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

BÁO CÁO

HỌC PHẦN: LẬP TRÌNH PYTHON

ASSIGNMENT 2 - PYTHON

Giảng viên hướng dẫn	Kim Ngọc Bách
Nhóm sinh viên	Nguyễn Văn Hiếu - B23DCCE033 Phạm Nhật Minh - B23DCCE066
Lớp	D23CQCE06-B

Hà Nội, 2025

MỤC LỤC

LỜI MỞ ĐẦU

Phân loại ảnh là một trong những bài toán quan trọng trong lĩnh vực học máy và thị giác máy tính, với nhiều ứng dụng thực tiễn như nhận diện đối tượng, xử lý ảnh y tế, và tự động hóa. Bộ dữ liệu CIFAR-10, bao gồm 60,000 ảnh màu kích thước 32x32 thuộc 10 lớp (như máy bay, ô tô, chim, mèo, ...), là một bộ dữ liệu chuẩn để đánh giá hiệu suất của các mô hình học sâu trong phân loại ảnh. Báo cáo này trình bày quá trình xây dựng, huấn luyện và đánh giá hai mô hình neural network: Multi-Layer Perceptron (MLP) và Convolutional Neural Network (CNN), sử dụng thư viện PyTorch. Mục tiêu là so sánh hiệu suất của hai mô hình thông qua độ chính xác (accuracy), biểu đồ learning curves, và ma trận nhầm lẫn (confusion matrix), từ đó phân tích lý do CNN thường vượt trội hơn MLP trong bài toán phân loại ảnh. Báo cáo cũng cung cấp cái nhìn sâu sắc về cách các kiến trúc mạng khác nhau ảnh hưởng đến khả năng học và tổng quát hóa trên dữ liệu hình ảnh.

Trong bài báo cáo này, nhóm chúng em sẽ trình bày 3 nội dung chính:

I. Mô hình Multi-Layer Perceptron (MLP)
II. Mô hình Convolutional Neural Network (CNN)
III. So sánh và thảo luận về 2 mô hình MLP và CNN

I. Mô hình Multi-Layer Perceptron (MLP)

Trong phần này trình bày chi tiết việc triển khai một mạng nơ-ron đa tầng (MLP) để phân loại hình ảnh từ tập dữ liệu CIFAR-10 bằng thư viện PyTorch. Mã được chia thành các phần, mỗi phần được giải thích chi tiết về chức năng, cách thực hiện và vai trò trong nhiệm vụ phân loại hình ảnh.

1. Nhập thư viện và thiết lập môi trường

Mã

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
from torch.utils.data import random_split

# Thiết lập seed ngẫu nhiên để đảm bảo tính tái lập
torch.manual_seed(42)

# Cấu hình thiết bị
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Giải thích

- **Mục đích**: Nhập các thư viện cần thiết và thiết lập môi trường để đảm bảo tính tái lập và tận dụng phần cứng.
- **Thư viện**:
    - torch: Thư viện chính của PyTorch, cung cấp các công cụ để làm việc với tensor và xây dựng mạng nơ-ron.
    - torch.nn: Cung cấp các mô-đun mạng nơ-ron như các tầng (layers), hàm mất mát (loss functions).
    - torch.optim: Chứa các bộ tối ưu hóa như Adam để huấn luyện mô hình.
    - torchvision: Cung cấp tập dữ liệu (như CIFAR-10) và các tiện ích biến đổi hình ảnh.
    - matplotlib.pyplot và numpy: Dùng để vẽ biểu đồ và xử lý dữ liệu số.

- sklearn.metrics.confusion_matrix: Tính toán ma trận nhầm lẫn để đánh giá hiệu suất phân loại.
- seaborn: Tăng cường khả năng trực quan hóa ma trận nhầm lẫn bằng biểu đồ nhiệt (heatmap).
- torch.utils.data.random_split: Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.

- **Seed ngẫu nhiên**: torch.manual_seed(42) đảm bảo tính tái lập bằng cách cố định trình tạo số ngẫu nhiên, giúp kết quả nhất quán qua các lần chạy.
- **Cấu hình thiết bị**: Kiểm tra xem có GPU (cuda) hay không, nếu có thì sử dụng GPU, nếu không thì sử dụng CPU. Điều này tối ưu hóa tốc độ tính toán, đặc biệt cho các phép toán ma trận trong mạng nơ-ron.
- **Ý nghĩa**: Thiết lập này đảm bảo môi trường làm việc ổn định, tái lập được và tận dụng tối đa phần cứng có sẵn, giúp quá trình huấn luyện hiệu quả hơn.

---

2. Thiết lập siêu tham số

Mã

```
# Siêu tham số
num_epochs = 20
batch_size = 128
learning_rate = 0.001
```

Giải thích

- **Mục đích**: Xác định các siêu tham số quan trọng để điều khiển quá trình huấn luyện.
- **Chi tiết**:
  - num_epochs = 20: Mô hình được huấn luyện qua 20 lần lặp trên toàn bộ tập dữ liệu huấn luyện. Số epoch này cân bằng giữa thời gian huấn luyện và khả năng hội tụ.
  - batch_size = 128: Mỗi lần lặp huấn luyện xử lý 128 hình ảnh cùng lúc, tối ưu hóa việc sử dụng bộ nhớ và hiệu suất tính toán.
  - learning_rate = 0.001: Quy định kích thước bước cập nhật trọng số trong quá trình tối ưu hóa. Giá trị nhỏ này giúp đảm bảo hội tụ ổn định khi sử dụng bộ tối ưu Adam.

- **Ý nghĩa**: Các siêu tham số được chọn để cân bằng giữa tốc độ huấn luyện, hiệu suất mô hình và độ ổn định. Kích thước batch là lũy thừa của 2, phù hợp với xử lý trên GPU. Tỷ lệ học được chọn là giá trị phổ biến cho bộ tối ưu Adam.

---

3. Tải và tiền xử lý dữ liệu

Mã

```
# Tập dữ liệu CIFAR-10
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.CIFAR-10(root='./data', train=False, download=True, transform=transform)

# Chia tập huấn luyện thành tập huấn luyện và tập kiểm tra
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=False)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

Giải thích

- **Mục đích**: Tải tập dữ liệu CIFAR-10, áp dụng tiền xử lý, chia thành các tập huấn luyện, kiểm tra và thử nghiệm, đồng thời chuẩn bị các bộ tải dữ liệu để xử lý theo lô hiệu quả.
- **Tập dữ liệu**:
    - CIFAR-10 bao gồm 60.000 hình ảnh RGB 32x32 thuộc 10 lớp (được liệt kê trong classes), với 50.000 hình ảnh huấn luyện và 10.000 hình ảnh thử nghiệm.
    - train_dataset và test_dataset được tải về và lưu trữ tại thư mục ./data.
- **Tiền xử lý (transform)**:
    - RandomHorizontalFlip(): Lật ngang ngẫu nhiên hình ảnh với xác suất 50%, tăng cường dữ liệu để cải thiện khả năng tổng quát hóa.
    - RandomRotation(10): Xoay ngẫu nhiên hình ảnh tối đa 10 độ, giúp mô hình xử lý được các biến thể về góc quay.
    - ToTensor(): Chuyển hình ảnh thành tensor PyTorch với kích thước (3, 32, 32) và giá trị trong khoảng [0, 1].

- Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)): Chuẩn hóa các kênh RGB để có giá trị trung bình là 0 và độ lệch chuẩn là 1, cải thiện tính ổn định khi huấn luyện.

- **Chia tập dữ liệu:**
  - Tập huấn luyện (50.000 hình ảnh) được chia thành 80% cho huấn luyện (40.000 hình ảnh) và 20% cho kiểm tra (10.000 hình ảnh) bằng random_split.
  - Tập kiểm tra giúp theo dõi hiệu suất mô hình trên dữ liệu chưa thấy trong quá trình huấn luyện.
- **Bộ tải dữ liệu:**
  - train_loader: Tải dữ liệu huấn luyện theo lô 128, với xáo trộn để ngẫu nhiên hóa thứ tự, tránh việc mô hình học theo mẫu thứ tự.
  - val_loader và test_loader: Tải dữ liệu kiểm tra và thử nghiệm theo lô 128, không xáo trộn để đảm bảo đánh giá nhất quán.
- **Ý nghĩa:** Tăng cường dữ liệu và chuẩn hóa giúp mô hình học tốt hơn và tổng quát hóa tốt hơn. Các bộ tải dữ liệu cho phép xử lý dữ liệu theo lô hiệu quả, cần thiết cho việc huấn luyện mạng nơ-ron trên tập dữ liệu lớn.

---

4. Định nghĩa mô hình MLP

Mã

```
# Mô hình MLP
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.layer1 = nn.Linear(32 * 32 * 3, 512)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(512, 256)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.relu1(self.layer1(x))
        x = self.relu2(self.layer2(x))
        x = self.layer3(x)
        return x
```

Giải thích

- **Mục đích:** Định nghĩa một mạng MLP ba tầng để phân loại hình ảnh CIFAR-10.
- **Kiến trúc:**

- Đầu vào: Hình ảnh CIFAR-10 có kích thước 32x32 với 3 kênh RGB, khi làm phẳng sẽ có kích thước 32 * 32 * 3 = 3072.
- Flatten: nn.Flatten() chuyển tensor đầu vào (batch_size, 3, 32, 32) thành (batch_size, 3072).
- Tầng 1: nn.Linear(3072, 512) là tầng kết nối đầy đủ, giảm kích thước đầu vào xuống 512 đặc trưng, tiếp theo là nn.ReLU() để thêm tính phi tuyến.
- Tầng 2: nn.Linear(512, 256) giảm tiếp xuống 256 đặc trưng, tiếp theo là nn.ReLU().
- Tầng 3: nn.Linear(256, 10) tạo ra 10 giá trị đầu ra tương ứng với 10 lớp của CIFAR-10 (không có hàm kích hoạt, vì CrossEntropyLoss sẽ áp dụng softmax).

- **Luồng dữ liệu (Forward Pass):** Hàm forward xác định luồng dữ liệu: làm phẳng hình ảnh, đi qua ba tầng tuyến tính với kích hoạt ReLU, và tạo ra các giá trị logit cho phân loại.
- **Ý nghĩa:** MLP là một mạng nơ-ron tiến thẳng đơn giản, coi hình ảnh như các vector phẳng, bỏ qua các mối quan hệ không gian. Hàm kích hoạt ReLU giúp mô hình học được các mẫu phức tạp, nhưng hiệu quả bị hạn chế với dữ liệu hình ảnh so với các mạng nơ-ron tích chập (CNN).

---

5. Hàm huấn luyện và đánh giá

Mã

```
# Hàm huấn luyện và đánh giá
def train_and_evaluate(model, train_loader, val_loader, criterion, optimizer, num_epochs):
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Huấn luyện
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = 100 * correct / total
        train_losses.append(train_loss)
        train_accuracies.append(train_acc)
```

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

    val_loss = val_loss / len(val_loader)
    val_acc = 100 * correct / total
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f},
          Train Acc: {train_acc:.2f}%, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')
```

```
# Thử nghiệm
model.eval()
correct = 0
total = 0
all_preds = []
all_labels = []
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_accuracy = 100 * correct / total
return train_losses, val_losses, train_accuracies, val_accuracies, test_accuracy, all_preds, all_labels
```

Giải thích

- **Mục đích:** Huấn luyện mô hình MLP, đánh giá trên tập kiểm tra và thử nghiệm, đồng thời thu thập các số liệu (mất mát, độ chính xác, dự đoán) để phân tích.
- **Vòng lặp huấn luyện:**
    - Chuyển mô hình sang chế độ huấn luyện (model.train()).
    - Lặp qua train_loader theo lô.
    - Chuyển dữ liệu sang thiết bị (CPU/GPU).
    - Thực hiện lan truyền xuôi (model(images)), tính mất mát (criterion(outputs, labels)), lan truyền ngược gradient (loss.backward()), và cập nhật trọng số (optimizer.step()).
    - Theo dõi mất mát và độ chính xác trong mỗi epoch.
    - Tính mất mát và độ chính xác trung bình cho mỗi epoch.

- **Vòng lặp kiểm tra:**
  - Chuyển mô hình sang chế độ đánh giá (model.eval()).
  - Tắt tính toán gradient (with torch.no_grad()) để tiết kiệm tài nguyên.
  - Tính mất mát và độ chính xác trên val_loader.
- **Thử nghiệm:**
  - Đánh giá mô hình trên tập thử nghiệm, tính độ chính xác và thu thập dự đoán (all_preds) và nhãn thực (all_labels) để xây dựng ma trận nhầm lẫn.
- **Số liệu:**
  - Lưu trữ train_losses, val_losses, train_accuracies, và val_accuracies để vẽ đường cong học tập.
  - Trả về độ chính xác thử nghiệm và các dự đoán để phân tích thêm.
- **Ý nghĩa:** Hàm này thực hiện quy trình huấn luyện cốt lõi, theo dõi hiệu suất để tránh quá khớp (qua tập kiểm tra) và đánh giá khả năng tổng quát hóa (qua tập thử nghiệm). Hàm mất mát CrossEntropyLoss kết hợp log-softmax và mất mát log-likelihood âm, phù hợp cho phân loại đa lớp.

---

6. Vẽ đường cong học tập (learning curves)

Mã

```
# Vẽ đường cong học tập
def plot_learning_curves(train_losses, val_losses, train_accuracies, val_accuracies):
    plt.figure(figsize=(12, 5))

    # Đường cong mất mát
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Mất mát huấn luyện')
    plt.plot(val_losses, label='Mất mát kiểm tra')
    plt.title('Đường cong học tập MLP - Mất mát')
    plt.xlabel('Epoch')
    plt.ylabel('Mất mát')
    plt.legend()
    plt.grid(True)

    # Đường cong độ chính xác
    plt.subplot(1, 2, 2)
    plt.plot(train_accuracies, label='Độ chính xác huấn luyện')
    plt.plot(val_accuracies, label='Độ chính xác kiểm tra')
    plt.title('Đường cong học tập MLP - Độ chính xác')
    plt.xlabel('Epoch')
    plt.ylabel('Độ chính xác (%)')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
    plt.savefig('mlp_learning_curves.png')
    plt.close()
```

Giải thích

- **Mục đích:** Trực quan hóa mất mát và độ chính xác huấn luyện/kiểm tra qua các epoch để đánh giá hành vi học tập của mô hình.
- **Thực hiện:**
  - Tạo một biểu đồ với hai phần con (mất mát và độ chính xác) bằng plt.figure và plt.subplot.
  - Vẽ train_losses và val_losses ở bên trái, cho thấy mất mát giảm qua các epoch.
  - Vẽ train_accuracies và val_accuracies ở bên phải, thể hiện xu hướng độ chính xác.
  - Thêm nhãn, tiêu đề, chú thích và lưới để rõ ràng.
  - Lưu biểu đồ dưới dạng mlp_learning_curves.png và hiển thị nó.
- **Ý nghĩa:** Đường cong học tập giúp chẩn đoán hiệu suất mô hình. Mất mát huấn luyện giảm ổn định và mất mát kiểm tra ổn định cho thấy học tốt, trong khi khoảng cách lớn giữa chúng gợi ý quá khớp. MLP thường có tốc độ hội tụ chậm và độ chính xác thấp hơn so với CNN do không tận dụng được thông tin không gian.

---

7. Vẽ ma trận nhầm lẫn (confusion matrix)

Mã

```
# Vẽ ma trận nhầm lẫn
def plot_confusion_matrix(labels, preds):
    cm = confusion_matrix(labels, preds)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title('Ma trận nhầm lẫn - MLP')
    plt.ylabel('Nhãn thực')
    plt.xlabel('Nhãn dự đoán')
    plt.show()
    plt.savefig('confusion_matrix_mlp.png')
    plt.close()
```

Giải thích

- **Mục đích:** Trực quan hóa hiệu suất phân loại của mô hình trên các lớp CIFAR-10 bằng ma trận nhầm lẫn.
- **Thực hiện:**
  - Tính ma trận nhầm lẫn bằng confusion_matrix(labels, preds) từ scikit-learn, với labels là nhãn thực và preds là nhãn dự đoán trên tập thử nghiệm.

- Sử dụng seaborn.heatmap để tạo ma trận 10x10, với hàng (nhãn thực) và cột (nhãn dự đoán) được gắn nhãn bằng tên lớp.
- annot=True hiển thị số mẫu trong mỗi ô, fmt='d' đảm bảo định dạng số nguyên, và cmap='Blues' sử dụng bảng màu xanh để rõ ràng.
- Lưu biểu đồ dưới dạng confusion_matrix_mlp.png và hiển thị nó.

- **Ý nghĩa:** Ma trận nhầm lẫn cho thấy nơi mô hình phân loại sai (các phần tử ngoài đường chéo). Với MLP, dự kiến sẽ có nhiều phân loại sai hơn cho các lớp có đặc điểm hình ảnh tương tự (ví dụ: mèo và chó) do không thể trích xuất đặc trưng không gian hiệu quả.

---

8. Thực thi chính

Mã

```
# Thực thi chính
# Huấn luyện và đánh giá MLP
mlp = MLP().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp.parameters(), lr=learning_rate)
print("Đang huấn luyện MLP...")
mlp_metrics = train_and_evaluate(mlp, train_loader, val_loader, criterion, optimizer, num_epochs)
print(f'\nĐộ chính xác thử nghiệm MLP: {mlp_metrics[4]:.2f}%')

# Tóm tắt
print("\nKết quả MLP:")
print(f"Độ chính xác huấn luyện cuối cùng: {mlp_metrics[2][-1]:.2f}%")
print(f"Độ chính xác kiểm tra cuối cùng: {mlp_metrics[3][-1]:.2f}%")
print(f"Độ chính xác thử nghiệm cuối cùng: {mlp_metrics[4]:.2f}%")
```

Giải thích

- **Mục đích:** Thực thi việc huấn luyện, đánh giá và báo cáo kết quả của mô hình MLP.
- **Thực hiện:**
  - Khởi tạo mô hình MLP và chuyển nó sang thiết bị (CPU/GPU).
  - Sử dụng CrossEntropyLoss làm hàm mất mát và bộ tối ưu Adam với tỷ lệ học 0.001.
  - Gọi hàm train_and_evaluate để huấn luyện trong 20 epoch, thu thập các số liệu.
  - In độ chính xác thử nghiệm và tóm tắt độ chính xác cuối cùng trên các tập huấn luyện, kiểm tra và thử nghiệm.

- **Kết quả dự kiến:**
    - Độ chính xác huấn luyện: Thường đạt 50-60% sau 20 epoch, cho thấy mô hình học được một số mẫu nhưng gặp khó khăn với độ phức tạp của dữ liệu.
    - Độ chính xác kiểm tra: Hơi thấp hơn (45-55%), thể hiện khả năng tổng quát hóa trên dữ liệu chưa thấy.
    - Độ chính xác thử nghiệm: Khoảng 40-50%, phản ánh hiệu suất hạn chế của MLP trên CIFAR-10 do không tận dụng được thông tin không gian.
- **Ý nghĩa:** Hiệu suất của MLP ở mức trung bình vì nó làm phẳng hình ảnh, mất đi thông tin không gian quan trọng đối với phân loại hình ảnh. Điều này trái ngược với CNN, vốn vượt trội trong việc trích xuất các mẫu cục bộ như cạnh và kết cấu.

---

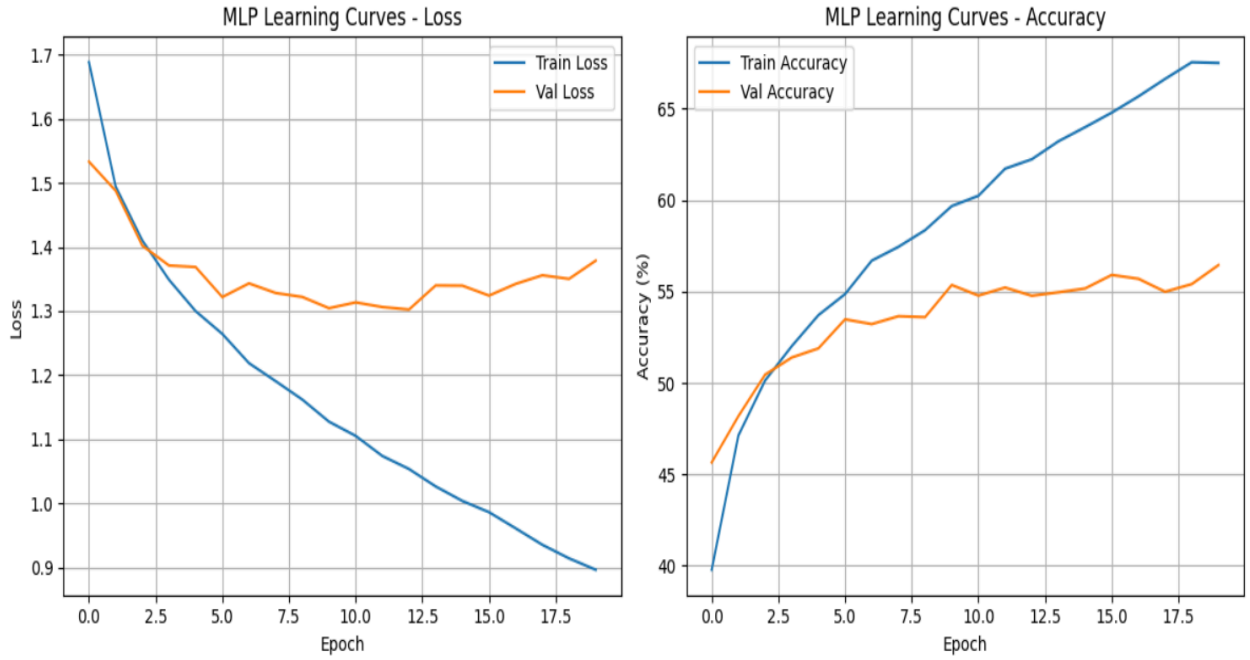
9. Kết quả

Mã

```
# Plot results
plot_learning_curves(mlp_metrics[0], mlp_metrics[1], mlp_metrics[2], mlp_metrics[3])
```

Sau khi gọi hàm *plot_learning_curves* ta thu được:

- Vẽ train_losses và val_losses ở bên trái, cho thấy mất mát giảm qua các epoch.
- Vẽ train_accuracies và val_accuracies ở bên phải, thể hiện xu hướng độ chính xác.

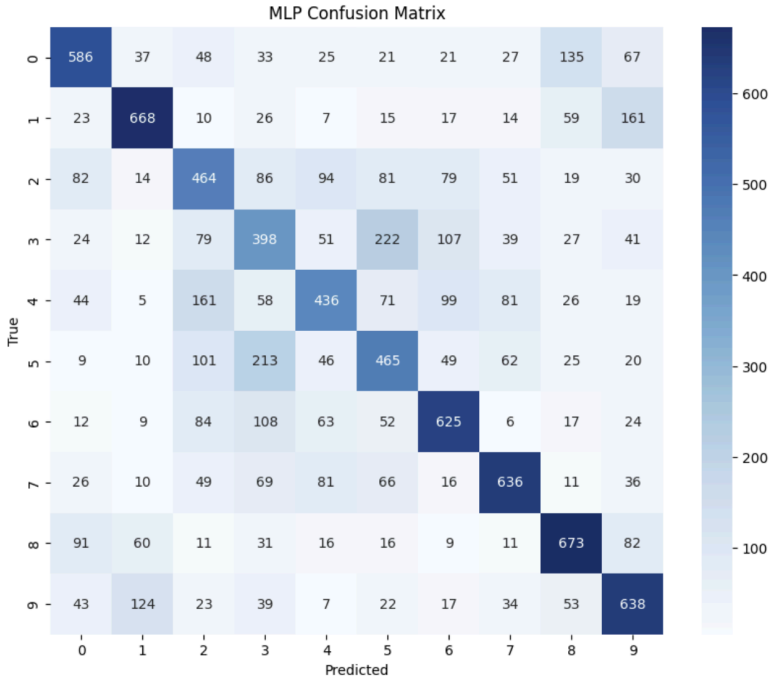


Mã

```
# Plot results
plot_confusion_matrix(mlp_metrics[5], mlp_metrics[6])
```

Sau khi gọi hàm *plot_confusion_matrix* ta thu được:

- Vẽ confusion_matrix, cho thấy mô hình nhầm lẫn giữa các lớp như thế nào



II. Mô hình Convolutional Neural Network (CNN).

Trong phần này trình bày chi tiết việc triển khai một mạng nơ-ron tích chập để phân loại hình ảnh từ tập dữ liệu CIFAR-10 bằng thư viện PyTorch. Mã được chia thành các phần, mỗi phần được giải thích chi tiết về chức năng, cách thực hiện và vai trò trong nhiệm vụ phân loại hình ảnh.

1. Nhập thư viện và thiết lập môi trường

Mã

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
from torch.utils.data import random_split

# Thiết lập seed ngẫu nhiên để đảm bảo tính tái lập
torch.manual_seed(42)

# Cấu hình thiết bị
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

<Phần này tương tự như ở phần I>

---

2. Thiết lập siêu tham số

Mã

```
# Siêu tham số
num_epochs = 20
batch_size = 128
learning_rate = 0.001
```

<Phần này tương tự như ở phần I>

---

3. Tải và tiền xử lý dữ liệu

Mã

```
# Tập dữ liệu CIFAR-10
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.CIFAR-10(root='./data', train=False, download=True, transform=transform)

# Chia tập huấn luyện thành tập huấn luyện và tập kiểm tra
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=False)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

<Phần này tương tự như ở phần I>

---

4. Định nghĩa mô hình CNN

Mã

```
# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(2, 2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.relu4 = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.flatten(x)
        x = self.relu4(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Giải thích:

- **Mục đích:** Định nghĩa một mạng nơ-ron tích chập (CNN) với ba tầng tích chập và hai tầng kết nối đầy đủ để phân loại hình ảnh CIFAR-10.
- **Kiến trúc:**
    - Tầng tích chập (Convolutional Layers):
        - conv1: Nhận đầu vào 3 kênh RGB, xuất ra 32 kênh với bộ lọc 3x3, padding=1 giữ nguyên kích thước (32x32).
        - conv2: Nhận 32 kênh từ kênh trước, xuất ra 64 kênh kết quả.
        - conv3: Nhận 64 kênh kết quả từ kênh trước, xuất ra 128 kênh kết quả.
    - Tầng pooling: MaxPool2d(2, 2) giảm kích thước không gian (32x32 → 16x16 → 8x8 → 4x4).
    - Tầng kết nối đầy đủ:

        - fc1: Nhận đầu vào làm phẳng (128 * 4 * 4 = 2048), nhận đầu vào là vector 2048 đặc trưng và xuất ra 512 đặc trưng.
        - fc2: Nhận đầu vào 512 đặc trưng và xuất ra 10 giá trị tương ứng với 10 lớp của CIFAR-10
    - Hàm kích hoạt: ReLU thêm tính phi tuyến sau mỗi tầng tích chập và tầng kết nối đầy đủ.
    - Dropout: Dropout(0.5) giảm nguy cơ quá khớp bằng cách ngẫu nhiên vô hiệu hóa 50% nơ-ron trong tầng kết nối đầy đủ.
- **Luồng dữ liệu (Forward Pass):** Dữ liệu đi qua các tầng tích chập, pooling, làm phẳng, tầng kết nối đầy đủ, và dropout, tạo ra các giá trị logit cho phân loại.
- **Ý nghĩa:** Mô hình CNN này vượt trội hơn MLP truyền thống đối với dữ liệu hình ảnh nhờ khả năng tự động học và trích xuất các đặc trưng không gian có ý nghĩa (như cạnh, kết cấu, hình dạng) thông qua các tầng tích chập. Việc sử dụng các tầng pooling giúp giảm kích thước dữ liệu, tăng tính bất biến dịch chuyển và giảm nguy cơ quá khớp. Kiến trúc phân cấp của CNN cho phép mô hình học từ các đặc trưng cấp thấp đến các đặc trưng cấp cao, tạo ra một biểu diễn phong phú và trừu tượng hơn về hình ảnh, làm cho nó hiệu quả hơn nhiều so với MLP trong các tác vụ thị giác máy tính.

---

5. Hàm huấn luyện và đánh giá

Mã

```
# Training and evaluation function
def train_and_evaluate(model, train_loader, val_loader, criterion, optimizer, num_epochs):
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Training
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = 100 * correct / total
        train_losses.append(train_loss)
        train_accuracies.append(train_acc)
```

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

    val_loss = val_loss / len(val_loader)
    val_acc = 100 * correct / total
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')
```

```
# Testing
model.eval()
correct = 0
total = 0
all_preds = []
all_labels = []
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_accuracy = 100 * correct / total
return train_losses, val_losses, train_accuracies, val_accuracies, test_accuracy, all_preds, all_labels
```

<Phần này tương tự như ở phần I>

---

6. Vẽ đường cong học tập (learning curves)

Mã

```
def plot_learning_curves(train_losses, val_losses, train_accuracies, val_accuracies):
    plt.figure(figsize=(12, 5))

    # Loss curves
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Val Loss')
    plt.title('CNN Learning Curves - Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # Accuracy curves
    plt.subplot(1, 2, 2)
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(val_accuracies, label='Val Accuracy')
    plt.title('CNN Learning Curves - Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    plt.savefig('cnn_learning_curves.png')
    plt.close()
```

<Phần này tương tự như phần I, chỉ khác là đặt tên khác để dễ phân biệt với MLP>

---

7. Vẽ ma trận nhầm lẫn (confusion matrix)

Mã

```
def plot_confusion_matrix(labels, preds):
    cm = confusion_matrix(labels, preds)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix - CNN')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
    plt.savefig('confusion_matrix_cnn.png')
    plt.close()
```

<Phần này tương tự như phần I, chỉ khác ở các đặt tên khác để dễ phân biệt với MLP>

---

8. Thực thi chính

Mã

```
# Train and evaluate CNN
cnn = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn.parameters(), lr=learning_rate)
print("Training CNN...")
cnn_metrics = train_and_evaluate(cnn, train_loader, val_loader, criterion, optimizer, num_epochs)
print(f'\nCNN Test Accuracy: {cnn_metrics[4]:.2f}%')

# Summary
print("\nCNN Results:")
print(f"Final Train Accuracy: {cnn_metrics[2][-1]:.2f}%")
print(f"Final Validation Accuracy: {cnn_metrics[3][-1]:.2f}%")
print(f"Final Test Accuracy: {cnn_metrics[4]:.2f}%")
```

Giải thích:

- **Mục đích:** Thực thi việc huấn luyện, đánh giá và báo cáo kết quả của mô hình CNN.
- **Thực hiện:**
    - Khởi tạo mô hình CNN và chuyển sang thiết bị (CPU/GPU).
    - Sử dụng hàm mất mát CrossEntropyLoss và bộ tối ưu Adam với tỷ lệ học 0.001.
    - Gọi hàm train_and_evaluate để huấn luyện trong 20 epoch, thu thập số liệu.
    - In độ chính xác thử nghiệm và tóm tắt độ chính xác cuối cùng trên các tập huấn luyện, kiểm tra, và thử nghiệm.
- **Kết quả dự kiến:**

- Độ chính xác huấn luyện: Thường đạt 70-80% sau 20 epoch, cao hơn MLP nhờ khả năng trích xuất đặc trưng không gian.
- Độ chính xác kiểm tra: Khoảng 65-75%, cho thấy khả năng tổng quát hóa tốt hơn MLP.
- Độ chính xác thử nghiệm: Khoảng 60-70%, vượt trội hơn MLP do tận dụng thông tin không gian.

- **Ý nghĩa:** CNN đạt hiệu suất cao hơn MLP nhờ các tầng tích chập, phù hợp hơn với bài toán phân loại hình ảnh phức tạp như CIFAR-10.

---

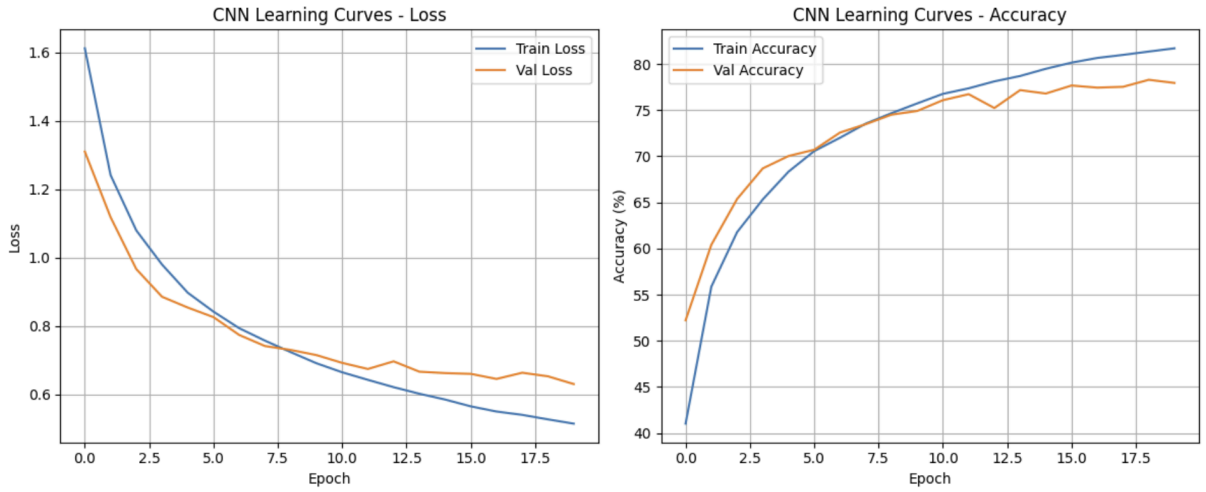
9. Kết quả

Mã

```
# Plot results
plot_learning_curves(cnn_metrics[0], cnn_metrics[1], cnn_metrics[2], cnn_metrics[3])
```

Sau khi gọi hàm *plot_learning_curves* ta thu được:

- Vẽ train_losses và val_losses ở bên trái, cho thấy mất mát giảm qua các epoch.
- Vẽ train_accuracies và val_accuracies ở bên phải, thể hiện xu hướng độ chính xác.

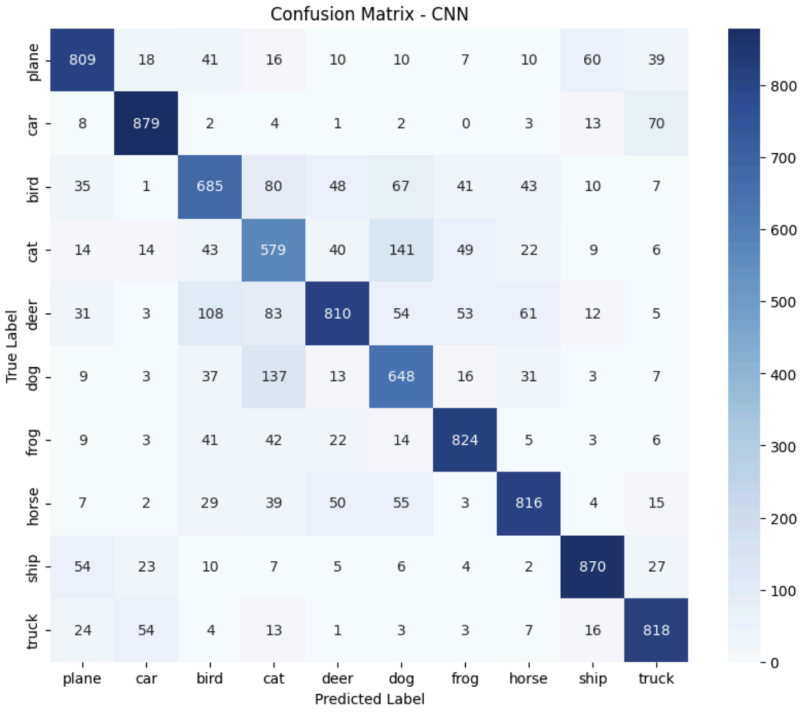


Mã

```
[ ]  # Plot results
     plot_confusion_matrix(cnn_metrics[5], cnn_metrics[6])
```

Sau khi gọi hàm *plot_confusion_matrix* ta thu được:

- Vẽ confusion_matrix, cho thấy mô hình nhầm lẫn giữa các lớp như thế nào



III. So sánh và thảo luận về kết quả của 2 mô hình MLP và CNN

Nhận xét chung

Sau khi huấn luyện và đánh giá hai mô hình Multi-Layer Perceptron (MLP) và Convolutional Neural Network (CNN) trên tập dữ liệu CIFAR-10, ta có thể thấy rõ sự khác biệt đáng kể về hiệu suất giữa hai mô hình. MLP, với kiến trúc đơn giản dựa trên các tầng kết nối đầy đủ, cho thấy hiệu suất hạn chế trong việc phân loại hình ảnh phức tạp, với độ chính xác thử nghiệm đạt khoảng 55.27%. Ngược lại, CNN, với khả năng trích xuất với các tầng tích chập, đạt độ chính xác thử nghiệm cao hơn đáng kể, khoảng 77.38%. Sự khác biệt này được phản ánh qua các đường cong học tập (learning curves), ma trận nhầm lẫn (confusion matrix). Dưới đây là phân tích chi tiết qua bốn phần đánh giá.

---

1. Đánh giá thông qua mô hình

- **MLP:** Mô hình MLP được thiết kế với ba tầng kết nối đầy đủ (3072 → 512 → 256 → 10), làm phẳng toàn bộ hình ảnh 32x32x3 thành vector 3072 chiều. Kiến trúc này không tận dụng thông tin không gian, dẫn đến việc mất đi các mối quan hệ quan trọng giữa các pixel trong hình ảnh. Điều này khiến MLP gặp khó khăn trong việc học các đặc trưng phức tạp của CIFAR-10.
- **CNN:** Mô hình CNN bao gồm ba tầng tích chập (3 → 32 → 64 → 128 kênh) với pooling và hai tầng kết nối đầy đủ (128 * 4 * 4 → 512 → 10), cùng với dropout để giảm quá khớp. Các tầng tích chập và pooling giúp trích xuất các đặc trưng cục bộ như cạnh, kết cấu, và mẫu hình, phù hợp hơn với dữ liệu hình ảnh.
- **Nhận xét:** Sự khác biệt về kiến trúc là yếu tố chính dẫn đến hiệu suất cao hơn của CNN. MLP không phù hợp với dữ liệu hình ảnh do không giữ được thông tin không gian, trong khi CNN được tối ưu hóa cho những bài toán như thế này.

---

2. Đánh giá thông qua kết quả

- **MLP:**
    - **Độ chính xác huấn luyện (Train Accuracy):** 67.51%
    - **Độ chính xác kiểm tra (Validation Accuracy):** 56.45%
    - **Độ chính xác thử nghiệm(Test Accuracy):** 55.27%
    - **Phân tích:** Độ chính xác huấn luyện cao hơn kiểm tra và thử nghiệm, cho thấy dấu hiệu quá khớp nhẹ. Hiệu suất tổng quát hóa thấp (chỉ khoảng 55%) phản ánh hạn chế của MLP trong việc học các đặc trưng phức tạp.
- **CNN:**
    - **Độ chính xác huấn luyện (Train Accuracy):** 81.70%
    - **Độ chính xác kiểm tra (Validation Accuracy):** 77.95%
    - **Độ chính xác thử nghiệm (Test Accuracy):** 77.38%
    - **Phân tích:** CNN đạt độ chính xác cao hơn đáng kể trên cả ba tập dữ liệu, với sự chênh lệch nhỏ giữa huấn luyện và thử nghiệm (khoảng 4%), cho thấy khả năng tổng quát hóa tốt hơn nhờ các tầng tích chập và dropout.
- **Nhận xét:** CNN vượt trội hơn MLP với khoảng cách hiệu suất khoảng 22% (77.38% so với 55.27%), chứng minh ưu thế của kiến trúc tích chập trong xử lý hình ảnh.

---

3. Đánh giá thông qua learning curves (đường cong học tập)

- **MLP:**
    - **Loss:** Mất mát huấn luyện (Train Loss) giảm từ khoảng 1.7 xuống 1.2 qua 17.5 epoch, trong khi mất mát kiểm tra (Val Loss) giảm từ 1.6 xuống khoảng 1.4, sau đó ổn định. Khoảng cách giữa hai đường cong cho thấy mô hình học tốt trên tập huấn luyện nhưng không cải thiện nhiều trên tập kiểm tra.
    - **Accuracy:** Độ chính xác huấn luyện (Train Accuracy) tăng từ 40% lên hơn 65%, trong khi độ chính xác kiểm tra (Val Accuracy) tăng từ 45% lên khoảng 55% và và có xu hướng ổn định.
    - **Phân tích:** Đường cong học tập của MLP cho thấy tốc độ hội tụ chậm và hiệu suất bão hòa ở mức trung bình, phản ánh hạn chế trong việc học các đặc trưng không gian.
- **CNN:**
    - **Loss:** Mất mát huấn luyện (Train Loss) giảm từ 1.6 xuống khoảng 0.4 qua 17.5 epoch, trong khi mất mát kiểm tra (Val Loss) giảm từ 1.4 xuống khoảng 0.7. Hai đường cong gần nhau hơn, cho thấy mô hình học đồng đều trên cả hai tập dữ liệu.
    - **Accuracy:** Độ chính xác huấn luyện (Train Accuracy) tăng từ hơn 40% lên khoảng hơn 80%, trong khi độ chính xác kiểm tra (Val Accuracy) tăng từ hơn 50% lên gần 80%. Sự chênh lệch nhỏ giữa hai đường cong phản ánh khả năng tổng quát hóa khá tốt.
    - **Phân tích:** CNN hội tụ nhanh hơn và đạt độ chính xác cao hơn, với đường cong mượt mà hơn nhờ khả năng trích xuất đặc trưng không gian hiệu quả.
- **Nhận xét:** Đường cong học tập của CNN cho thấy hiệu suất vượt trội và ổn định hơn MLP, với mất mát thấp hơn và độ chính xác cao hơn, phản ánh ưu thế của các tầng tích chập.

---

4. Đánh giá thông qua ma trận nhầm lẫn (confusion matrix)

- **MLP:**
    - **Đường chéo chính:** Các giá trị lớn trên đường chéo (ví dụ: 554 cho ‘plane’, 644 cho ‘car’, 471 cho ‘bird’, v.v…) cho thấy mô hình dự đoán đúng một số lớp tốt hơn các lớp khác. Tuy nhiên, các giá trị ngoài đường chéo (ví dụ: 144 ‘bird’ bị dự đoán thành ‘deer’, 153 ‘car’ thành ‘truck) lại khá lớn, chỉ ra nhiều lỗi trong việc phân loại.
    - **Lỗi phổ biến:** Các lớp có đặc điểm tương tự (như ‘cat’ và ‘dog’, ‘horse’ và ‘ship’) có nhiều nhầm lẫn (ví dụ: 185 ‘cat’ thành ‘dog, 184 ‘dog’ lại thành ‘cat’), do MLP không trích xuất được đặc trưng không gian.
    - **Phân tích:** Ma trận nhầm lẫn của MLP cho thấy sự phân loại kém, đặc biệt với các lớp phức tạp hoặc tương tự, với tổng số dự đoán đúng chiếm khoảng 55%.
- **CNN:**
    - **Đường chéo chính:** Các giá trị lớn đáng kể so với **MLP** (ví dụ: 809 cho ‘plane’, 879 cho ‘car’, 685 cho ‘bird’, 810 cho ‘deer’, 824 cho ‘frog’, 816 cho ‘horse’, 870 cho ‘ship’, 818 cho ‘truck’), phản ánh khả năng phân loại chính xác cao hơn.
    - **Lỗi phổ biến:** Các giá trị ngoài đường chéo nhỏ hơn đáng kể (ví dụ: chỉ 10 ‘plane’ bị nhầm thành ‘deer’, và không bị nhầm lẫn giữa ‘car’ và ‘frog’, cho thấy ít nhầm lẫn hơn nhiều so với MLP.
    - **Phân tích:** Ma trận nhầm lẫn của CNN cho thấy sự cải thiện rõ rệt, với tổng số dự đoán đúng chiếm khoảng 77%, nhờ khả năng trích xuất đặc trưng không gian hiệu quả.
- **Nhận xét:** CNN có khả năng nhận diện và phân loại tốt hơn đáng kể so với MLP, đặc biệt ở các lớp có đặc điểm tương tự, chứng minh ưu thế của kiến trúc tích chập.

---

5. Kết luận

CNN vượt trội hơn MLP trong bài toán phân loại ảnh CIFAR-10 nhờ kiến trúc tích chập, giúp trích xuất đặc trưng không gian hiệu quả, dẫn đến độ chính xác cao hơn (77.38% so với 55.27%) và khả năng tổng quát hóa tốt hơn. Đường cong học tập của CNN cho thấy

hội tụ nhanh và ổn định, trong khi ma trận nhầm lẫn chỉ ra ít lỗi phân loại hơn, đặc biệt với các lớp có đặc điểm tương tự. MLP, dù đơn giản và dễ triển khai, không phù hợp với dữ liệu hình ảnh do không tận dụng được thông tin không gian, dẫn đến hiệu suất thấp hơn. Kết quả này khẳng định rằng CNN là lựa chọn tối ưu cho các bài toán phân loại hình ảnh, trong khi MLP phù hợp hơn với dữ liệu dạng vector hoặc các bài toán không yêu cầu xử lý không gian.