BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

----📖-----

BÀI TẬP LỚN

Môn học: Lập Trình Python

Giảng viên hướng dẫn: **Kim Ngọc Bách**

Sinh viên thực hiện:

Nguyễn Văn Ninh: B23DCAT232

Dương Anh Đức: B23DCCE018

Lớp: **D23CQCE06-B**

Niên khóa: **2024 - 2025**

Hệ đào tạo: **Đại học chính quy**

Hà Nội, Tháng 06/2025

Mục Lục

1. Giới thiệu

Đoạn mã này triển khai một quy trình hoàn chỉnh để huấn luyện và đánh giá hai loại mô hình mạng nơ-ron – Mạng Perceptron Đa lớp (MLP) và Mạng Nơ-ron Tích chập (CNN) – cho bài toán phân loại hình ảnh trên bộ dữ liệu CIFAR-10. Mục tiêu là so sánh hiệu suất của hai kiến trúc này trên cùng một tác vụ và trực quan hóa kết quả.

2. Cấu trúc tổng quan của mã nguồn

Mã nguồn được tổ chức thành các phần chính sau:

- Nhập thư viện: Các thư viện cần thiết cho PyTorch, xử lý dữ liệu, tính toán và trực quan hóa.

- Thiết lập toàn cục: Đặt seed cho tính tái lập và chọn thiết bị (CPU/GPU).

- Định nghĩa mô hình: Hai lớp MLPNet và CNNNet kế thừa từ nn.Module của PyTorch.

- Lớp quản lý quy trình: Lớp CIFAR10Classifier đóng gói toàn bộ logic từ tải dữ liệu, xây dựng, huấn luyện, đánh giá mô hình đến trực quan hóa.

- Hàm main: Điểm khởi đầu của chương trình, điều phối các bước thực thi.

3. Các thư viện sử dụng

- torch, torch.nn, torch.optim, torch.nn.functional: Nền tảng PyTorch cho việc xây dựng và huấn luyện mạng nơ-ron.

- torchvision, torchvision.transforms: Cung cấp bộ dữ liệu CIFAR-10 và các phép biến đổi hình ảnh (augmentation, normalization).

- torch.utils.data.DataLoader, torch.utils.data.random_split: Hỗ trợ tải dữ liệu theo batch và chia tập dữ liệu.

- matplotlib.pyplot, seaborn: Dùng để vẽ đồ thị (đường cong học tập, ma trận nhầm lẫn).

- numpy: Thư viện tính toán số học, thường dùng để chuyển đổi tensor sang array cho các thư viện khác.

- sklearn.metrics: Cung cấp các hàm tính toán ma trận nhầm lẫn và báo cáo phân loại.

- warnings: Để bỏ qua các cảnh báo không cần thiết.

4. Thiết lập ban đầu và cấu hình

- torch.manual_seed(42) và np.random.seed(42): Đảm bảo kết quả có thể tái lập được qua các lần chạy.

- device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'): Tự động chọn GPU nếu có, nếu không thì dùng CPU. Điều này quan trọng để tăng tốc độ huấn luyện.

- warnings.filterwarnings('ignore'): Ẩn các cảnh báo có thể xuất hiện trong quá trình chạy.

5. Định nghĩa mô hình

5.1. Mạng Perceptron Đa lớp (MLPNet)

- Kiến trúc:

+) Lớp Flatten để duỗi thẳng ảnh đầu vào (32x32x3) thành vector 1D.

+) Ba lớp ẩn fully connected (FC) với kích thước lần lượt là 1024, 512, 256.

+) Mỗi lớp FC được theo sau bởi BatchNorm1d (chuẩn hóa batch), hàm kích hoạt ReLU và lớp Dropout (với tỷ lệ 0.4) để chống overfitting.

+) Lớp output FC cuối cùng có num_classes (mặc định là 10) nơ-ron, không có hàm kích hoạt (do CrossEntropyLoss sẽ tự động áp dụng softmax).

- Input size: Mặc định là 32*32*3 = 3072 (kích thước ảnh CIFAR-10).

- Output size: num_classes (số lớp của CIFAR-10).

5.2. Mạng Nơ-ron Tích chập (CNNNet)

- Kiến trúc:

+) Khối tích chập 1: Conv2d (3 kênh đầu vào, 32 kênh đầu ra, kernel 3x3, padding 1) -> BatchNorm2d -> ReLU -> MaxPool2d (kernel 2x2).

+) Khối tích chập 2: Conv2d (32 kênh đầu vào, 64 kênh đầu ra, kernel 3x3, padding 1) -> BatchNorm2d -> ReLU -> MaxPool2d (kernel 2x2).

+) Khối tích chập 3: Conv2d (64 kênh đầu vào, 128 kênh đầu ra, kernel 3x3, padding 1) -> BatchNorm2d -> ReLU -> MaxPool2d (kernel 2x2).

+) Sau 3 lớp pooling, kích thước ảnh giảm từ 32x32 xuống 4x4.

+) Flatten để duỗi thẳng feature map (128x4x4).

+) Lớp FC ẩn với 256 nơ-ron, theo sau bởi ReLU và Dropout (tỷ lệ 0.5).

+) Lớp output FC với num_classes nơ-ron.

- Input: Ảnh màu 3 kênh.

- Output: num_classes (số lớp của CIFAR-10).

6. Lớp quản lý CIFAR10Classifier

Đây là lớp trung tâm điều phối toàn bộ quy trình.

6.1. Khởi tạo (__init__)

- Lưu trữ tên các lớp của CIFAR-10.

- Khởi tạo các biến thành viên: device, train_loader, val_loader, test_loader, các mô hình (mlp_model, cnn_model), và các dictionary để lưu trữ lịch sử huấn luyện (mlp_history, cnn_history).

6.2. Tải và tiền xử lý dữ liệu (load_and_preprocess_data)

- Tải dữ liệu: Sử dụng torchvision.datasets.CIFAR10 để tải tập huấn luyện và kiểm tra.

- Tiền xử lý và Augmentation:

+) transform_train: Áp dụng cho tập huấn luyện, bao gồm:

* RandomHorizontalFlip: Lật ảnh ngẫu nhiên theo chiều ngang.

* RandomRotation(10): Xoay ảnh ngẫu nhiên một góc tối đa 10 độ.

* ToTensor(): Chuyển ảnh PIL/NumPy thành Tensor.

* Normalize: Chuẩn hóa giá trị pixel sử dụng trung bình và độ lệch chuẩn của CIFAR-10.

+) transform_test: Áp dụng cho tập validation và test, chỉ bao gồm ToTensor() và Normalize.

- Phân chia dữ liệu: Chia tập huấn luyện ban đầu thành tập huấn luyện mới (90%) và tập validation (10%) bằng random_split.

+)Lưu ý quan trọng: Đảm bảo tập validation sử dụng transform_test (không có augmentation) để đánh giá khách quan.

- Tạo DataLoaders: DataLoader cho các tập train, validation, và test để tải dữ liệu theo batch, xáo trộn (shuffle) cho tập train.

6.3. Xây dựng mô hình (build_models)

- Khởi tạo các đối tượng MLPNet và CNNNet.

- Di chuyển các mô hình lên self.device (GPU hoặc CPU).

- In ra số lượng tham số của mỗi mô hình.

6.4. Huấn luyện mô hình (train_model và train_models)

- train_model(model, model_name, epochs, lr): Hàm chung để huấn luyện một mô hình.

+) Loss function: nn.CrossEntropyLoss().

+) Optimizer: optim.Adam với learning rate lr (mặc định 0.001) và weight_decay (L2 regularization) là 1e-4.

+) Learning Rate Scheduler: optim.lr_scheduler.StepLR giảm learning rate đi gamma=0.5 sau mỗi step_size=7 epochs.

+) Early Stopping:

* Theo dõi val_acc. Nếu val_acc không cải thiện trong patience=5 epochs liên tiếp, quá trình huấn luyện sẽ dừng sớm.

+) Vòng lặp huấn luyện:

Qua từng epoch:

Pha huấn luyện:

1. Đặt mô hình ở chế độ model.train().

2. Lặp qua các batch trong train_loader.

3. Tính toán loss, thực hiện backpropagation và cập nhật trọng số.

4. Tính toán accuracy và loss trên tập huấn luyện.

Pha validation:

1. Đặt mô hình ở chế độ model.eval().

2. Tắt tính toán gradient (torch.no_grad()).

3. Lặp qua các batch trong val_loader.

4. Tính toán accuracy và loss trên tập validation.

5. Lưu trữ các metrics (train loss/acc, val loss/acc) vào history.

6. In thông tin huấn luyện của epoch.

7. Cập nhật learning rate scheduler.

8. train_models(epochs): Gọi hàm train_model lần lượt cho MLP và CNN.

6.5. Đánh giá mô hình (evaluate_model)

- Đặt mô hình ở chế độ model.eval().

- Tắt tính toán gradient.

- Lặp qua các batch trong test_loader.

- Tính toán loss và accuracy trên tập test.

- Thu thập tất cả các dự đoán (all_predictions) và nhãn thật (all_targets) để sử dụng cho ma trận nhầm lẫn và báo cáo phân loại.

- In kết quả đánh giá.

6.6. Trực quan hóa kết quả

- plot_learning_curves():

+)Sử dụng matplotlib để vẽ 4 đồ thị:

* MLP - Training & Validation Loss vs. Epochs

* MLP - Training & Validation Accuracy vs. Epochs

* CNN - Training & Validation Loss vs. Epochs

* CNN - Training & Validation Accuracy vs. Epochs

- plot_confusion_matrices(mlp_pred, mlp_targets, cnn_pred, cnn_targets):

+) Tính toán ma trận nhầm lẫn cho MLP và CNN
bằng sklearn.metrics.confusion_matrix.

+) Sử dụng seaborn.heatmap để vẽ ma trận nhầm lẫn.

+) In ra báo cáo phân loại chi tiết (precision, recall, F1-score) cho từng lớp
bằng sklearn.metrics.classification_report.

- display_sample_predictions(num_samples):

+) Lấy ngẫu nhiên num_samples ảnh từ test_loader.

+) Thực hiện dự đoán bằng cả hai mô hình MLP và CNN.

+) Hiển thị các ảnh mẫu, nhãn thật và nhãn dự đoán của cả hai mô hình.

+) Quan trọng: Thực hiện un-normalize ảnh trước khi hiển thị để màu sắc được chính xác.

7. Hàm thực thi chính (main)

- Tạo một đối tượng CIFAR10Classifier.

- Thực hiện tuần tự các bước:

1. load_and_preprocess_data(): Tải và chuẩn bị dữ liệu.

2. build_models(): Xây dựng kiến trúc MLP và CNN.

3. train_models(epochs=15): Huấn luyện cả hai mô hình trong 15 epochs.

4. evaluate_model(): Đánh giá cả hai mô hình trên tập test.

5. plot_learning_curves(): Vẽ đường cong học tập.

6. plot_confusion_matrices(): Vẽ ma trận nhầm lẫn và in báo cáo.

7. display_sample_predictions(): Hiển thị một số dự đoán mẫu.

8. Phân tích và nhận xét

- Cấu trúc tốt: Mã được tổ chức tốt thành các lớp và phương thức rõ ràng, giúp dễ đọc, dễ hiểu và bảo trì. Lớp CIFAR10Classifier đóng vai trò trung tâm, quản lý hiệu quả toàn bộ quy trình.

- Kỹ thuật hiện đại:

+) Sử dụng **Batch Normalization** để ổn định và tăng tốc quá trình huấn luyện.

+) Sử dụng **Dropout** để giảm overfitting.

+) Áp dụng **Data Augmentation** (lật, xoay) để tăng tính đa dạng của dữ liệu huấn luyện, giúp mô hình khái quát hóa tốt hơn.

+) Sử dụng **Adam Optimizer** và **Learning Rate Scheduler** để tối ưu hóa hiệu quả.

+) Triển khai **Early Stopping** để tránh lãng phí tài nguyên và overfitting khi mô hình không còn cải thiện.

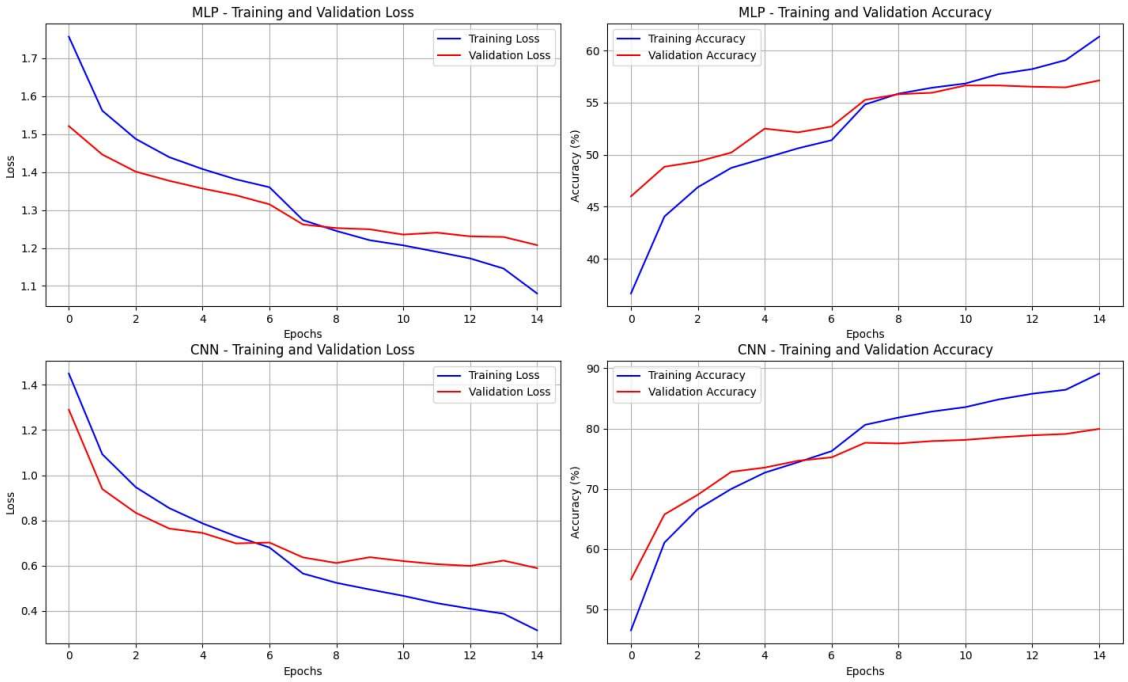
- So sánh mô hình: Mã nguồn được thiết kế để so sánh trực tiếp hiệu suất của MLP và CNN trên cùng một tác vụ và bộ dữ liệu. CNN thường được kỳ vọng sẽ cho kết quả tốt hơn MLP trên các tác vụ thị giác máy tính do khả năng nắm bắt các đặc trưng không gian của ảnh.

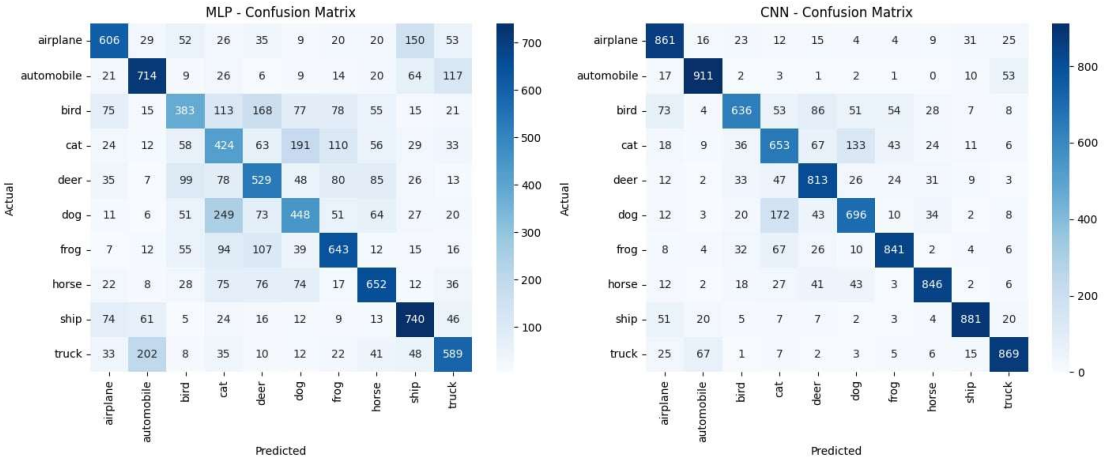
- Trực quan hóa toàn diện: Cung cấp nhiều hình thức trực quan hóa (đường cong học tập, ma trận nhầm lẫn, dự đoán mẫu) giúp người dùng dễ dàng phân tích và hiểu kết quả của mô hình.

- Tính tái lập: Việc đặt seed đảm bảo kết quả nhất quán giữa các lần chạy.

- Hiệu suất: Tự động sử dụng GPU (nếu có) giúp tăng tốc đáng kể thời gian huấn luyện.

9. Hình Ảnh Minh Họa





Sample Predictions Comparison