như thế này đã cho thấy mô hình đã học rất tốt và đạt ổn định cao.