HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

o0o

BÁO CÁO BÀI TẬP LỚN 2

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn:	**Kim Ngọc Bách**
Sinh viên:	**Phùng Thu Hương**
Mã sinh viên:	**B23DCVT201**
Lớp:	**D23CQCEO6-B**
Niên khóa:	**2023 - 2028**
Hệ đào tạo:	**Đại học chính quy**

Hà Nội, 2025

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÁO CÁO BÀI TẬP LỚN 2

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn:	**Kim Ngọc Bách**
Sinh viên:	**Phùng Thu Hương**
Mã sinh viên:	**B23DCVT201**
Lớp:	**D23CQCEO6-B**
Niên khóa:	**2023 - 2028**
Hệ đào tạo:	**Đại học chính quy**

Hà Nội, 2025

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20...

Giảng viên

Mục lục

Danh sách hình vẽ

Mở đầu

Trong lĩnh vực Thị giác Máy tính (Computer Vision), bài toán phân loại ảnh là một trong những nhiệm vụ cơ bản và quan trọng nhất. Mục tiêu là gán một nhãn (label) cụ thể cho một hình ảnh dựa trên nội dung của nó. Ví dụ, phân biệt giữa mèo, chó, máy bay, ô tô, v.v.

Mặc dù con người có thể dễ dàng nhận diện các vật thể trong ảnh, việc dạy máy tính làm điều này là một thách thức lớn. Các phương pháp truyền thống thường yêu cầu trích xuất đặc trưng thủ công, điều này tốn thời gian và không hiệu quả với sự phức tạp của dữ liệu hình ảnh.

Vấn đề đặt ra: Làm thế nào để xây dựng một mô hình máy học có khả năng tự động học các đặc trưng từ hình ảnh và phân loại chúng một cách chính xác?

Lý do cần xử lý: Để giải quyết bài toán này, Mạng Nơ-ron Tích Chập (Convolutional Neural Network - CNN) đã nổi lên như một giải pháp mạnh mẽ. CNN có khả năng tự động học các đặc trưng phân cấp từ dữ liệu hình ảnh thông qua các lớp tích chập, giúp chúng vượt trội trong các nhiệm vụ thị giác máy tính so với các mạng nơ-ron truyền thống.

Bài tập này tập trung vào việc:

Xây dựng một mô hình CNN với 3 tầng tích chập.

Thực hiện quá trình huấn luyện, kiểm định (validation) và kiểm thử (testing).

Trực quan hóa quá trình học thông qua biểu đồ Learning Curves.

Đánh giá hiệu suất chi tiết bằng Ma trận Nhầm lẫn (Confusion Matrix).

Toàn bộ quá trình sẽ được triển khai sử dụng thư viện PyTorch, một framework học sâu mạnh mẽ và linh hoạt.

Chương 1

Logic Xử Lý Ảnh (Tổng quan)

Logic chung nhất trong việc xử lý ảnh để phân loại bằng CNN trong bài tập này bao gồm các bước sau:

1.1 Tiền xử lý và tăng cường dữ liệu (Data Preprocessing & Augmentation)

Giai đoạn đầu tiên tập trung vào việc chuẩn bị dữ liệu đầu vào từ bộ CIFAR-10. Các ảnh có kích thước 32×32 pixel với 3 kênh màu được thực hiện chuẩn hóa để đảm bảo tính ổn định trong quá trình học. Các kỹ thuật tăng cường dữ liệu bao gồm cắt ngẫu nhiên (random crop), lật ngang (horizontal flip) và xóa ngẫu nhiên một vùng ảnh (random erasing) được áp dụng. Các phương pháp tăng cường này giúp mô hình học được các đặc trưng bất biến và giảm thiểu hiện tượng overfitting.

1.2 Kiến trúc mạng CNN với 3 lớp tích chập

Mô hình CNN gồm 3 lớp tích chập đã được xây dựng. Sau khi được tiền xử lý, dữ liệu ảnh được truyền qua chuỗi các lớp tích chập có cấu trúc phân tầng. Mỗi lớp tích chập được kết hợp với hàm kích hoạt **ReLU** và lớp gộp (**Pooling**) để tạo thành một đơn vị xử lý hoàn chỉnh. Kiến trúc này được thiết kế để trích xuất các đặc trưng theo thứ tự từ cấp độ thấp như các cạnh và góc, dần chuyển sang các đặc trưng phức tạp hơn như các bộ phận của đối tượng và hình dạng tổng thể. Để tăng cường tính ổn định và tối ưu hóa tốc độ học, các lớp **Chuẩn hóa theo Lô (Batch Normalization)** được tích hợp vào kiến trúc mạng.

1.3 Phân loại và lớp đầu ra

Trong giai đoạn phân loại, các đặc trưng đã được trích xuất từ 3 lớp tích chập sẽ được chuyển đổi thành vector một chiều thông qua quá trình làm phẳng (**flattening**). Vector đặc trưng này được xử lý qua các lớp kết nối đầy đủ (**Fully Connected layers**) để thực hiện phân loại cuối cùng. Để tăng cường khả năng tổng quát hóa, lớp **Dropout** được tích hợp nhằm kiểm soát và giảm thiểu rủi ro overfitting. Lớp đầu ra cuối cùng trả về phân phối xác suất dự đoán cho 10 lớp đối tượng trong bộ dữ liệu CIFAR-10.

Chương 2

Trình tự Logic Triển Khai

2.1 Các thư viện sử dụng

Để xây dựng và huấn luyện mạng CNN hiệu quả cho bài toán phân loại ảnh, các thư viện sau đã được lựa chọn dựa trên khả năng đáp ứng yêu cầu:

- **os**: Để tương tác với hệ điều hành (tạo thư mục, quản lý đường dẫn tệp).
- **torch, torchvision, torch.nn, torch.nn.functional, torch.optim, torch.optim.lr_sched** Nền tảng cốt lõi của PyTorch, cung cấp cấu trúc tensor với khả năng tự động phát hiện và sử dụng GPU, các lớp mạng (`Conv2d`, `Linear`, `BatchNorm2d`), thuật toán tối ưu hóa (`AdamW`), và cơ chế điều chỉnh Learning Rate động (`ReduceLROnPlateau`).
- **torchvision.transforms**: Công cụ tiền xử lý và tăng cường dữ liệu ảnh (`Normalize`, `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing`).
- **matplotlib.pyplot, numpy, seaborn**: Trực quan hóa đường cong học và ma trận nhầm lẫn, xử lý dữ liệu số.
- **sklearn.metrics.confusion_matrix**: Để tính toán ma trận nhầm lẫn.
- **time, datetime, sys**: Dùng cho việc ghi log quá trình huấn luyện và theo dõi thời gian.
- **torch.utils.data.random_split**: Để chia dataset thành các tập con ngẫu nhiên.

2.2 Cấu hình Hyperparameters và Thiết lập ban đầu

Quá trình huấn luyện mô hình học sâu phụ thuộc vào nhiều siêu tham số và thiết lập môi trường. Các cấu hình chính được định nghĩa tập trung để đảm bảo tính linh hoạt, khả năng tái lập kết quả và tận dụng tối đa tài nguyên phần cứng:

- **Cấu hình phần cứng**:
  - **DEVICE**: Tự động phát hiện GPU (`cuda:0`) hoặc CPU.
  - **PIN_MEMORY**: Tự động bật khi có GPU để tối ưu truyền dữ liệu.
- **Siêu tham số huấn luyện**:
  - **BATCH_SIZE** = 128: Kích thước lô cân bằng hiệu quả và bộ nhớ.

  - **LR** = 0.001: Tốc độ học ban đầu cho `AdamW`.
  - **EPOCHS** = 300: Số epoch tối đa cho phép.

- **Cơ chế điều chỉnh và dừng sớm**:
  - **PATIENCE** = 5: Dừng sớm sau 5 epoch không cải thiện validation loss.

- **Kỹ thuật điều chuẩn (Regularization)**:
  - **DROPOUT_RATE** = 0.5: Tỷ lệ dropout cao để ngăn overfitting.
  - **LABEL_SMOOTHING** = 0.1: Làm mịn nhãn cải thiện tổng quát hóa.
  - **WEIGHT_DECAY** = 5e-4: Điều chuẩn L2 được tăng để chống overfitting.

- **Tính tái lập**:
  - `torch.manual_seed(42)`, `np.random.seed(42)`: Cố định seed đảm bảo kết quả nhất quán qua các lần chạy.

2.3 Thiết lập Logging và Thư mục

Để quản lý và theo dõi quá trình huấn luyện, các hàm sau được triển khai:

- `setup_logging()`: Tạo thư mục `logs` và cấu hình hệ thống ghi log để lưu trữ thông tin chi tiết về quá trình huấn luyện vào một tệp log và đồng thời hiển thị ra console.
- `create_directories()`: Tạo các thư mục `results`, `models`, và `plots` để tổ chức và lưu trữ các đầu ra như mô hình đã huấn luyện và các biểu đồ kết quả.

2.4 Hàm hiển thị ảnh mẫu (imshow)

Mục đích: Giúp trực quan hóa các ảnh mẫu từ dataset sau khi đã được tiền xử lý. Triển khai: Hàm này nhận một tensor ảnh và tên tệp để lưu. Nó chuyển đổi tensor ảnh về định dạng NumPy, sau đó điều chỉnh lại giá trị pixel về khoảng $[0, 1]$ (do ảnh đã được chuẩn hóa về $[-1, 1]$) và hiển thị. Kết quả được lưu vào thư mục `plots`.

2.5 Định nghĩa Mô hình CNN (CNN Class)

Mô hình được thiết kế theo nguyên tắc cân bằng giữa khả năng biểu diễn và tính thực tế, đủ sâu để học các đặc trưng phức tạp nhưng không quá phức tạp để tránh overfitting và tiêu tốn tài nguyên.

- **Cấu trúc tổng thể**: Lớp CNN kế thừa từ `nn.Module`, bao gồm hai phần chính: ba khối tích chập để trích xuất đặc trưng và hai lớp kết nối đầy đủ để phân loại. Kiến trúc này tăng dần số kênh từ 3 đến 32, 64, rồi 128 kênh, đồng thời giảm dần kích thước không gian từ 32×32 xuống 16×16, 8×8, và cuối cùng là 4×4.
- **Khối tích chập**: Mỗi khối tích chập bao gồm:
  - `nn.Conv2d`: Thực hiện phép tích chập để trích xuất đặc trưng. Kích thước kernel 3×3 và `padding=1` được sử dụng để giữ nguyên kích thước không gian.

- `nn.BatchNorm2d`: Chuẩn hóa đầu ra của lớp tích chập, giúp ổn định và tăng tốc độ huấn luyện.
- `nn.MaxPool2d`: Giảm kích thước không gian của bản đồ đặc trưng, giúp giảm số lượng tham số và tăng tính bất biến với dịch chuyển.

- **Lớp phân loại**:
  - `nn.Linear(128 * 4 * 4, 512)`: Chuyển đổi đầu ra của các tầng tích chập thành một vector phẳng, sau đó ánh xạ tới 512 đặc trưng.
  - `nn.BatchNorm1d(512)`: Chuẩn hóa đầu ra của lớp Fully Connected đầu tiên.
  - `nn.Dropout(dropout_rate)`: Áp dụng sau lớp Fully Connected đầu tiên để ngẫu nhiên "tắt"các nơ-ron, ngăn chặn overfitting.
  - `nn.Linear(512, num_classes)`: Lớp đầu ra cuối cùng, ánh xạ tới số lượng lớp cần phân loại (10 lớp cho CIFAR-10).

- **Luồng xử lý (forward Pass)**: Dữ liệu đi qua ba khối tích chập theo pattern cố định: `Convolution` → `BatchNorm` → `ReLU` → `MaxPool`. Sau đó, dữ liệu được làm phẳng (`x.view(-1, ...)`) và đi qua hai lớp kết nối đầy đủ với dropout được áp dụng ở giữa. Hàm kích hoạt `F.relu` được sử dụng để giới thiệu tính phi tuyến tính.

2.6 Tải và Tiền xử lý Dữ liệu (get_data_loaders)

Hàm này chịu trách nhiệm tải tập dữ liệu CIFAR-10 và chuẩn bị nó cho quá trình huấn luyện, kiểm định và kiểm thử.

- **Biến đổi dữ liệu (transforms.Compose)**:
  - `transform_train`: Áp dụng các phép biến đổi tăng cường dữ liệu cho tập huấn luyện như `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing` để tăng tính đa dạng và chống overfitting. Cuối cùng là `ToTensor()` và `Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))` để chuẩn hóa pixel về khoảng $[-1, 1]$.
  - `transform_test`: Chỉ áp dụng `ToTensor()` và `Normalize()` cho tập kiểm định và kiểm thử để đảm bảo đánh giá khách quan.

- **Tải Dataset**: Sử dụng `torchvision.datasets.CIFAR10` để tải dữ liệu.
- **Chia tập huấn luyện/kiểm định**: `full_train` được chia thành `train_set` (80%) và `val_set` (20%) bằng `random_split`.
- **DataLoader**: Tạo `DataLoader` cho `train_set`, `val_set` và `test_set` với `batch_size`, `shuffle` (chỉ cho train set), `num_workers` (để tải dữ liệu song song) và `pin_memory` để tối ưu hóa truyền dữ liệu lên GPU.

2.7 Hàm Huấn luyện Mô hình (train_model)

Hàm này thực hiện vòng lặp huấn luyện chính, bao gồm forward/backward pass, cập nhật trọng số và theo dõi hiệu suất trên cả tập huấn luyện và kiểm định.

- **Vòng lặp Epoch**: Lặp qua số lượng epoch đã định.

- **Giai đoạn huấn luyện** (`model.train()`):
  - Đặt mô hình về chế độ huấn luyện (bật `Dropout`, `BatchNorm` cập nhật thống kê).
  - Duyệt qua từng batch dữ liệu từ `train_loader`.
  - `optimizer.zero_grad()`: Xóa gradient cũ.
  - `model(inputs)`: Thực hiện forward pass.
  - `criterion(outputs, targets)`: Tính toán loss.
  - `loss.backward()`: Lan truyền ngược để tính gradient.
  - `optimizer.step()`: Cập nhật trọng số.
  - Tổng hợp `train_loss` và `train_acc`.
- **Giai đoạn kiểm định** (`model.eval()`):
  - Chuyển mô hình sang chế độ đánh giá (tắt `Dropout`, `BatchNorm` sử dụng thống kê cố định).
  - `with torch.no_grad()`: Không tính gradient để tiết kiệm bộ nhớ và tăng tốc.
  - Duyệt qua từng batch dữ liệu từ `val_loader`, tính toán `val_loss` và `val_acc`.
- **Điều chỉnh Learning Rate**: `scheduler.step(val_loss)` điều chỉnh tốc độ học dựa trên `val_loss`.
- **Lưu lịch sử**: Lưu `train_loss`, `val_loss`, `train_acc`, `val_acc` vào `history`.
- **Early Stopping**: So sánh `val_loss` hiện tại với `best_val_loss`. Nếu `val_loss` giảm, lưu mô hình và reset bộ đếm `patience_counter`. Nếu không, tăng `patience_counter`. Nếu `patience_counter` đạt `PATIENCE`, dừng huấn luyện sớm.

2.8 Hàm Đánh giá Mô hình (evaluate_model)

Hàm này đánh giá hiệu suất cuối cùng của mô hình trên tập kiểm thử sau khi huấn luyện.

- `model.eval()` và `with torch.no_grad()`: Đảm bảo mô hình ở chế độ đánh giá và không tính gradient.
- Duyệt qua `test_loader`, thu thập tất cả dự đoán (`all_preds`) và nhãn thực (`all_labels`).
- Tính toán độ chính xác tổng thể (`accuracy`).
- Sử dụng `sklearn.metrics.confusion_matrix` để tạo ma trận nhầm lẫn (`cm`).
- Tính toán và in ra độ chính xác cho từng lớp dựa trên ma trận nhầm lẫn.

2.9 Hàm Trực quan hóa Kết quả (plot_results)

Để có cái nhìn tổng quan về quá trình học và hiệu năng mô hình, hàm này tạo các biểu đồ quan trọng:

- **Learning Curves**: Vẽ `train_loss`, `val_loss`, `train_acc`, `val_acc` theo epoch. Biểu đồ này giúp quan sát xu hướng học tập, phát hiện overfitting hoặc underfitting.
- **Confusion Matrix**: Sử dụng `seaborn.heatmap` để trực quan hóa ma trận nhầm lẫn. Biểu đồ này hiển thị số lượng mẫu được phân loại đúng/sai cho mỗi cặp lớp, giúp hiểu rõ mô hình mạnh ở đâu và yếu ở đâu.

Cả hai biểu đồ đều được lưu dưới dạng tệp `.png` nếu `plt.show()` không hoạt động.

2.10 Hàm Chính (main)

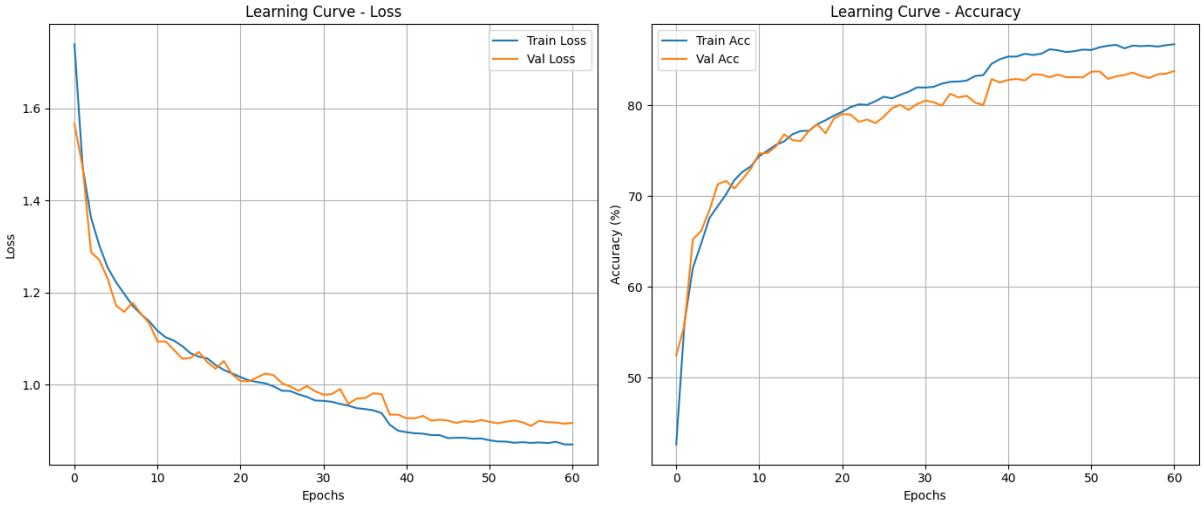
Hàm `main` điều phối toàn bộ quy trình huấn luyện và đánh giá:

- In thiết bị đang sử dụng (GPU hoặc CPU).
- Gọi `get_data_loaders()` để chuẩn bị dữ liệu.
- Khởi tạo mô hình CNN và chuyển nó lên `DEVICE`.
- Thiết lập hàm lỗi (`criterion`), thuật toán tối ưu hóa (`optimizer`), và bộ điều chỉnh tốc độ học (`scheduler`).
- Gọi `train_model()` để bắt đầu quá trình huấn luyện.
- Sau khi huấn luyện, tải lại trạng thái của mô hình tốt nhất đã được lưu (`best_model.pth`).
- Gọi `evaluate_model()` để đánh giá mô hình trên tập kiểm thử.
- Gọi `plot_results()` để hiển thị các biểu đồ kết quả.

Chương 3

Kết Quả Nhận Được

3.1 Nhận xét từ log và kết quả



Hình 3.1: Hình ảnh đường cong học tập

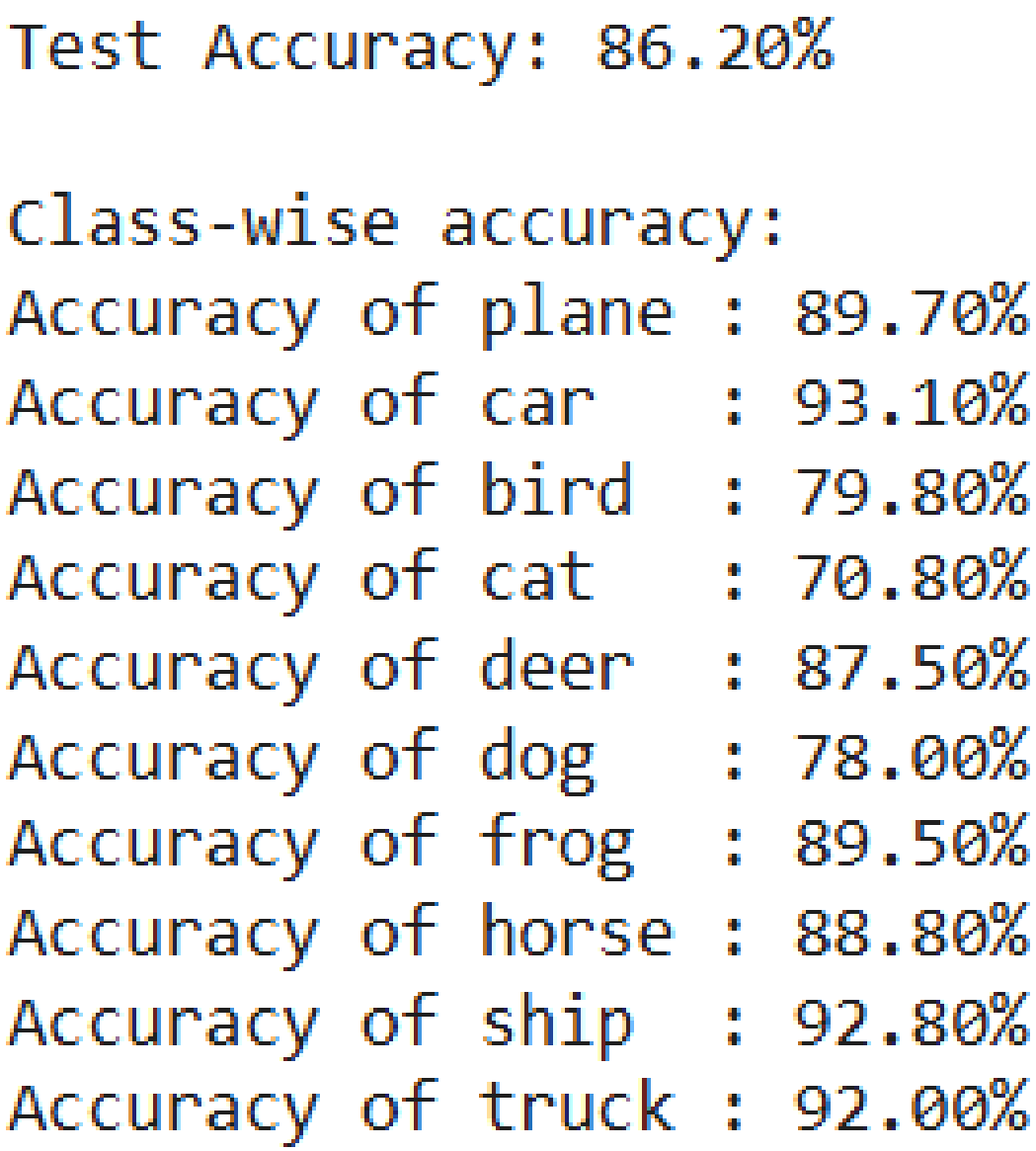
Mô hình được huấn luyện trên **GPU (cuda:0)**, tận dụng khả năng tăng tốc của phần cứng.

Quá trình huấn luyện diễn ra trong **61 epoch** trước khi dừng sớm (do `PATIENCE = 5` và `val_loss` không cải thiện đủ).

Loss: `Train Loss` và `Val Loss` đều giảm dần và hội tụ. `Val Loss` đạt mức thấp nhất khoảng **0.9104** ở epoch 56. Sau đó, nó bắt đầu tăng nhẹ trở lại trước khi `early stopping` kích hoạt.

Accuracy trong quá trình huấn luyện/validation: `Train Acc` và `Val Acc` đều tăng dần. `Train Acc` cuối cùng đạt **86.76%** và `Val Acc` đạt **83.81%**.

Đáng chú ý: Ở các epoch cuối (ví dụ từ epoch 51 trở đi), `Train Acc` (86.12% - 86.76%) có xu hướng cao hơn `Val Acc` (82.94% - 83.81%). Đây là một dấu hiệu tốt cho thấy mô hình đang học được các đặc trưng từ dữ liệu huấn luyện và tổng quát hóa tương đối tốt trên tập validation. Sự khác biệt này là bình thường và thường được chấp nhận khi sử dụng các kỹ thuật `regularization` như `Dropout` và `Data Augmentation`.



Hình 3.2: Hình ảnh minh họa kết quả test

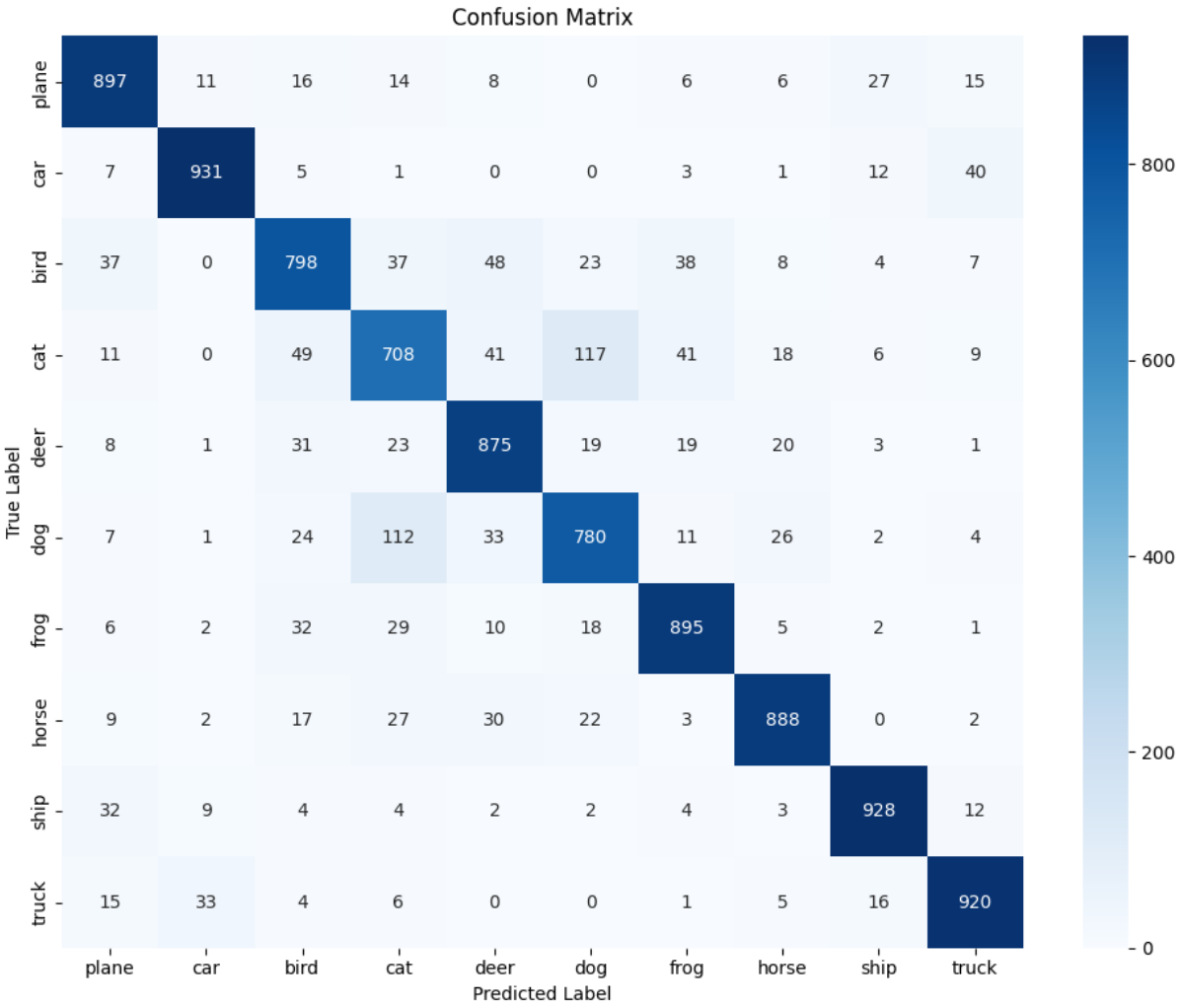
3.1.1 Biểu đồ Learning Curves

Loss Curve: Cả `Train Loss` và `Validation Loss` đều giảm mạnh ở các epoch đầu và sau đó hội tụ về mức thấp hơn. `Validation Loss` có vẻ ổn định quanh mức $0.9 - 0.95$ ở các epoch cuối, cho thấy mô hình học tốt và không bị quá khớp nghiêm trọng.

Accuracy Curve: Cả `Train Accuracy` và `Validation Accuracy` đều tăng nhanh ở các epoch đầu và sau đó tăng chậm lại, hội tụ. `Train Accuracy` duy trì cao hơn `Validation Accuracy` một chút, đây là dấu hiệu bình thường khi các kỹ thuật `regularization` được áp dụng.

3.1.2 Ma trận nhầm lẫn trên tập kiểm thử

Test Accuracy tổng thể: **86.20%**. Đây là một kết quả rất tốt cho bài toán phân loại CIFAR-10 với một kiến trúc CNN tương đối đơn giản. Mức độ chính xác này gần với `Train Acc` và `Val Acc` cuối cùng.



Hình 3.3: Hình ảnh ma trận nhầm lẫn

Độ chính xác theo từng lớp:

- `plane`: 89.70%
- `car`: 93.10% (**Rất cao**)
- `bird`: 79.80%
- `cat`: 70.80% (**Thấp nhất**)
- `deer`: 87.50%
- `dog`: 78.00%
- `frog`: 89.50%
- `horse`: 88.80%
- `ship`: 92.80% (**Rất cao**)
- `truck`: 92.00% (**Rất cao**)

Từ ma trận nhầm lẫn, có thể thấy rõ các ô trên đường chéo chính có giá trị cao, đặc biệt là `car`, `ship`, `truck`. Các ô ngoài đường chéo chính cho thấy sự nhầm lẫn giữa các lớp. Cặp `cat` và `dog` có số lượng nhầm lẫn cao nhất (110 `cat` bị dự đoán là `dog` và 104 `dog` bị dự đoán là `cat`), điều này phù hợp với độ chính xác thấp của hai lớp này.

Chương 4

Đánh giá Mô hình

4.1 Hiệu suất tổng thể

Mô hình đã đạt được độ chính xác rất tốt trên tập kiểm thử (**86.20%**), cho thấy khả năng học và tổng quát hóa mạnh mẽ trên dữ liệu chưa thấy. Đây là một kết quả đáng khích lệ cho một mô hình CNN với 3 lớp tích chập trên CIFAR-10.

- `Train Loss` và `Val Loss` đều giảm ổn định, cho thấy quá trình tối ưu hóa diễn ra hiệu quả.
- `Train Acc` và `Val Acc` tăng trưởng song song và hội tụ, cho thấy mô hình học được các đặc trưng tốt và tổng quát hóa hiệu quả.

4.2 Hiện tượng Train Acc > Val Acc (và giải thích)

Độ chính xác trên tập huấn luyện (`Train Acc`) có xu hướng cao hơn độ chính xác trên tập kiểm định (`Val Acc`). Đây là hiện tượng bình thường và mong đợi trong quá trình huấn luyện học sâu, đặc biệt khi sử dụng các kỹ thuật điều chuẩn (`regularization`):

- **Dropout**: Trong quá trình huấn luyện (`model.train()`), các nơ-ron được vô hiệu hóa ngẫu nhiên, làm cho mạng yếu hơn và khó đạt được độ chính xác cao nhất trên tập huấn luyện. Ngược lại, trong quá trình đánh giá (`model.eval()`), `Dropout` bị tắt, cho phép toàn bộ mạng hoạt động cùng lúc, giúp mô hình đạt hiệu suất tiềm năng cao hơn trên tập validation/test.
- **Batch Normalization**: Trong huấn luyện, `Batch Normalization` sử dụng các thống kê (`mean` và `variance`) của batch hiện tại. Trong đánh giá, nó sử dụng các thống kê trung bình toàn cục (`running means/variances`) được tính toán trong quá trình huấn luyện, mang lại ước tính ổn định hơn và thường giúp đạt hiệu suất tốt hơn.
- **Data Augmentation**: Các phép biến đổi như `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing` làm cho dữ liệu huấn luyện đa dạng và khó hơn. Mô hình phải học từ các phiên bản biến đổi của ảnh, trong khi dữ liệu validation/test không trải qua các biến đổi ngẫu nhiên này, có thể khiến chúng "dễ"hơn đối với mô hình đã được huấn luyện tốt.
- **Label Smoothing**: Làm giảm độ tự tin của mô hình vào một nhãn duy nhất trong huấn luyện, có thể dẫn đến `Train Loss` cao hơn một chút và gián tiếp ảnh hưởng đến `Train Acc` so với `Val Acc` không bị ảnh hưởng bởi kỹ thuật này.

Tóm lại, hiện tượng `Train Acc` cao hơn `Val Acc` một chút là dấu hiệu tốt cho thấy các kỹ thuật `regularization` đã hoạt động hiệu quả, giúp mô hình không bị quá khớp (`overfitting`) với dữ liệu huấn luyện mà vẫn duy trì khả năng tổng quát hóa tốt.

4.3 Khả năng Overfitting/Underfitting

- **Không có dấu hiệu Overfitting rõ ràng**: Mặc dù `Train Acc` cao hơn `Val Acc`, sự chênh lệch không quá lớn và `Val Loss` vẫn giảm cho đến khi `early stopping` kích hoạt, cho thấy mô hình vẫn đang tổng quát hóa tốt. Các đường `Learning Curve` (loss và accuracy) cho thấy sự hội tụ hợp lý.
- **Không Underfitting**: Độ chính xác trên cả tập huấn luyện và kiểm định đều cao và ổn định, chứng tỏ mô hình đủ phức tạp để học được các đặc trưng cần thiết từ dữ liệu.

4.4 Độ chính xác theo lớp

Phân tích độ chính xác theo từng lớp cung cấp cái nhìn sâu sắc hơn về hiệu suất của mô hình:

- Các lớp như `car` (**93.10%**), `ship` (**92.80%**), `truck` (**92.00%**), `plane` (**89.70%**), `frog` (**89.50%**), `horse` (**88.80%**) được phân loại rất tốt. Điều này cho thấy mô hình đã học được các đặc trưng rõ ràng và phân biệt tốt các đối tượng này, đặc biệt là các phương tiện giao thông.
- Các lớp `bird` (**79.80%**) và `dog` (**78.00%**) có độ chính xác thấp hơn.
- Lớp `cat` (**70.80%**) là lớp có độ chính xác thấp nhất. Ma trận nhầm lẫn xác nhận rằng `cat` và `dog` là hai lớp thường xuyên bị nhầm lẫn nhất (214 trường hợp), điều này rất phổ biến trong các bài toán phân loại hình ảnh vật nuôi do sự tương đồng về đặc điểm thị giác.

4.5 Đánh giá về lựa chọn Hyperparameters và Kỹ thuật

- **Kiến trúc CNN 3 lớp**: Đủ mạnh để học các đặc trưng phức tạp trên CIFAR-10.
- **Data Augmentation**: Rất quan trọng để tăng cường tính đa dạng của dữ liệu và cải thiện khả năng tổng quát hóa.
- **Batch Normalization**: Giúp ổn định quá trình huấn luyện và tăng tốc độ hội tụ.
- **Dropout, Label Smoothing, Weight Decay**: Các kỹ thuật `regularization` này đã phát huy tác dụng tốt trong việc ngăn chặn `overfitting` và giúp mô hình đạt được hiệu suất tổng thể cao trên tập test.
- **ReduceLROnPlateau Scheduler**: Hiệu quả trong việc điều chỉnh `LR`, giúp mô hình thoát khỏi các điểm cực tiểu cục bộ và tìm được điểm tối ưu tốt hơn. `LR` đã giảm từ 1.0e-03 xuống 1.0e-04 và cuối cùng là 1.0e-06.

- **Early Stopping (PATIENCE=5)**: Đã giúp dừng quá trình huấn luyện tại epoch 61, tiết kiệm thời gian và ngăn chặn `overfitting` tiềm ẩn mà không cần chạy hết 300 epoch.

Chương 5

Kết luận

Bài tập đã thành công trong việc xây dựng, huấn luyện và đánh giá một mạng CNN 3 lớp tích chập để phân loại hình ảnh trên tập dữ liệu CIFAR-10 bằng PyTorch. Mô hình đạt được độ chính xác **86.20%** trên tập kiểm thử, một kết quả rất tốt. Các kỹ thuật như `Data Augmentation`, `Batch Normalization`, `Dropout`, `Label Smoothing`, `Weight Decay`, `Learning Rate Scheduling` và `Early Stopping` đã được áp dụng hiệu quả để cải thiện hiệu suất, ngăn chặn `overfitting` và đảm bảo khả năng tổng quát hóa tốt của mô hình.

Mặc dù lớp `cat` vẫn là thách thức lớn nhất đối với mô hình do sự tương đồng cao với `dog`, kết quả tổng thể cho thấy một mô hình phân loại ảnh mạnh mẽ đã được xây dựng. Để cải thiện hơn nữa, có thể xem xét:

- Phân tích sâu hơn ma trận nhầm lẫn để hiểu rõ nguyên nhân nhầm lẫn giữa các lớp.
- Thử nghiệm với các kiến trúc CNN tiên tiến hơn (ví dụ: `ResNet`, `DenseNet`, `VGG`).
- Tinh chỉnh `hyperparameters` chi tiết hơn, đặc biệt là `DROPOUT_RATE`, `LABEL_SMOOTHING`, `WEIGHT_DECAY` và các thông số của `scheduler`.
- Sử dụng các kỹ thuật `Data Augmentation` nâng cao hơn hoặc các phương pháp xử lý mất cân bằng lớp nếu cần.