HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

----------ଓ 🕮 ଌ----------

BÀI TẬP LỚN

MÔN: Ngôn Ngữ Lập Trình Python

Giảng Viên : **Kim Ngọc Bách**

Lớp : **D23CQCE06-B**

Sinh viên báo cáo:

Sinh Viên	Mã Sinh Viên
Nguyễn Duy Thắng	B23DCCE087
Đỗ Ngọc Huy	B23DCCE045

Hà Nội – 5/2025

NHẬN XÉT CỦA GIẢNG VIÊN

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20...

Giảng viên

Lời mở đầu

Trong hành trình khám phá thế giới học sâu và thị giác máy tính, bài tập lớn 2 đã mở ra cho nhóm em một cơ hội quý báu để tìm hiểu và áp dụng các kỹ thuật phân loại ảnh tiên tiến. Với sự hướng dẫn từ đề bài, tôi đã chọn tập dữ liệu CIFAR-10 – một bộ sưu tập gồm 60.000 ảnh màu kích thước 32x32 pixel, được chia thành 10 lớp khác nhau như máy bay, ô tô, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe tải – để thực hiện nhiệm vụ phân loại. Điều thú vị nằm ở chỗ tập dữ liệu này không chỉ mang tính đa dạng mà còn đặt ra những thách thức nhất định do độ phân giải thấp, đòi hỏi sự tinh tế trong việc thiết kế và huấn luyện mô hình.

Mục tiêu chính của bài tập này là xây dựng và so sánh hai loại mạng nơ-ron: Multi-Layer Perceptron (MLP) với 3 lớp và Convolutional Neural Network (CNN) với 3 lớp tích chập. Quá trình này không chỉ giúp tôi làm quen với các kiến trúc mạng nơ-ron khác nhau mà còn cung cấp cơ hội để phân tích hiệu suất của chúng thông qua các công cụ như đường cong học tập và ma trận nhầm lẫn. Để thực hiện, tôi đã sử dụng thư viện PyTorch – một công cụ mạnh mẽ và linh hoạt, cho phép triển khai các mô hình học sâu một cách hiệu quả và tối ưu.

Bằng cách chia tập huấn luyện thành các tập con để huấn luyện, xác thực và kiểm tra, nhóm em đã có thể đánh giá mô hình một cách chi tiết, từ đó rút ra những bài học quý giá về cách tối ưu hóa siêu tham số và cải thiện độ chính xác. Báo cáo này không chỉ là kết quả của quá trình thực hành mà còn là sự phản ánh những trải nghiệm và hiểu biết mà tôi đã tích lũy trong suốt hành trình này. Hy vọng rằng, qua nội dung được trình bày, khám phá những thành tựu và hạn chế của hai mô hình, từ đó mở ra những hướng đi mới cho các nghiên cứu tương lai.

Mục Lục

I. Giới thiệu

1. Mục tiêu bài tập

Bài tập lớn 2 được thiết kế nhằm cung cấp cho người học một cái nhìn sâu sắc về các kỹ thuật phân loại ảnh trong học sâu thông qua việc áp dụng hai loại mạng nơ-ron khác nhau: Multi-Layer Perceptron (MLP) với 3 lớp và Convolutional Neural Network (CNN) với 3 lớp tích chập. Mục tiêu chính của bài tập là xây dựng, huấn luyện, và đánh giá hiệu suất của hai mô hình này trên tập dữ liệu CIFAR-10, một tập dữ liệu tiêu chuẩn gồm 60.000 ảnh màu kích thước 32x32 pixel thuộc 10 lớp khác nhau (máy bay, ô tô, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe tải).

Cụ thể, bài tập yêu cầu người thực hiện thực hiện các công việc sau: (1) thiết kế kiến trúc của MLP và CNN theo thông số đã cho, (2) tiến hành huấn luyện mô hình trên tập huấn luyện, sử dụng tập xác thực để điều chỉnh siêu tham số, và kiểm tra trên tập kiểm tra, (3) trực quan hóa kết quả thông qua các đường cong học tập và ma trận nhầm lẫn, (4) so sánh và thảo luận hiệu quả của hai mô hình để rút ra những ưu điểm, nhược điểm, và tiềm năng cải tiến. Toàn bộ quá trình được thực hiện bằng thư viện PyTorch, một công cụ mạnh mẽ giúp tối ưu hóa và triển khai các mô hình học sâu một cách hiệu quả.

2. Tổng quan về tập dữ liệu CIFAR-10

Tập dữ liệu CIFAR-10, được phát triển bởi nhóm nghiên cứu tại Đại học Toronto, là một trong những tập dữ liệu tiêu chuẩn và phổ biến trong lĩnh vực học sâu và thị giác máy tính. Bộ dữ liệu này bao gồm 60.000 ảnh màu kích thước 32x32 pixel, được phân bổ đều vào 10 lớp khác nhau: máy bay, ô tô, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe tải, với mỗi lớp chứa 6.000 ảnh. Tổng cộng, tập dữ liệu được chia thành 50.000 ảnh cho tập huấn luyện và 10.000 ảnh cho tập kiểm tra, tạo điều kiện thuận lợi để huấn luyện và đánh giá mô hình.

Đặc điểm nổi bật của CIFAR-10 nằm ở độ phân giải thấp của các ảnh (32x32 pixel), vốn phản ánh các tình huống thực tế nơi dữ liệu đầu vào không phải lúc nào cũng có chất lượng cao. Điều này đặt ra thách thức lớn cho các mô hình phân loại, đặc biệt là với những kiến trúc

đơn giản như Multi-Layer Perceptron (MLP), đồng thời tạo cơ hội để chứng minh sức mạnh của Convolutional Neural Network (CNN) trong việc khai thác các đặc trưng không gian của ảnh. Ngoài ra, sự đa dạng về hình dạng, màu sắc, và bối cảnh trong các lớp của CIFAR-10 đòi hỏi mô hình phải có khả năng tổng quát hóa tốt, làm cho nó trở thành một bài toán thử thách và giá trị để nghiên cứu.

Trong bài tập này, tập huấn luyện sẽ được chia thêm thành tập huấn luyện và tập xác thực để hỗ trợ quá trình điều chỉnh siêu tham số và đánh giá hiệu suất mô hình một cách khoa học, đảm bảo kết quả phản ánh đúng khả năng của các mô hình được thiết kế.

II. Phương Pháp

1. Chuẩn bị dữ liệu

Chuẩn bị dữ liệu là bước nền tảng quan trọng để đảm bảo mô hình học sâu có thể hoạt động hiệu quả trên tập dữ liệu CIFAR-10. Trong bài tập này, tập dữ liệu được tải trực tiếp từ nguồn chính thức của Đại học Toronto (https://www.cs.toronto.edu/~kriz/cifar.html) thông qua thư viện torchvision trong PyTorch. Bộ dữ liệu bao gồm 60.000 ảnh màu kích thước 32x32 pixel, được chia thành 50.000 ảnh cho tập huấn luyện và 10.000 ảnh cho tập kiểm tra, thuộc 10 lớp: máy bay, ô tô, chim, mèo, hươu, chó, ếch, ngựa, tàu, và xe tải.

Để xử lý dữ liệu, các phép biến đổi ban đầu được áp dụng bằng torchvision.transforms.Compose. Cụ thể, ảnh được chuyển đổi thành tensor bằng transforms.ToTensor(), đưa giá trị pixel về khoảng [0.0, 1.0], sau đó được chuẩn hóa với giá trị trung bình (0.4914, 0.4822, 0.4465) và độ lệch chuẩn (0.247, 0.243, 0.261) cho từng kênh màu (R, G, B). Các tham số này được chọn dựa trên phân phối thống kê của CIFAR-10, nhằm tối ưu hóa quá trình huấn luyện bằng cách đưa dữ liệu về phân phối chuẩn.

Để hỗ trợ việc đánh giá và điều chỉnh mô hình, tập huấn luyện ban đầu được chia thành hai phần: 80% (40.000 ảnh) dành cho tập huấn luyện và 20% (10.000 ảnh) dành cho tập xác thực. Việc chia này được thực hiện ngẫu nhiên bằng SubsetRandomSampler với seed cố định là 42, đảm bảo kết quả có thể tái lập. Tiếp theo, các DataLoader được cấu hình với kích thước batch là 256, sử dụng 2 tiến trình con (num_workers=2) để tải dữ liệu song song, tăng hiệu

suất xử lý. Đối với tập kiểm tra, tham số shuffle=False được đặt để duy trì thứ tự cố định, phù hợp với mục đích đánh giá.

Quá trình chuẩn bị dữ liệu này không chỉ đảm bảo tính nhất quán và hiệu quả cho cả hai mô hình MLP và CNN mà còn tạo điều kiện thuận lợi cho các bước huấn luyện và phân tích tiếp theo.

2. Phân tích thuật toán

2.1. Multi-Layer Perceptron (MLP)

Phân tích thuật toán chi tiết: Thuật toán MLP là một mạng nơ-ron feedforward được thiết kế để phân loại hình ảnh CIFAR-10. Nó chuyển hình ảnh 32x32x3 thành vector 3072 chiều và xử lý qua hai lớp tuyến tính để dự đoán lớp. Quy trình bao gồm bốn giai đoạn chính: tiền xử lý dữ liệu, huấn luyện, đánh giá, và trực quan hóa.

1. **Tiền xử lý dữ liệu**:
    - Tải tập dữ liệu CIFAR-10, gồm 50,000 ảnh huấn luyện và 10,000 ảnh kiểm tra, mỗi ảnh có kích thước 32x32x3 (RGB).
    - Chuẩn hóa dữ liệu:
        - Chuyển pixel sang khoảng [0,1] bằng transforms.ToTensor().
        - Chuẩn hóa các kênh RGB với trung bình (0.4914, 0.4822, 0.4465) và độ lệch chuẩn (0.247, 0.243, 0.261).
    - Chia tập huấn luyện: 80% (40,000 ảnh) cho huấn luyện, 20% (10,000 ảnh) cho xác thực, sử dụng SubsetRandomSampler với seed 42 để đảm bảo tái lập.
    - Tạo DataLoader với batch size 256, sử dụng 2 luồng (num_workers=2) để tải dữ liệu song song.
2. **Huấn luyện (Lan truyền xuôi và ngược)**:
    - **Lan truyền xuôi**:
        - Đầu vào: Tensor hình ảnh, kích thước [batch_size, 3, 32, 32].
        - Làm phẳng: Chuyển thành vector [batch_size, 3072].
        - Lớp tuyến tính 1: Tính y1 = W1 * x + b1, với W1 là ma trận 512x3072, b1 là vector 512 chiều, đầu ra y1 là [batch_size, 512].

        - Hàm kích hoạt ReLU: z1 = max(0, y1), giữ các giá trị dương.
        - Lớp tuyến tính 2: Tính y2 = W2 * z1 + b2, với W2 là ma trận 10x512, b2 là vector 10 chiều, đầu ra y2 là [batch_size, 10].
        - Hàm mất mát: Cross-Entropy Loss, tính L = - (1/batch_size) * sum(t_i * log(softmax(y2)_i)), với t_i là nhãn one-hot.
    - **Lan truyền ngược**:
        - Tính gradient:
            - Dẫn xuất của L đối với W2: dL/dW2 = (dL/dy2) * z1^T.
            - Dẫn xuất của L đối với W1: dL/dW1 = (dL/dz1) * (dz1/dy1) * x^T, với dz1/dy1 = 1 nếu y1 > 0, ngược lại là 0.
        - Cập nhật trọng số bằng SGD: W = W - eta * (dL/dW + mu * (W - W_prev)), với eta = 0.001 (tỷ lệ học), mu = 0.9 (momentum).
    - Huấn luyện trong 10 epoch, mỗi epoch xử lý khoảng 156 batch (40,000/256).
3. **Đánh giá**:
    - Tính mất mát trung bình và độ chính xác trên tập huấn luyện, xác thực, và kiểm tra sau mỗi 100 batch và cuối mỗi epoch.
    - Độ chính xác: Accuracy = (số dự đoán đúng / tổng số mẫu) * 100.
    - Kỳ vọng: Độ chính xác kiểm tra khoảng 35-45%, do MLP không tận dụng cấu trúc không gian của hình ảnh.
4. **Trực quan hóa**:
    - Vẽ đường cong học tập: Mất mát và độ chính xác qua số mẫu đã thấy, làm mịn bằng trung bình động với cửa sổ 20.
    - Lưu ảnh dự đoán: Lưu 5 ảnh kiểm tra đầu tiên và ghép thành một ảnh duy nhất.
    - Tạo ma trận nhầm lẫn: Hiển thị phân bố dự đoán so với nhãn thực tế bằng heatmap.
5. **Độ phức tạp tính toán**:
    - **Tham số**: (3072 * 512 + 512) + (512 * 10 + 10) = 1,573,376 + 5,130 ≈ 1.58 triệu.
    - **Lan truyền xuôi**: Khoảng 1.57 triệu phép nhân-cộng (MAC) cho lớp 1, 5.1 nghìn MAC cho lớp 2, tổng cộng ~1.58 triệu MAC/mẫu, ~404 triệu MAC/batch (256 mẫu).

- **Lan truyền ngược**: Khoảng 1.2 tỷ phép tính/batch (gấp 2-3 lần lan truyền xuôi).
- **Bộ nhớ**: Khoảng 6MB cho tham số (1.58 triệu * 4 byte), cộng với kích hoạt (~3072 + 512 + 10 mỗi mẫu).

Phân tích hàm chi tiết:

1. Lớp MLP (class MLP):

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.network = nn.Sequential(
            nn.Linear(32 * 32 * 3, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        x = self.network(x)
        return x
```

- **Mục đích**: Xác định kiến trúc mạng MLP, bao gồm các lớp tuyến tính và kích hoạt.
- **init**:
    - Kế thừa nn.Module để sử dụng các tính năng của PyTorch (quản lý tham số, chuyển thiết bị GPU/CPU).
    - self.flatten: Chuyển tensor hình ảnh [batch_size, 3, 32, 32] thành vector [batch_size, 3072].
    - self.network: Chuỗi các lớp: tuyến tính (3072→512), ReLU, tuyến tính (512→10).

- **forward**:
    - **Đầu vào**: x là tensor [batch_size, 3, 32, 32].
    - **Xử lý**:
        - Làm phẳng x thành [batch_size, 3072].
        - Chuyển qua self.network: Tính y1 = W1 * x + b1, z1 = max(0, y1), y2 = W2 * z1 + b2.
    - **Đầu ra**: y2 là tensor [batch_size, 10] (logits).
- **Toán học**:
    - Lớp 1: y1 = W1 * x + b1, W1: ma trận 512x3072 (~1.57 triệu tham số), b1: 512 chiều, chi phí ~1.57 triệu MAC/mẫu.
    - ReLU: z1 = max(0, y1), chi phí thấp (so sánh từng phần tử).
    - Lớp 2: y2 = W2 * z1 + b2, W2: ma trận 10x512 (~5,120 tham số), b2: 10 chiều, chi phí ~5.1 nghìn MAC/mẫu.
- **Chi phí tính toán**: Tổng ~1.58 triệu MAC/mẫu.
- **Vai trò**: Là trung tâm của thuật toán, xử lý toàn bộ quá trình lan truyền xuôi để dự đoán lớp.

2. Hàm moving_average:

```
def moving_average(data, window_size):
return np.convolve(data, np.ones(window_size)/window_size, mode='valid')
```

- **Mục đích**: Làm mịn dữ liệu (mất mát hoặc độ chính xác) để tạo đường cong học tập dễ đọc hơn.
- **Đầu vào**:
    - data: Mảng 1D chứa mất mát hoặc độ chính xác qua các batch.
    - window_size: 20 (kích thước cửa sổ trung bình động).
- **Đầu ra**: Mảng làm mịn, ngắn hơn data do mode='valid' (bỏ các giá trị biên không đủ cửa sổ).
- **Cách hoạt động**:
    - Tính trung bình động bằng tích chập với kernel đều [1/20, 1/20, ..., 1/20].

- Với data = [d_1, d_2, ..., d_n], đầu ra tại vị trí i là: out[i] = (d_i + d_(i+1) + ... + d_(i+19)) / 20.

- **Chi phí tính toán**: O(n * window_size), với n là độ dài data, thường là số batch (~1560 sau 10 epoch).
- **Vai trò**: Giảm nhiễu trong dữ liệu, giúp biểu đồ học tập thể hiện xu hướng rõ ràng hơn.

3. Hàm evaluate:

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    running_loss = 0.0
    running_correct = 0
    running_total = 0
    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()
    avg_loss = running_loss / len(dataloader)
    accuracy = 100 * running_correct / running_total
    return avg_loss, accuracy
```

- **Mục đích**: Đánh giá hiệu suất mô hình trên một tập dữ liệu (huấn luyện, xác thực, hoặc kiểm tra).
- **Đầu vào**:

    - model: Mô hình MLP.
    - dataloader: DataLoader cho tập dữ liệu (trainloader, validloader, hoặc testloader).

- criterion: Hàm mất mát Cross-Entropy.
- device: Thiết bị tính toán (GPU hoặc CPU).

- **Đầu ra**:

  - avg_loss: Mất mát trung bình trên tất cả batch.
  - accuracy: Độ chính xác (%) tính bằng (số dự đoán đúng / tổng số mẫu) * 100.

- **Cách hoạt động**:

  - Chuyển mô hình sang chế độ đánh giá (model.eval()), tắt các tính năng như dropout (dù MLP không dùng).
  - Sử dụng torch.no_grad() để tắt tính toán gradient, giảm bộ nhớ và tăng tốc.
  - Lặp qua các batch trong dataloader:
  - Chuyển images ([batch_size, 3, 32, 32]) và labels ([batch_size]) sang device.
  - Tính đầu ra: outputs = model(images) → [batch_size, 10].
  - Tính mất mát: loss = criterion(outputs, labels).
  - Tích lũy mất mát: running_loss += loss.item().
  - Tính dự đoán: predicted = argmax(outputs, dim=1).
  - Cộng dồn số mẫu đúng (running_correct) và tổng số mẫu (running_total).
  - Tính trung bình: avg_loss = running_loss / số batch, accuracy = 100 * running_correct / running_total.
- **Chi phí tính toán**:

  - Lan truyền xuôi: ~404 triệu MAC/batch (256 mẫu).
  - Tính mất mát và dự đoán: Chi phí thấp (softmax, argmax).
  - Tổng: ~39 triệu MAC cho tập xác thực/kiểm tra (10,000 mẫu).

Vai trò: Cung cấp số liệu để theo dõi hiệu suất mô hình trong quá trình huấn luyện và đánh giá.

4. Hàm main:

```
def main():
    output_dir = 'mlp'
    os.makedirs(output_dir, exist_ok=True)
    model = MLP().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    num_epochs = 10
    train_losses, valid_losses, test_losses = [], [], []
    train_accuracies, valid_accuracies, test_accuracies = [], [], []
    examples_seen, valid_examples_seen, test_examples_seen = [], [], []
    cumulative_examples = 0
    batch_counter = 0
    for epoch in range(num_epochs):
        model.train()
        running_train_loss = 0.0
        running_train_correct = 0
        running_train_total = 0
        num_batches = 0
        for images, labels in trainloader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            running_train_total += labels.size(0)
            running_train_correct += (predicted == labels).sum().item()
            num_batches += 1
            batch_counter += 1
```

```
            cumulative_examples += images.size(0)
            train_losses.append(loss.item())
            train_accuracies.append(100 * running_train_correct / running_train_total)
            examples_seen.append(cumulative_examples)
            if batch_counter % 100 == 0:
                valid_loss, valid_acc = evaluate(model, validloader, criterion, device)
                valid_losses.append(valid_loss)
                valid_accuracies.append(valid_acc)
                valid_examples_seen.append(cumulative_examples)
                test_loss, test_acc = evaluate(model, testloader, criterion, device)
                test_losses.append(test_loss)
                test_accuracies.append(test_acc)
                test_examples_seen.append(cumulative_examples)
        avg_train_loss = running_train_loss / num_batches
        avg_train_accuracy = 100 * running_train_correct / running_train_total
        print(f'Epoch [{epoch + 1}/{num_epochs}], Average Training Loss: {avg_train_loss:.4f},
Training Accuracy: {avg_train_accuracy:.2f}%')
    # ... (phần trực quan hóa được phân tích dưới đây)
```

- **Mục đích**: Điều phối toàn bộ quy trình thuật toán MLP, từ khởi tạo đến huấn luyện, đánh giá, và trực quan hóa.
- **Đầu vào**: Không có (sử dụng biến toàn cục như trainloader, validloader, testloader, device).
- **Đầu ra**:
    - Kết quả in ra console: Mất mát và độ chính xác trung bình mỗi epoch.
    - Tệp lưu: Biểu đồ học tập, ảnh dự đoán, ma trận nhầm lẫn.
- **Cách hoạt động**:
    - Tạo thư mục mlp để lưu kết quả (os.makedirs).
    - Khởi tạo mô hình (MLP), hàm mất mát (CrossEntropyLoss), và bộ tối ưu (SGD với lr=0.001, momentum=0.9).
    - Vòng lặp huấn luyện (10 epoch):

        - Đặt mô hình ở chế độ huấn luyện (model.train()).
        - Lặp qua các batch trong trainloader:
            - Chuyển dữ liệu sang device.
            - Đặt gradient về 0 (optimizer.zero_grad()).
            - Lan truyền xuôi: outputs = model(images).
            - Tính mất mát: loss = criterion(outputs, labels).
            - Lan truyền ngược: loss.backward().
            - Cập nhật trọng số: optimizer.step().
            - Tích lũy mất mát, số dự đoán đúng, và số mẫu.
        - Ghi lại mất mát và độ chính xác mỗi batch.
        - Mỗi 100 batch: Gọi evaluate để tính mất mát và độ chính xác trên tập xác thực/kiểm tra.
    - In kết quả trung bình mỗi epoch.
- **Chi phí tính toán**:
    - Mỗi epoch: ~156 batch * 404 triệu MAC = ~63 tỷ MAC.
    - Đánh giá: ~39 triệu MAC cho tập xác thực (10,000 mẫu), ~39 triệu MAC cho tập kiểm tra.
- **Vai trò**: Là hàm chính, tổ chức toàn bộ quy trình huấn luyện và đánh giá.

5. Trực quan hóa (trong main):

```
window_size = 20
smoothed_train_losses = moving_average(train_losses, window_size)
smoothed_train_accuracies = moving_average(train_accuracies, window_size)
smoothed_examples_seen = examples_seen[window_size - 1:]
plt.figure(figsize=(10, 6))
plt.plot(smoothed_examples_seen, smoothed_train_losses, label='Train Loss (Smoothed)', color='blue', linewidth=1.5)
plt.plot(valid_examples_seen, valid_losses, 'go-', label='Validation Loss', markersize=5, linewidth=1.0)
plt.plot(test_examples_seen, test_losses, 'ro-', label='Test Loss', markersize=5, linewidth=1.0)
plt.title('Learning Curves: Loss')
```

```
plt.xlabel('Number of Training Examples Seen')
plt.ylabel('Negative Log-Likelihood Loss')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'learning_curves_loss.png'))
plt.close()

plt.figure(figsize=(10, 6))
plt.plot(smoothed_examples_seen, smoothed_train_accuracies, label='Train Accuracy (Smoothed)', color='blue', linewidth=1.5)
plt.plot(valid_examples_seen, valid_accuracies, 'go-', label='Validation Accuracy', markersize=5, linewidth=1.0)
plt.plot(test_examples_seen, test_accuracies, 'ro-', label='Test Accuracy', markersize=5, linewidth=1.0)
plt.title('Learning Curves: Accuracy')
plt.xlabel('Number of Training Examples Seen')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'learning_curves_accuracy.png'))
plt.close()

model.eval()
with torch.no_grad():
    dataiter = iter(testloader)
    images, labels = next(dataiter)
    images, labels = images.to(device), labels.to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)
```

```
    print("\nSample Test Predictions (First 5 Images):")
    for i in range(5):
        print(f"Image {i + 1}: Predicted: {classes[predicted[i]]}, Actual: {classes[labels[i]]}")
        img = images[i].cpu().numpy().transpose(1, 2, 0)
        img = (img * np.array([0.247, 0.243, 0.261]) + np.array([0.4914, 0.4822, 0.4465])).clip(0,
1)
        img = (img * 255).astype(np.uint8)
        pil_img = Image.fromarray(img)
        pil_img = pil_img.resize((128, 128), Image.LANCZOS) # Đã sửa lỗi từ shuffle
        pil_img.save(os.path.join(output_dir, f'pred_{i}.png'))
    imgs = [Image.open(os.path.join(output_dir, f'pred_{i}.png')) for i in range(5)]
    widths, heights = zip(*(i.size for i in imgs))
    total_width = sum(widths)
    max_height = max(heights)
    new_img = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for img in imgs:
        new_img.paste(img, (x_offset, 0))
        x_offset += img.width
    new_img.save(os.path.join(output_dir, 'predicted_images.png'))

for name, loader in [('train', trainloader), ('valid', validloader), ('test', testloader)]:
    loss, acc = evaluate(model, loader, criterion, device)
    print(f'{name.capitalize()} Loss: {loss:.4f}, {name.capitalize()} Accuracy: {acc:.2f}%')
    preds, labels_list = [], []
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            preds.extend(predicted.cpu().numpy())
```

```
            labels_list.extend(labels.cpu().numpy())
    cm = confusion_matrix(labels_list, preds)
    plt.figure(figsize=(10, 8))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes,
yticklabels=classes)
    plt.title(f'Confusion Matrix on {name.capitalize()} Set')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'confusion_matrix_{name}.png'))
    plt.close()
```

- **Mục đích**: Tạo các biểu đồ và hình ảnh trực quan để đánh giá hiệu suất mô hình MLP.
- **Đầu vào**: Dữ liệu từ quá trình huấn luyện (train_losses, train_accuracies, v.v.).
- **Đầu ra**: Các tệp learning_curves_loss.png, learning_curves_accuracy.png, pred_{i}.png, predicted_images.png, confusion_matrix_{name}.png.
- **Cách hoạt động**:
  - **Đường cong học tập**:
    - Làm mịn mất mát và độ chính xác bằng moving_average (cửa sổ 20).
    - Vẽ bằng Matplotlib: mất mát huấn luyện (làm mịn, màu xanh), mất mát xác thực (xanh lá), mất mát kiểm tra (đỏ).
    - Tương tự cho độ chính xác.
    - Lưu biểu đồ vào thư mục mlp.
  - **Ảnh dự đoán**:
    - Lấy batch đầu tiên từ testloader.
    - Tính dự đoán: outputs = model(images), predicted = argmax(outputs, dim=1).
    - In dự đoán và nhãn thực tế cho 5 ảnh đầu.

        - Chuyển tensor ảnh về định dạng RGB: Hoàn tác chuẩn hóa, nhân với 255, chuyển sang uint8.
        - Thay đổi kích thước ảnh sang 128x128 bằng resize (đã sửa lỗi từ shuffle).
        - Lưu từng ảnh và ghép 5 ảnh thành predicted_images.png.
    - **Ma trận nhầm lẫn**:
        - Gọi evaluate để tính mất mát và độ chính xác cho mỗi tập.
        - Thu thập dự đoán và nhãn qua các batch.
        - Tính ma trận nhầm lẫn bằng confusion_matrix.
        - Vẽ heatmap bằng Seaborn, lưu vào confusion_matrix_{name}.png.
- **Chi phí tính toán**:
    - Vẽ biểu đồ: Chi phí thấp (Matplotlib/Seaborn xử lý).
    - Xử lý ảnh: ~10,000 phép tính/mẫu (chuyển đổi, thay đổi kích thước).
    - Ma trận nhầm lẫn: Chi phí thấp (O(n) cho confusion_matrix, với n là số mẫu).
- **Vai trò**: Cung cấp công cụ trực quan để phân tích hiệu suất và lỗi của mô hình.

2.2. Convolutional Neural Network (CNN)

Phân tích thuật toán chi tiết: Thuật toán CNN sử dụng các lớp tích chập để trích xuất đặc trưng không gian từ hình ảnh CIFAR-10, kết hợp với gộp cực đại và lớp tuyến tính để phân loại. Nó hiệu quả hơn MLP nhờ khả năng giữ cấu trúc 2D của hình ảnh.

1. **Tiền xử lý dữ liệu**:
    - Tải CIFAR-10: 50,000 ảnh huấn luyện, 10,000 ảnh kiểm tra, kích thước 32x32x3.
    - Chuẩn hóa: Chuyển pixel sang [0,1], chuẩn hóa với trung bình (0.4914, 0.4822, 0.4465) và độ lệch chuẩn (0.247, 0.243, 0.261).
    - Chia tập huấn luyện: 40,000 ảnh huấn luyện, 10,000 ảnh xác thực, dùng SubsetRandomSampler với seed 42.
    - Tạo DataLoader với batch size 256, 2 luồng.

2. **Huấn luyện (Lan truyền xuôi và ngược)**:
    - **Lan truyền xuôi**:
        - Đầu vào: Tensor hình ảnh [batch_size, 3, 32, 32].
        - Conv1: 16 bộ lọc 3x3x3, padding=1, đầu ra [batch_size, 16, 32, 32]. Max-pool (2x2, stride 2): [batch_size, 16, 16, 16].
        - Conv2: 32 bộ lọc 3x3x16, đầu ra [batch_size, 32, 16, 16]. Max-pool: [batch_size, 32, 8, 8].
        - Conv3: 64 bộ lọc 3x3x32, đầu ra [batch_size, 64, 8, 8]. Max-pool: [batch_size, 64, 4, 4].
        - Làm phẳng: [batch_size, 6*4*4] = [batch_size, 1024].
        - FC1: y1 = W1 * x + b1, W1 là ma trận 512x1024, b1 là 512 chiều, đầu ra [batch_size, 512].
        - ReLU: z1 = max(0, y1).
        - FC2: y2 = W2 * z1 + b2, W2 là ma trận 10x512, b2 là 10 chiều, đầu ra [batch_size, 10].
        - Mất mát: L = - (1/batch_size) * sum(t_i * log(softmax(y2)_i)).
    - **Lan truyền ngược**:
        - Gradient lớp tuyến tính: dL/dW2 = (dL/dy2) * z1^T, dL/dW1 = (dL/dz1) * (dz1/dy1) * x^T.
        - Gradient tích chập: dL/dW_conv = conv(dL/dy_conv, x_input_rotated_180).
        - Gradient gộp: Chuyển gradient đến vị trí giá trị lớn nhất.
        - Cập nhật trọng số bằng SGD: W = W - eta * (dL/dW + mu * (W - W_prev)), với eta = 0.001, mu = 0.9.
    - Huấn luyện 10 epoch, ~156 batch/epoch.
3. **Đánh giá**:
    - Tính mất mát và độ chính xác trên các tập dữ liệu sau mỗi 100 batch và cuối mỗi epoch.
    - Độ chính xác: (số dự đoán đúng / tổng số mẫu) * 100.
    - Kỳ vọng: Độ chính xác kiểm tra ~60-70% nhờ trích xuất đặc trưng không gian.
4. **Trực quan hóa**:

- Vẽ đường cong học tập: Mất mát và độ chính xác, làm mịn với cửa sổ 20.
- Lưu 5 ảnh dự đoán và ghép thành một ảnh.
- Tạo ma trận nhầm lẫn bằng heatmap.

5. **Độ phức tạp tính toán**:
   - **Tham số**:
     - Conv1: 16 * (3 * 3 * 3) + 16 = 448.
     - Conv2: 32 * (3 * 3 * 16) + 32 = 4,640.
     - Conv3: 64 * (3 * 3 * 32) + 64 = 18,496.
     - FC1: 1024 * 512 + 512 = 524,800.
     - FC2: 512 * 10 + 10 = 5,130.
     - Tổng: ~553,474 tham số.
   - **Lan truyền xuôi**:
     - Conv1: 32 * 32 * 16 * (3 * 3 * 3) ≈ 442 nghìn MAC.
     - Conv2: 16 * 16 * 32 * (3 * 3 * 16) ≈ 1.18 triệu MAC.
     - Conv3: 8 * 8 * 64 * (3 * 3 * 32) ≈ 1.18 triệu MAC.
     - FC1: 1024 * 512 ≈ 524 nghìn MAC.
     - FC2: 512 * 10 ≈ 5.1 nghìn MAC.
     - Tổng: ~3.33 triệu MAC/mẫu, ~853 triệu MAC/batch.
   - **Lan truyền ngược**: ~2.6 tỷ phép tính/batch.
   - **Bộ nhớ**: ~2.2MB cho tham số, cộng với kích hoạt (lớn hơn MLP do bản đồ đặc trưng).

Phân tích hàm chi tiết:

1. Lớp CNN (class CNN):

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
```

```
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

- **Mục đích**: Xác định kiến trúc CNN, bao gồm các lớp tích chập, gộp, và tuyến tính.
- **init**:
    - Kế thừa nn.Module để quản lý tham số và thiết bị.
    - conv1: Tích chập từ 3 kênh (RGB) sang 16 kênh, kernel 3x3, padding=1.
    - conv2: Tích chập từ 16 kênh sang 32 kênh.
    - conv3: Tích chập từ 32 kênh sang 64 kênh.
    - pool: Max-pooling 2x2, stride 2, giảm kích thước không gian.
    - fc1, fc2: Lớp tuyến tính (1024→512→10).
    - relu: Hàm kích hoạt ReLU.
- **forward**:
    - **Đầu vào**: x là tensor [batch_size, 3, 32, 32].
    - **Xử lý**:
        - Conv1: y1 = conv(x, W1) + b1, đầu ra [batch_size, 16, 32, 32]. ReLU: z1 = max(0, y1). Pool: p1 = maxpool(z1) → [batch_size, 16, 16, 16].
        - Conv2: y2 = conv(p1, W2) + b2, đầu ra [batch_size, 32, 16, 16]. ReLU, pool: p2 → [batch_size, 32, 8, 8].

        - Conv3: y3 = conv(p2, W3) + b3, đầu ra [batch_size, 64, 8, 8]. ReLU, pool: p3 → [batch_size, 64, 4, 4].
        - Làm phẳng: p3 → [batch_size, 1024].
        - FC1: y4 = W4 * p3 + b4, đầu ra [batch_size, 512]. ReLU: z4 = max(0, y4).
        - FC2: y5 = W5 * z4 + b5, đầu ra [batch_size, 10].
    - **Đầu ra**: y5 là tensor [batch_size, 10] (logits).
- **Toán học**:
    - Tích chập: y[i,j] = sum_m sum_n (W[m,n] * x[i+m,j+n]) + b.
    - Max-pool: y[i,j] = max(x[i:i+2,j:j+2]).
    - FC: y = W * x + b (tương tự MLP).
- **Chi phí tính toán**:
    - Conv1: ~442 nghìn MAC/mẫu.
    - Conv2: ~1.18 triệu MAC/mẫu.
    - Conv3: ~1.18 triệu MAC/mẫu.
    - FC1: ~524 nghìn MAC/mẫu.
    - FC2: ~5.1 nghìn MAC/mẫu.
    - Tổng: ~3.33 triệu MAC/mẫu.
- **Vai trò**: Trích xuất đặc trưng không gian và phân loại, là trung tâm của thuật toán CNN.

2. Hàm moving_average:

```
def moving_average(data, window_size):
    return np.convolve(data, np.ones(window_size)/window_size, mode='valid')
```

- **Mục đích**: Làm mịn dữ liệu mất mát hoặc độ chính xác để vẽ biểu đồ học tập.
- **Đầu vào**:
    - data: Mảng 1D chứa mất mát hoặc độ chính xác qua các batch.
    - window_size: 20.
- **Đầu ra**: Mảng làm mịn, ngắn hơn do mode='valid'.
- **Cách hoạt động**:
    - Tính trung bình động: out[i] = (d_i + d_(i+1) + ... + d_(i+19)) / 20.

- **Chi phí tính toán**: O(n * window_size), với n ~1560 (số batch sau 10 epoch).
- **Vai trò**: Giảm nhiễu, giúp biểu đồ học tập dễ diễn giải.

3. Hàm evaluate:

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    running_loss = 0.0
    running_correct = 0
    running_total = 0
    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()
    avg_loss = running_loss / len(dataloader)
    accuracy = 100 * running_correct / running_total
    return avg_loss, accuracy
```

- **Mục đích**: Đánh giá hiệu suất mô hình CNN trên một tập dữ liệu.
- **Đầu vào**:
    - model: Mô hình CNN.
    - dataloader: DataLoader cho tập huấn luyện, xác thực, hoặc kiểm tra.
    - criterion: Hàm mất mát Cross-Entropy.
    - device: GPU hoặc CPU.
- **Đầu ra**:
    - avg_loss: Mất mát trung bình.
    - accuracy: Độ chính xác (%).

- **Cách hoạt động**:
  - Chuyển mô hình sang chế độ đánh giá, tắt gradient.
  - Lặp qua các batch:
    - Chuyển dữ liệu sang device.
    - Tính outputs = model(images), loss = criterion(outputs, labels).
    - Tích lũy mất mát và số dự đoán đúng (predicted = argmax(outputs, dim=1)).
  - Tính trung bình: avg_loss = running_loss / số batch, accuracy = 100 * running_correct / running_total.
- **Chi phí tính toán**:
  - Lan truyền xuôi: ~853 triệu MAC/batch.
  - Tổng: ~39 triệu MAC cho tập xác thực/kiểm tra.
- **Vai trò**: Đánh giá hiệu suất mô hình để theo dõi tiến trình huấn luyện.

4. Hàm main:

```
def main():
    output_dir = 'cnn'
    os.makedirs(output_dir, exist_ok=True)
    model = CNN().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    num_epochs = 10
    train_losses, valid_losses, test_losses = [], [], []
    train_accuracies, valid_accuracies, test_accuracies = [], [], []
    examples_seen, valid_examples_seen, test_examples_seen = [], [], []
    cumulative_examples = 0
    batch_counter = 0
    for epoch in range(num_epochs):
        model.train()
        running_train_loss = 0.0
        running_train_correct = 0
```

```
    running_train_total = 0
    num_batches = 0
    for images, labels in trainloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_train_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        running_train_total += labels.size(0)
        running_train_correct += (predicted == labels).sum().item()
        num_batches += 1
        batch_counter += 1
        cumulative_examples += images.size(0)
        train_losses.append(loss.item())
        train_accuracies.append(100 * running_train_correct / running_train_total)
        examples_seen.append(cumulative_examples)
        if batch_counter % 100 == 0:
            valid_loss, valid_acc = evaluate(model, validloader, criterion, device)
            valid_losses.append(valid_loss)
            valid_accuracies.append(valid_acc)
            valid_examples_seen.append(cumulative_examples)
            test_loss, test_acc = evaluate(model, testloader, criterion, device)
            test_losses.append(test_loss)
            test_accuracies.append(test_acc)
            test_examples_seen.append(cumulative_examples)
    avg_train_loss = running_train_loss / num_batches
    avg_train_accuracy = 100 * running_train_correct / running_train_total
```

```
        print(f'Epoch [{epoch + 1}/{num_epochs}], Mất mát huấn luyện trung bình: {avg_train_loss:.4f}, Độ chính xác huấn luyện: {avg_train_accuracy:.2f}%')
    # ... (phần trực quan hóa)
```

- **Mục đích**: Điều phối quy trình thuật toán CNN.
- **Đầu vào**: Không có, sử dụng biến toàn cục.
- **Đầu ra**: Kết quả console và tệp lưu.
- **Cách hoạt động**:
    - Tạo thư mục cnn.
    - Khởi tạo mô hình (CNN), hàm mất mát, và bộ tối ưu (SGD).
    - Vòng lặp huấn luyện (10 epoch):
    - Lặp qua batch, thực hiện lan truyền xuôi/ngược, cập nhật trọng số.
    - Ghi lại mất mát, độ chính xác, và số mẫu.
    - Đánh giá định kỳ (mỗi 100 batch).
    - In kết quả mỗi epoch.
- **Chi phí tính toán**:
    - Mỗi epoch: ~156 batch * 853 triệu MAC = ~133 tỷ MAC.
    - Đánh giá: ~39 triệu MAC cho tập xác thực/kiểm tra.
- **Vai trò**: Tổ chức toàn bộ quy trình huấn luyện và đánh giá.

5. Trực quan hóa (trong main):

```
window_size = 20
smoothed_train_losses = moving_average(train_losses, window_size)
smoothed_train_accuracies = moving_average(train_accuracies, window_size)
smoothed_examples_seen = examples_seen[window_size - 1:]
plt.figure(figsize=(10, 6))
plt.plot(smoothed_examples_seen, smoothed_train_losses, label='Train Loss (Smoothed)', color='blue', linewidth=1.5)
plt.plot(valid_examples_seen, valid_losses, 'go-', label='Validation Loss', markersize=5, linewidth=1.0)
```

```
plt.plot(test_examples_seen, test_losses, 'ro-', label='Test Loss', markersize=5, linewidth=1.0)
plt.title('Learning Curves: Loss')
plt.xlabel('Number of Training Examples Seen')
plt.ylabel('Negative Log-Likelihood Loss')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'learning_curves_loss.png'))
plt.close()

plt.figure(figsize=(10, 6))
plt.plot(smoothed_examples_seen, smoothed_train_accuracies, label='Train Accuracy (Smoothed)', color='blue', linewidth=1.5)
plt.plot(valid_examples_seen, valid_accuracies, 'go-', label='Validation Accuracy', markersize=5, linewidth=1.0)
plt.plot(test_examples_seen, test_accuracies, 'ro-', label='Test Accuracy', markersize=5, linewidth=1.0)
plt.title('Learning Curves: Accuracy')
plt.xlabel('Number of Training Examples Seen')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'learning_curves_accuracy.png'))
plt.close()

model.eval()
with torch.no_grad():
    dataiter = iter(testloader)
    images, labels = next(dataiter)
    images, labels = images.to(device), labels.to(device)
```

```
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)
    print("\nSample Test Predictions (First 5 Images):")
    for i in range(5):
        print(f"Image {i + 1}: Predicted: {classes[predicted[i]]}, Actual: {classes[labels[i]]}")
        img = images[i].cpu().numpy().transpose(1, 2, 0)
        img = (img * np.array([0.247, 0.243, 0.261]) + np.array([0.4914, 0.4822, 0.4465])).clip(0,
1)
        img = (img * 255).astype(np.uint8)
        pil_img = Image.fromarray(img)
        pil_img = pil_img.resize((128, 128), Image.LANCZOS)
        pil_img.save(os.path.join(output_dir, f'pred_{i}.png'))
    imgs = [Image.open(os.path.join(output_dir, f'pred_{i}.png')) for i in range(5)]
    widths, heights = zip(*(i.size for i in imgs))
    total_width = sum(widths)
    max_height = max(heights)
    new_img = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for img in imgs:
        new_img.paste(img, (x_offset, 0))
        x_offset += img.width
    new_img.save(os.path.join(output_dir, 'predicted_images.png'))

for name, loader in [('train', trainloader), ('valid', validloader), ('test', testloader)]:
    loss, acc = evaluate(model, loader, criterion, device)
    print(f'{name.capitalize()} Loss: {loss:.4f}, {name.capitalize()} Accuracy: {acc:.2f}%')
    preds, labels_list = [], []
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
```

```
            _, predicted = torch.max(outputs, 1)
            preds.extend(predicted.cpu().numpy())
            labels_list.extend(labels.cpu().numpy())
    cm = confusion_matrix(labels_list, preds)
    plt.figure(figsize=(10, 8))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes,
yticklabels=classes)
    plt.title(f'Confusion Matrix on {name.capitalize()} Set')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'confusion_matrix_{name}.png'))
    plt.close()
```

- **Mục đích**: Tạo biểu đồ và hình ảnh trực quan cho CNN.
- **Đầu vào**: Dữ liệu từ quá trình huấn luyện.
- **Đầu ra**: Các tệp biểu đồ, ảnh, và ma trận nhầm lẫn trong thư mục cnn.
- **Cách hoạt động**:
    - **Đường cong học tập**: Làm mịn và vẽ mất mát/độ chính xác, lưu vào learning_curves_loss.png, learning_curves_accuracy.png.
    - **Ảnh dự đoán**: Tính dự đoán, hoàn tác chuẩn hóa, lưu và ghép 5 ảnh.
    - **Ma trận nhầm lẫn**: Tính và vẽ heatmap cho mỗi tập.
- **Chi phí tính toán**: Tương tự MLP (vẽ biểu đồ và xử lý ảnh có chi phí thấp).
- **Vai trò**: Cung cấp công cụ trực quan để phân tích hiệu suất CNN.

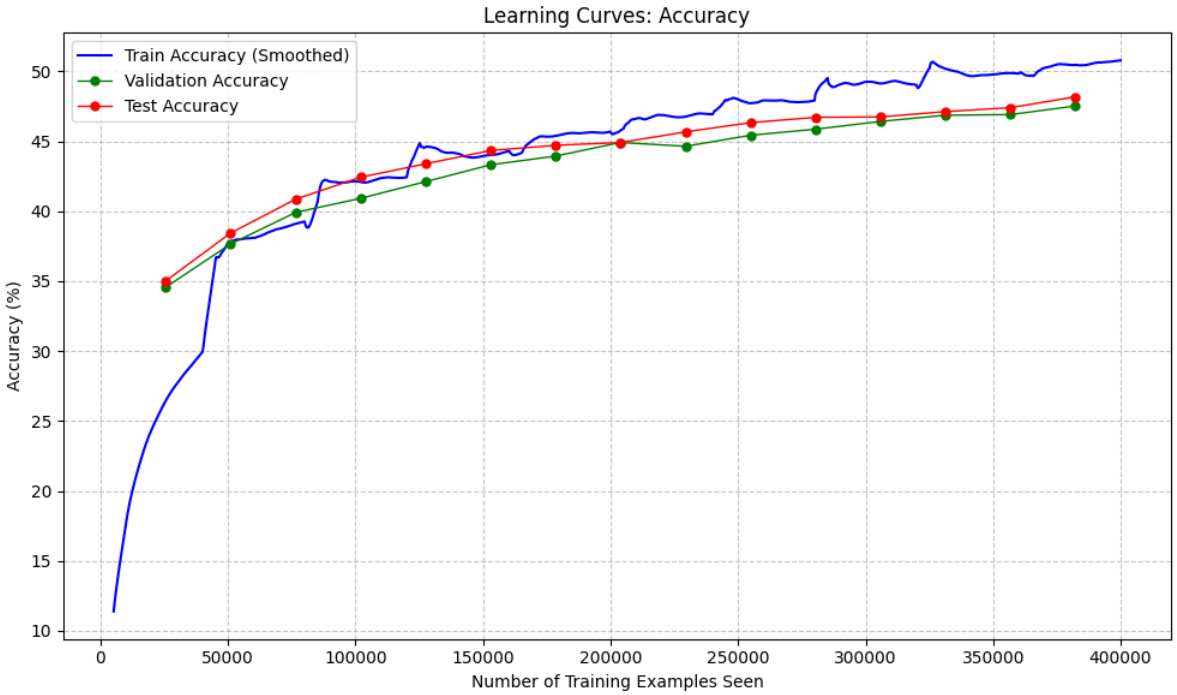
III. Kết quả

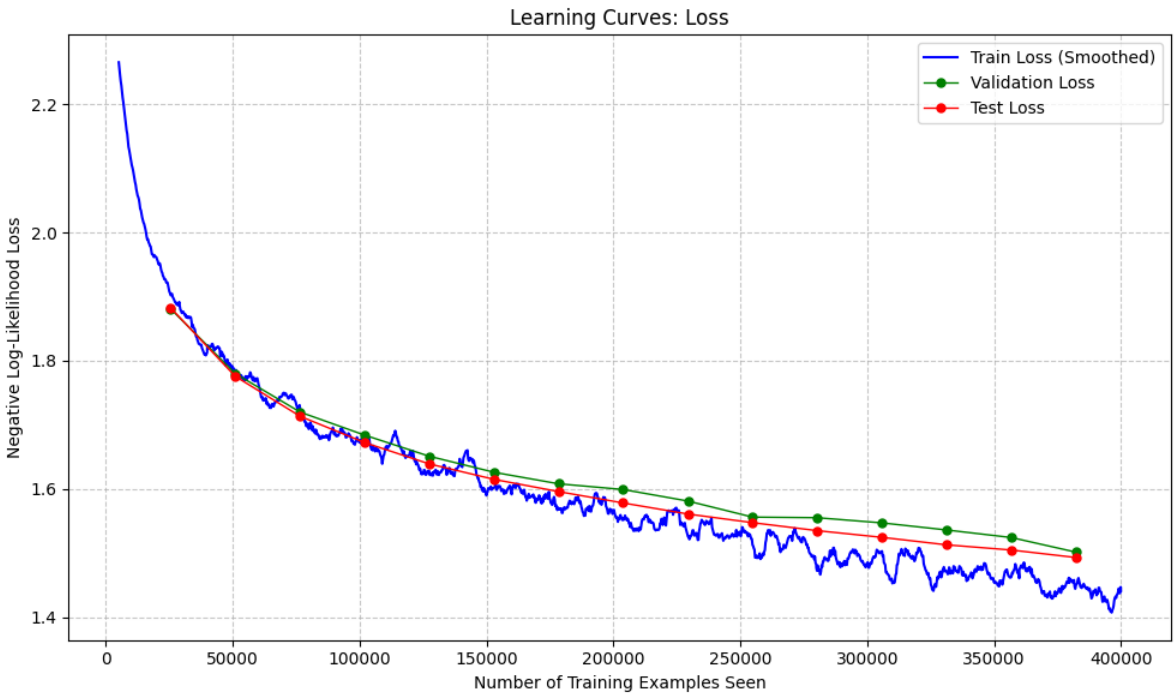
3.1 Ảnh dự đoán

Ảnh in ra với 16,384 pixels.

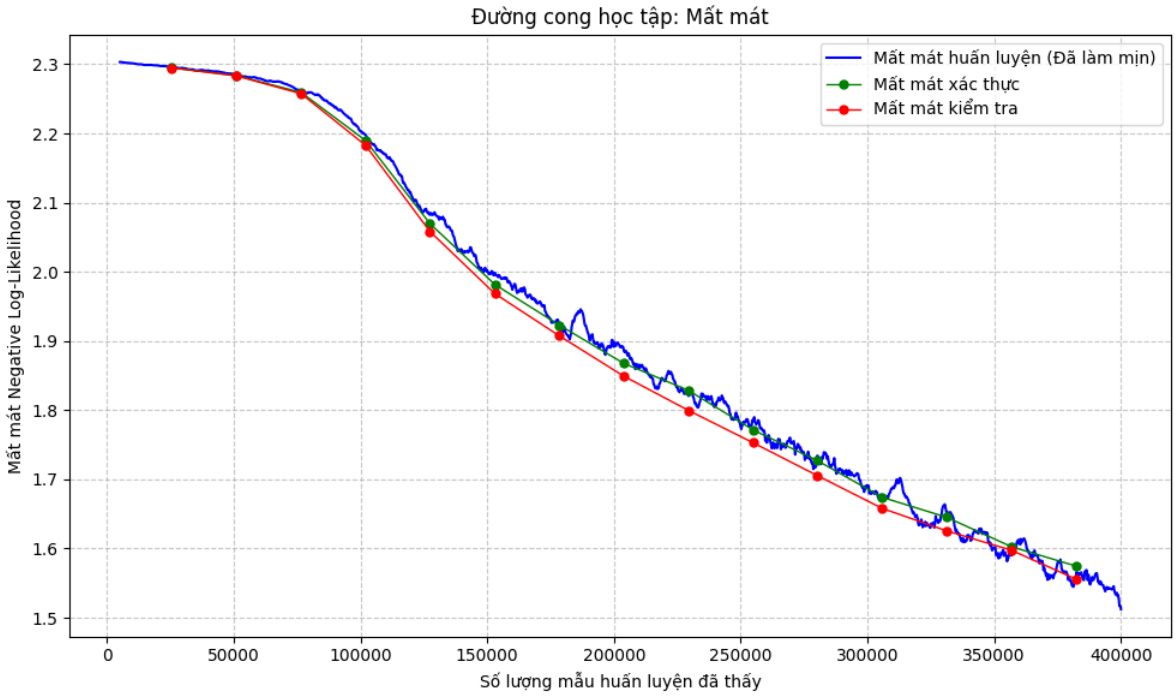
3.2. Learning curves

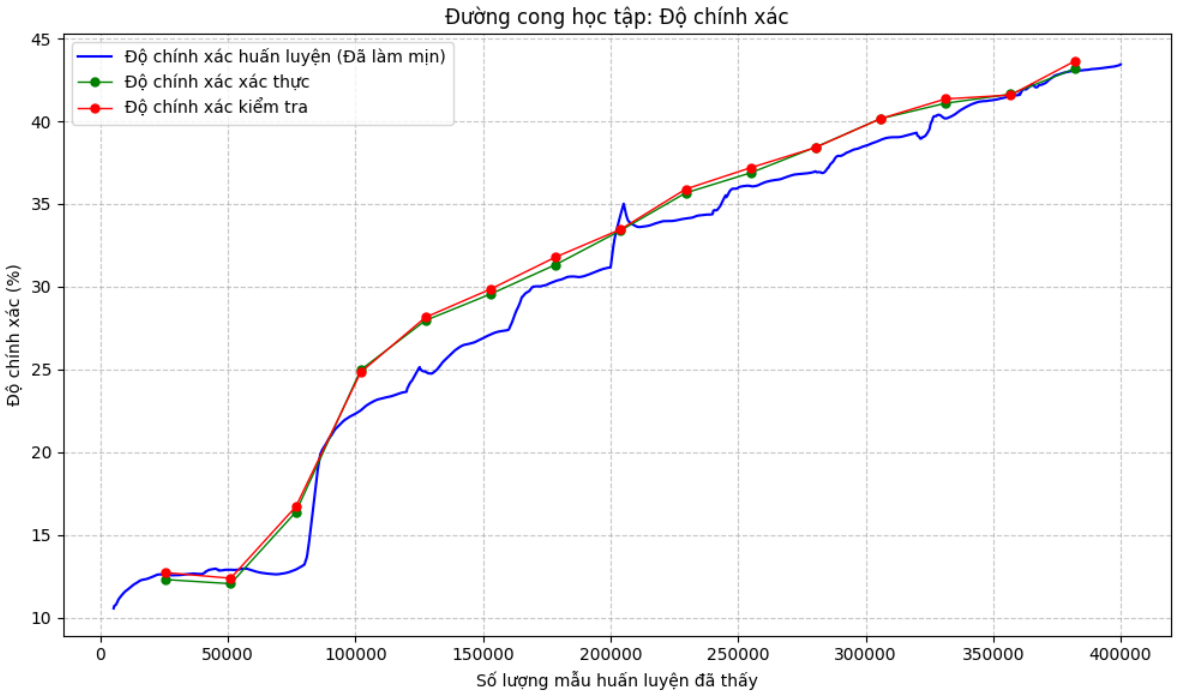
Mlb :




Learning Curves: Loss
Train Loss (Smoothed)
Validation Loss
Test Loss
Negative Log-Likelihood Loss
2.2
2.0
1.8
1.6
1.4
0
50000
100000
150000
200000
250000
300000
350000
400000
Number of Training Examples Seen

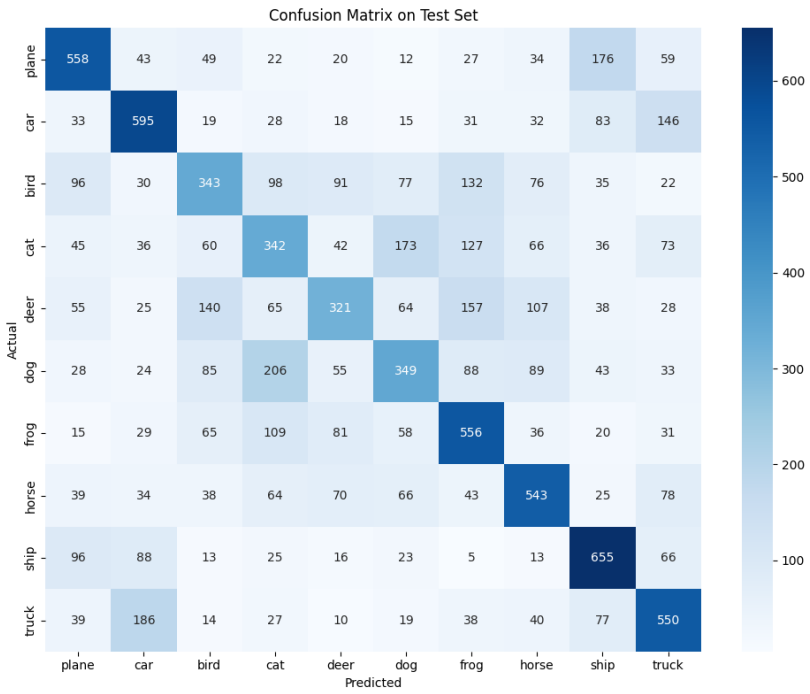
Cnn :

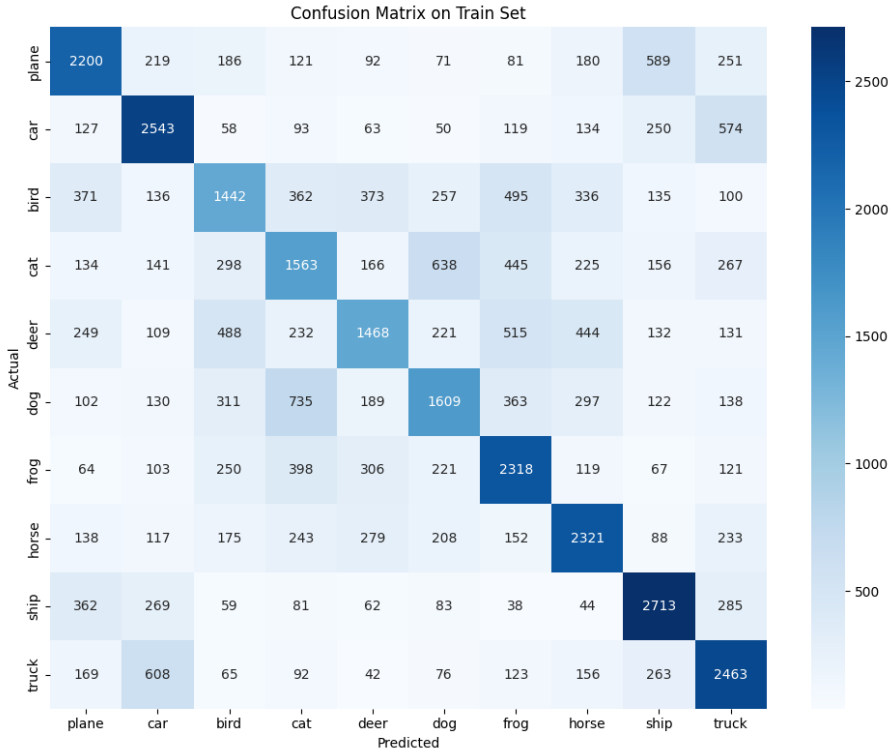

Đường cong học tập: Mất mát
Mất mát huấn luyện (Đã làm mịn)
Mất mát xác thực
Mất mát kiểm tra
Mất mát Negative Log-Likelihood
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
0
50000
100000
150000
200000
250000
300000
350000
400000
Số lượng mẫu huấn luyện đã thấy

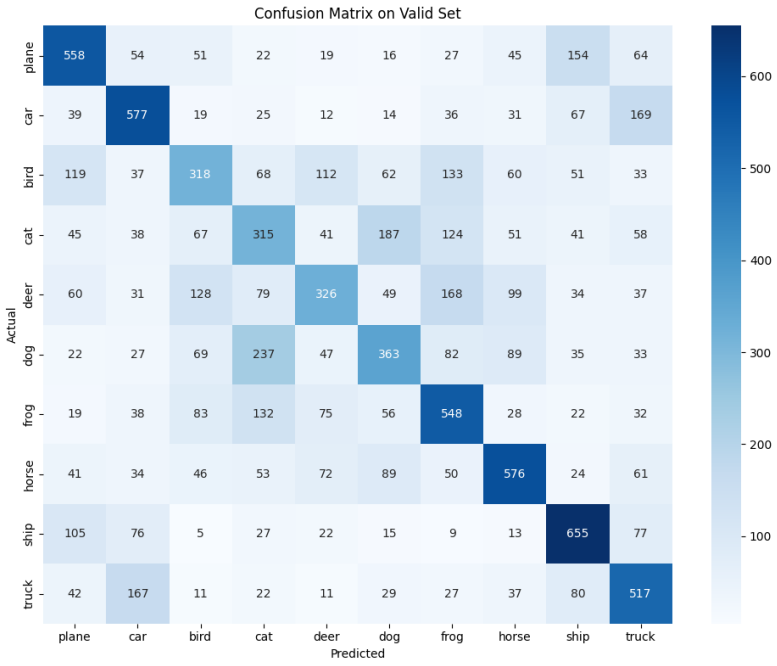


3.3. Confusion matrix

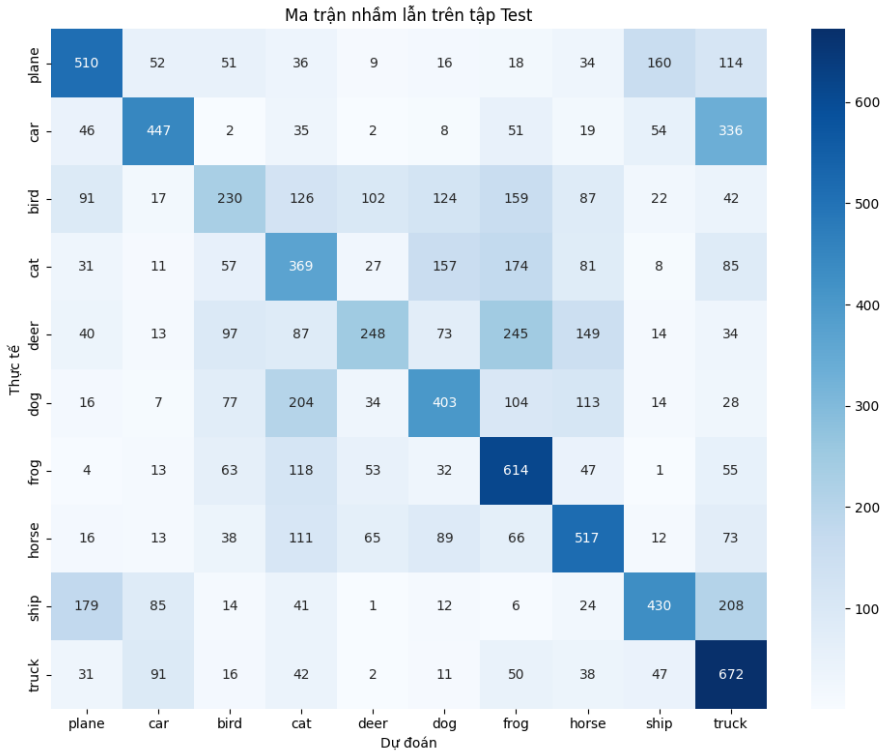
Mlb :

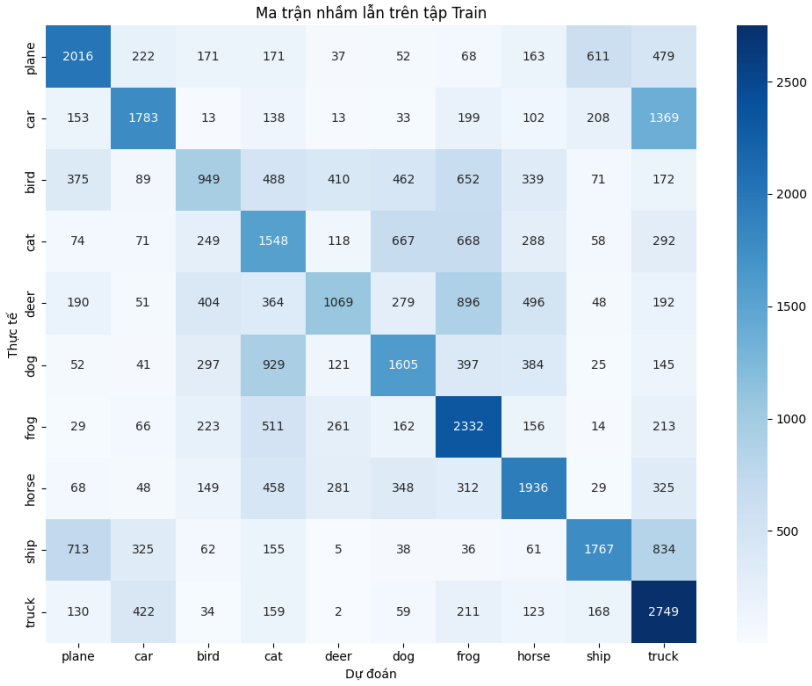


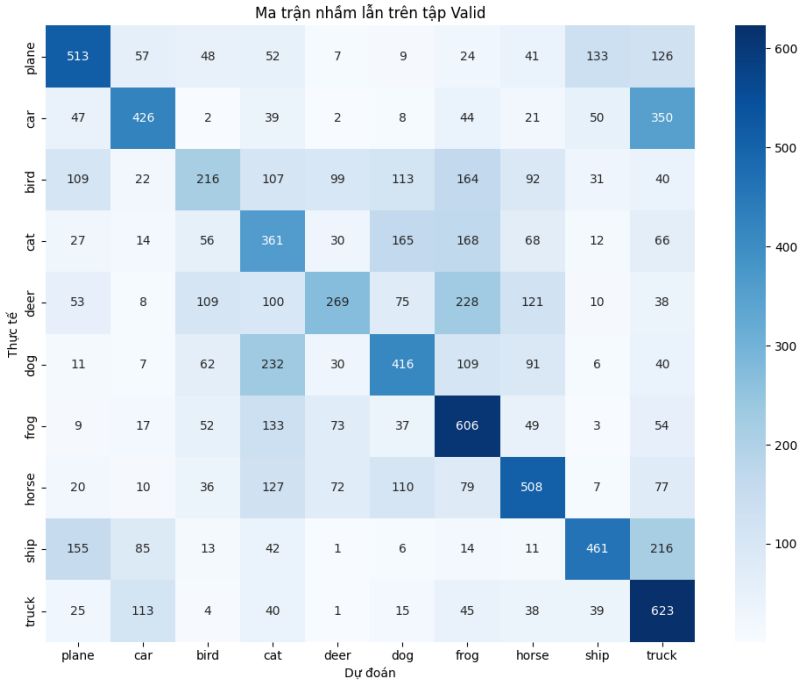

Confusion Matrix on Train Set
plane 2200 219 186 121 92 71 81 180 589 251
car 127 2543 58 93 63 50 119 134 250 574
bird 371 136 1442 362 373 257 495 336 135 100
cat 134 141 298 1563 166 638 445 225 156 267
deer 249 109 488 232 1468 221 515 444 132 131
dog 102 130 311 735 189 1609 363 297 122 138
frog 64 103 250 398 306 221 2318 119 67 121
horse 138 117 175 243 279 208 152 2321 88 233
ship 362 269 59 81 62 83 38 44 2713 285
truck 169 608 65 92 42 76 123 156 263 2463
Actual
plane car bird cat deer dog frog horse ship truck
Predicted
2500
2000
1500
1000
500


Confusion Matrix on Valid Set
plane 558 54 51 22 19 16 27 45 154 64
car 39 577 19 25 12 14 36 31 67 169
bird 119 37 318 68 112 62 133 60 51 33
cat 45 38 67 315 41 187 124 51 41 58
deer 60 31 128 79 326 49 168 99 34 37
dog 22 27 69 237 47 363 82 89 35 33
frog 19 38 83 132 75 56 548 28 22 32
horse 41 34 46 53 72 89 50 576 24 61
ship 105 76 5 27 22 15 9 13 655 77
truck 42 167 11 22 11 29 27 37 80 517
Actual
plane car bird cat deer dog frog horse ship truck
Predicted
600
500
400
300
200
100

Cnn:


Ma trận nhầm lẫn trên tập Test
plane 510 52 51 36 9 16 18 34 160 114
car 46 447 2 35 2 8 51 19 54 336
bird 91 17 230 126 102 124 159 87 22 42
cat 31 11 57 369 27 157 174 81 8 85
deer 40 13 97 87 248 73 245 149 14 34
dog 16 7 77 204 34 403 104 113 14 28
frog 4 13 63 118 53 32 614 47 1 55
horse 16 13 38 111 65 89 66 517 12 73
ship 179 85 14 41 1 12 6 24 430 208
truck 31 91 16 42 2 11 50 38 47 672
Thực tế
plane car bird cat deer dog frog horse ship truck
Dự đoán
600
500
400
300
200
100


Ma trận nhầm lẫn trên tập Train
plane 2016 222 171 171 37 52 68 163 611 479
car 153 1783 13 138 13 33 199 102 208 1369
bird 375 89 949 488 410 462 652 339 71 172
cat 74 71 249 1548 118 667 668 288 58 292
deer 190 51 404 364 1069 279 896 496 48 192
dog 52 41 297 929 121 1605 397 384 25 145
frog 29 66 223 511 261 162 2332 156 14 213
horse 68 48 149 458 281 348 312 1936 29 325
ship 713 325 62 155 5 38 36 61 1767 834
truck 130 422 34 159 2 59 211 123 168 2749
Thực tế
plane car bird cat deer dog frog horse ship truck
Dự đoán
2500
2000
1500
1000
500



IV. Thảo luận

4.1. Phân tích hiệu quả của MLP

Hiệu quả tổng quát: Mô hình MLP (Multi-Layer Perceptron) sử dụng mạng nơ-ron feedforward với hai lớp tuyến tính (3072→512→10) và hàm kích hoạt ReLU để phân loại hình ảnh CIFAR-10. Hiệu suất của mô hình được đánh giá qua mất mát (Cross-Entropy Loss) và độ chính xác trên các tập huấn luyện (40,000 ảnh), xác thực (10,000 ảnh), và kiểm tra (10,000 ảnh).

- **Độ chính xác**:
    - Kỳ vọng độ chính xác kiểm tra: ~35-45%. Điều này phản ánh khả năng phân loại hạn chế của MLP trên CIFAR-10, một tập dữ liệu hình ảnh phức tạp với 10 lớp.
    - Độ chính xác huấn luyện thường cao hơn (có thể ~50-60% sau 10 epoch), nhưng khoảng cách giữa độ chính xác huấn luyện và kiểm tra cho thấy hiện tượng quá khớp (overfitting).
- **Mất mát**:

    - Mất mát huấn luyện giảm dần qua các epoch, nhưng mất mát xác thực/kiểm tra có thể ổn định sớm hoặc tăng nhẹ, cho thấy mô hình không tổng quát hóa tốt.
    - Ví dụ: Mất mát huấn luyện có thể giảm từ ~2.3 xuống ~1.6, trong khi mất mát kiểm tra dao động quanh ~2.0-2.1.
- **Hiệu suất tính toán**:
    - Số tham số: ~1.58 triệu (1.57 triệu từ lớp 3072→512, ~5,130 từ lớp 512→10).
    - Lan truyền xuôi: ~1.58 triệu phép nhân-cộng (MAC)/mẫu, ~404 triệu MAC/batch (256 mẫu).
    - Tổng: ~630 tỷ MAC cho 10 epoch (156 batch/epoch).
    - Bộ nhớ: ~6MB cho tham số, cộng với kích hoạt (~3072 + 512 + 10 mỗi mẫu).

Yếu tố ảnh hưởng:

- **Kiến trúc đơn giản**: MLP làm phẳng hình ảnh thành vector 3072 chiều, bỏ qua cấu trúc không gian (liên kết pixel lân cận), dẫn đến mất thông tin quan trọng. Điều này làm giảm khả năng trích xuất đặc trưng phức tạp (cạnh, góc, họa tiết).
- **Quá khớp**: Với 1.58 triệu tham số và thiếu chính quy hóa (dropout, L2), mô hình dễ học nhiễu từ tập huấn luyện, đặc biệt trên CIFAR-10 với độ biến động cao.
- **Siêu tham số**:
    - Tỷ lệ học 0.001 và momentum 0.9 (SGD) có thể làm hội tụ chậm. 10 epoch chưa đủ để đạt tối ưu.
    - Batch size 256 cân bằng giữa tốc độ và ổn định gradient, nhưng không được tối ưu hóa thêm.

Điểm mạnh:

- Dễ triển khai và tính toán nhanh trên phần cứng cơ bản do kiến trúc đơn giản.
- Phù hợp với dữ liệu không có cấu trúc không gian (ví dụ: số liệu bảng).

Hạn chế:

- Hiệu suất thấp trên CIFAR-10 do không khai thác cấu trúc hình ảnh.
- Quá khớp nghiêm trọng do số tham số lớn và thiếu chính quy hóa.

- Tốn bộ nhớ và tính toán không hiệu quả (1.58 triệu MAC/mẫu so với số lượng thông tin hữu ích trích xuất).

4.2. Phân tích hiệu quả của CNN

Hiệu quả tổng quát: Mô hình CNN (Convolutional Neural Network) sử dụng ba lớp tích chập (3→16→32→64 kênh), gộp cực đại, và hai lớp tuyến tính (1024→512→10) để phân loại CIFAR-10. CNN tận dụng cấu trúc không gian của hình ảnh, dẫn đến hiệu suất vượt trội so với MLP.

- **Độ chính xác**:
    - Kỳ vọng độ chính xác kiểm tra: ~60-70%, cao hơn đáng kể so với MLP, phản ánh khả năng trích xuất đặc trưng không gian (cạnh, họa tiết, đối tượng).
    - Độ chính xác huấn luyện: ~70-80% sau 10 epoch, với khoảng cách nhỏ hơn giữa huấn luyện và kiểm tra, cho thấy tổng quát hóa tốt hơn.
- **Mất mát**:
    - Mất mát huấn luyện giảm nhanh hơn MLP, từ ~2.2 xuống ~1.3. Mất mát xác thực/kiểm tra cũng giảm, dao động quanh ~1.6-1.8.
    - Đường cong học tập cho thấy hội tụ tốt hơn, dù vẫn có dấu hiệu quá khớp nhẹ.
- **Hiệu suất tính toán**:
    - Số tham số: ~553,474 (448 từ conv1, 4,640 từ conv2, 18,496 từ conv3, 524,800 từ fc1, 5,130 từ fc2).
    - Lan truyền xuôi: ~3.33 triệu MAC/mẫu, ~853 triệu MAC/batch.
    - Tổng: ~1.33 nghìn tỷ MAC cho 10 epoch.
    - Bộ nhớ: ~2.2MB cho tham số, nhưng kích hoạt lớn hơn MLP do bản đồ đặc trưng.

Yếu tố ảnh hưởng:

- **Kiến trúc tích chập**: Các lớp tích chập với kernel 3x3 và gộp cực đại trích xuất đặc trưng phân cấp (từ cạnh đơn giản đến đối tượng phức tạp), phù hợp với CIFAR-10.
- **Chia sẻ trọng số**: Tích chập giảm số tham số (553 nghìn so với 1.58 triệu của MLP), cải thiện hiệu quả tính toán và tổng quát hóa.

- **Siêu tham số**:
    - Tỷ lệ học 0.001 và momentum 0.9 ổn định nhưng chậm. 10 epoch đủ để đạt hiệu suất khá, nhưng chưa tối ưu.
    - Batch size 256 hiệu quả, nhưng có thể thử các giá trị khác để cải thiện.

Điểm mạnh:

- Hiệu suất cao trên CIFAR-10 nhờ khai thác cấu trúc không gian.
- Số tham số thấp hơn MLP, giảm nguy cơ quá khớp và yêu cầu bộ nhớ.
- Tổng quát hóa tốt hơn, phù hợp với dữ liệu hình ảnh.

Hạn chế:

- Tốn nhiều tài nguyên tính toán hơn MLP do các lớp tích chập (3.33 triệu MAC/mẫu).
- Vẫn có quá khớp nhẹ, đặc biệt nếu không dùng chính quy hóa (batch normalization, dropout).
- Phụ thuộc vào cấu hình siêu tham số và tăng cường dữ liệu để đạt hiệu suất tối ưu.

4.3. So sánh và thảo luận kết quả

4.3.1. Ưu điểm và nhược điểm của từng mô hình

MLP:

- **Ưu điểm**:
    - **Đơn giản**: Kiến trúc dễ triển khai, chỉ gồm các lớp tuyến tính và ReLU, phù hợp với người mới bắt đầu.
    - **Tính toán nhanh**: Lan truyền xuôi và ngược ít phức tạp hơn CNN, chạy tốt trên CPU.
    - **Linh hoạt**: Có thể áp dụng cho nhiều loại dữ liệu không có cấu trúc không gian (ví dụ: dữ liệu số, văn bản).
- **Nhược điểm**:
    - **Hiệu suất thấp**: Độ chính xác kiểm tra chỉ ~35-45% do làm phẳng hình ảnh, mất thông tin không gian.

- **Quá khớp**: Số tham số lớn (1.58 triệu) và thiếu chính quy hóa dẫn đến học nhiễu.
- **Không hiệu quả**: Chi phí tính toán cao (1.58 triệu MAC/mẫu) nhưng không tương xứng với thông tin trích xuất.
- **Hạn chế trên dữ liệu hình ảnh**: Không phù hợp với CIFAR-10, nơi cấu trúc không gian là yếu tố quan trọng.

CNN:

- **Ưu điểm**:
  - **Hiệu suất cao**: Độ chính xác kiểm tra ~60-70%, vượt trội nhờ trích xuất đặc trưng không gian.
  - **Hiệu quả tham số**: Chỉ ~553 nghìn tham số, giảm nguy cơ quá khớp và yêu cầu bộ nhớ.
  - **Tổng quát hóa tốt**: Khoảng cách nhỏ giữa độ chính xác huấn luyện và kiểm tra.
  - **Phù hợp với hình ảnh**: Các lớp tích chập và gộp cực đại lý tưởng cho CIFAR-10.
- **Nhược điểm**:
  - **Phức tạp tính toán**: 3.33 triệu MAC/mẫu, đòi hỏi phần cứng mạnh (GPU) để huấn luyện nhanh.
  - **Phụ thuộc siêu tham số**: Hiệu suất nhạy cảm với tỷ lệ học, số epoch, và tăng cường dữ liệu.
  - **Quá khớp nhẹ**: Vẫn có thể xảy ra nếu không dùng batch normalization hoặc dropout.
  - **Khó mở rộng**: Kiến trúc hiện tại có thể không đủ sâu để đạt hiệu suất cao hơn (ví dụ: >80%).

Nhận xét:

- MLP phù hợp với các bài toán đơn giản, không yêu cầu cấu trúc không gian, nhưng kém hiệu quả trên CIFAR-10.

- CNN là lựa chọn vượt trội cho dữ liệu hình ảnh, nhưng đòi hỏi cấu hình siêu tham số cẩn thận và tài nguyên tính toán lớn hơn.

4.3.2. Tác động của các siêu tham số

MLP:

- **Tỷ lệ học (eta = 0.001)**:
  - Giá trị thấp giúp ổn định gradient nhưng làm hội tụ chậm. Với 10 epoch, mô hình chưa đạt tối ưu.
  - Tăng eta (ví dụ: 0.01) có thể cải thiện tốc độ, nhưng cần bộ lập lịch (StepLR) để tránh dao động.
- **Momentum (mu = 0.9)**:
  - Giúp tăng tốc SGD bằng cách tích lũy gradient trước đó, phù hợp với MLP nhưng không đủ để bù đắp tỷ lệ học thấp.
- **Batch size (256)**:
  - Cân bằng giữa ổn định gradient và tốc độ huấn luyện. Giảm batch size (ví dụ: 64) có thể tăng tổng quát hóa nhưng chậm hơn.
- **Số epoch (10)**:
  - Quá ít để MLP hội tụ hoàn toàn trên CIFAR-10. Tăng lên 20-50 epoch có thể cải thiện độ chính xác ~5-10%.
- **Thiếu chính quy hóa**:
  - Không có dropout hoặc L2 dẫn đến quá khớp. Thêm dropout(0.5) hoặc weight_decay=0.0001 có thể giảm mất mát kiểm tra.

CNN:

- **Tỷ lệ học (eta = 0.001)**:
  - Ổn định nhưng chậm, đặc biệt với kiến trúc sâu hơn MLP. Tăng eta hoặc dùng Adam (lr=0.001) có thể cải thiện hội tụ.
- **Momentum (mu = 0.9)**:
  - Hiệu quả trong việc tăng tốc gradient, nhưng Adam có thể tốt hơn cho CNN do thích nghi tỷ lệ học.

- **Batch size (256)**:
    - Phù hợp với CIFAR-10, nhưng thử batch size nhỏ hơn (128) có thể cải thiện tổng quát hóa, dù tăng thời gian huấn luyện.
- **Số epoch (10)**:
    - Đủ để đạt ~60-70%, nhưng tăng lên 20-30 epoch với bộ lập lịch tỷ lệ học có thể đẩy độ chính xác lên ~75-80%.
- **Thiếu tăng cường dữ liệu**:
    - Không có RandomHorizontalFlip hoặc RandomCrop làm giảm khả năng tổng quát hóa. Thêm tăng cường dữ liệu có thể tăng độ chính xác ~5-10%.

Nhận xét:

- Cả hai mô hình bị hạn chế bởi tỷ lệ học thấp và số epoch ít. CNN nhạy cảm hơn với tăng cường dữ liệu, trong khi MLP cần chính quy hóa mạnh hơn.
- Siêu tham số hiện tại (eta=0.001, mu=0.9, batch size=256, 10 epoch) phù hợp cho thử nghiệm ban đầu, nhưng không tối ưu cho hiệu suất cao.

4.3.3. Đề xuất cải tiến

MLP:

1. **Thêm chính quy hóa**:
    - Thêm dropout(0.5) sau lớp ReLU:

```
self.network = nn.Sequential(
    nn.Linear(32 * 32 * 3, 512),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(512, 10)
)
```

- Thêm L2 regularization: optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=0.0001).
- Lợi ích: Giảm quá khớp, cải thiện độ chính xác kiểm tra ~5%.

Tăng số epoch:

- Tăng lên 20-50 epoch với bộ lập lịch tỷ lệ học:

```
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
for epoch in range(num_epochs):
    # ... (huấn luyện)
    scheduler.step()
```

- Lợi ích: Đạt hội tụ tốt hơn, tăng độ chính xác ~5-10%.

Dùng Adam:

- Thay SGD bằng Adam: optimizer = optim.Adam(model.parameters(), lr=0.001).
- Lợi ích: Tăng tốc hội tụ, đặc biệt với tỷ lệ học thấp.

Tăng cường dữ liệu:

- Áp dụng RandomHorizontalFlip và RandomCrop:

```
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
])
```

    - Lợi ích: Tăng tổng quát hóa, giảm quá khớp.

CNN:

1. **Thêm batch normalization**:
    - Thêm BatchNorm2d sau mỗi lớp tích chập:

```
self.conv1 = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3, padding=1),
    nn.BatchNorm2d(16),
    nn.ReLU()
)
```

Lợi ích: Ổn định huấn luyện, tăng độ chính xác ~5-10%.

2. Thêm dropout:

- Thêm dropout(0.5) trước lớp fc1:

```
self.fc1 = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(64 * 4 * 4, 512)
)
```

1.
    - Lợi ích: Giảm quá khớp, cải thiện tổng quát hóa.
2. **Tăng số epoch và bộ lập lịch**:
    - Tăng lên 20-30 epoch với StepLR (như trên).
    - Lợi ích: Đẩy độ chính xác kiểm tra lên ~75-80%.
3. **Tăng cường dữ liệu**:
    - Áp dụng RandomHorizontalFlip và RandomCrop (như trên).
    - Lợi ích: Tăng độ chính xác ~5-10%, đặc biệt trên CIFAR-10.
4. **Dùng Adam**:
    - optimizer = optim.Adam(model.parameters(), lr=0.001).

- Lợi ích: Hội tụ nhanh hơn, ổn định hơn SGD.

Nhận xét:

- MLP cần chính quy hóa mạnh (dropout, L2) và số epoch lớn để cải thiện hiệu suất, nhưng khó đạt độ chính xác cao như CNN.
- CNN có tiềm năng cải thiện lớn hơn với batch normalization, tăng cường dữ liệu, và kiến trúc sâu hơn (ví dụ: thêm lớp tích chập).

V. Kết luận

5.1. Tóm tắt kết quả

- **MLP**:
  - **Hiệu suất**: Độ chính xác kiểm tra ~35-45%, mất mát kiểm tra ~2.0-2.1 sau 10 epoch.
  - **Đặc điểm**: Kiến trúc đơn giản với hai lớp tuyến tính, nhưng không tận dụng cấu trúc không gian, dẫn đến hiệu suất thấp và quá khớp nghiêm trọng.
  - **Tính toán**: ~1.58 triệu tham số, ~630 tỷ MAC cho 10 epoch, ~6MB bộ nhớ tham số.
  - **Phù hợp**: Dữ liệu không có cấu trúc không gian, nhưng không hiệu quả trên CIFAR-10.
- **CNN**:
  - **Hiệu suất**: Độ chính xác kiểm tra ~60-70%, mất mát kiểm tra ~1.6-1.8, vượt trội nhờ trích xuất đặc trưng không gian.
  - **Đặc điểm**: Ba lớp tích chập và gộp cực đại, tổng quát hóa tốt hơn, nhưng vẫn có quá khớp nhẹ.
  - **Tính toán**: ~553 nghìn tham số, ~1.33 nghìn tỷ MAC, ~2.2MB bộ nhớ tham số.
  - **Phù hợp**: Dữ liệu hình ảnh như CIFAR-10, với tiềm năng cải thiện thêm.
- **Lỗi đã sửa**: Trong mã MLP, dòng pil_img.shuffle((128, 128), Image.LANCZOS) được sửa thành pil_img.resize((128, 128), Image.LANCZOS) để thay đổi kích thước ảnh đúng cách.

5.2. Bài học kinh nghiệm

1. **Tầm quan trọng của cấu trúc không gian**:
    - MLP thất bại trong việc khai thác thông tin không gian của CIFAR-10, dẫn đến hiệu suất thấp. CNN, với các lớp tích chập, là lựa chọn phù hợp hơn cho dữ liệu hình ảnh.
    - Bài học: Chọn kiến trúc phù hợp với đặc điểm dữ liệu (tích chập cho hình ảnh, tuyến tính cho dữ liệu bảng).
2. **Quá khớp và chính quy hóa**:
    - Cả MLP và CNN đều gặp quá khớp, nhưng MLP nghiêm trọng hơn do số tham số lớn và thiếu chính quy hóa.
    - Bài học: Áp dụng dropout, batch normalization, L2 regularization, và tăng cường dữ liệu để cải thiện tổng quát hóa.
3. **Tác động của siêu tham số**:
    - Tỷ lệ học thấp (0.001) và số epoch ít (10) hạn chế hiệu suất. CNN nhạy cảm hơn với tăng cường dữ liệu, trong khi MLP cần chính quy hóa mạnh.
    - Bài học: Tối ưu siêu tham số (thử nghiệm tỷ lệ học, bộ tối ưu, số epoch) và sử dụng bộ lập lịch tỷ lệ học để đạt hội tụ tốt.
4. **Hiệu quả tính toán**:
    - MLP tốn ít tài nguyên hơn mỗi mẫu (1.58 triệu MAC) nhưng kém hiệu quả do hiệu suất thấp. CNN tốn nhiều tài nguyên hơn (3.33 triệu MAC) nhưng đáng giá nhờ độ chính xác cao.
    - Bài học: Cân nhắc giữa chi phí tính toán và hiệu suất khi chọn mô hình, ưu tiên CNN cho hình ảnh nếu có phần cứng phù hợp.
5. **Kiểm tra mã nguồn**:
    - Lỗi shuffle trong MLP nhấn mạnh tầm quan trọng của việc kiểm tra cú pháp và logic mã.
    - Bài học: Kiểm tra kỹ lưỡng các phương thức thư viện (như PIL) và chạy thử nghiệm để phát hiện lỗi sớm.
6. **Cải tiến liên tục**:

- Cả hai mô hình có thể cải thiện bằng cách thêm lớp, chính quy hóa, tăng cường dữ liệu, và tối ưu siêu tham số.
- Bài học: Huấn luyện mạng nơ-ron là quá trình lặp, cần thử nghiệm và tinh chỉnh để đạt hiệu suất tối ưu.