BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI TẬP LỚN

HỌC PHẦN: NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên: Kim Ngọc Bách

Lớp : D23CQCE06 - B

Sinh viên: Nguyễn Vĩnh Hưng B23DCCE042
Bùi Quang Thắng B23DCCN751

Hà Nội, Tháng 6/2025

Mục lục

1 Phần 1: Chuẩn bị dữ liệu

1.1 Giới thiệu

Phần này trình bày phân tích chi tiết về quá trình chuẩn bị dữ liệu trong mã code Python, bao gồm việc import thư viện, thiết lập tham số, biến đổi dữ liệu, tải và chia dữ liệu, cũng như tạo các data loader cho quá trình huấn luyện mô hình.

1.2 Phân tích mã code

1.2.1 Import thư viện và thiết lập tham số

Đoạn mã sau import các thư viện cần thiết và thiết lập các tham số cho quá trình tải và xử lý dữ liệu:

```
import torch
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
import numpy as np

# number of subprocesses to use for data loading
num_workers = 2
# how many samples per batch to load
batch_size = 20
# percentage of training set to use as validation
valid_size = 0.2
```

Giải thích:

- `from torchvision import datasets`: Import module để tải các tập dữ liệu phổ biến như CIFAR-10.
- `import torchvision.transforms as transforms`: Import module để thực hiện các biến đổi (transform) trên dữ liệu hình ảnh.
- `from torch.utils.data.sampler import SubsetRandomSampler`: Import công cụ để lấy mẫu ngẫu nhiên từ tập dữ liệu.
- `num_workers = 2`: Số lượng tiến trình phụ để tải dữ liệu song song, giúp tăng tốc quá trình xử lý.
- `batch_size = 20`: Số lượng mẫu trong mỗi batch được đưa vào mô hình trong một lần huấn luyện.
- `valid_size = 0.2`: Tỷ lệ 20% của tập huấn luyện được dùng làm tập validation (xác thực).

1.2.2 Biến đổi dữ liệu

Đoạn mã sau định nghĩa các biến đổi sẽ được áp dụng lên dữ liệu hình ảnh:

```
# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # randomly flip and rotate
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Giải thích:

- `transforms.Compose`: Tạo một chuỗi các biến đổi sẽ được áp dụng lên dữ liệu hình ảnh.
- `RandomHorizontalFlip()`: Lật ngẫu nhiên hình ảnh theo chiều ngang, tăng tính đa dạng của dữ liệu (data augmentation).
- `RandomRotation(10)`: Xoay ngẫu nhiên hình ảnh trong khoảng ± 10 độ, cũng để tăng tính đa dạng.
- `ToTensor()`: Chuyển hình ảnh từ định dạng PIL hoặc NumPy sang tensor của PyTorch (có giá trị từ 0 đến 1).
- `Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`: Chuẩn hóa dữ liệu bằng cách trừ trung bình (0.5) và chia cho độ lệch chuẩn (0.5) cho từng kênh RGB, giúp dữ liệu nằm trong khoảng [-1, 1].

1.2.3 Tải và chia dữ liệu

Đoạn mã sau tải tập dữ liệu CIFAR-10 và chia tập huấn luyện thành tập huấn luyện và validation:

```
# choose the training and test datasets
train_data = datasets.CIFAR10('data', train=True,
                              download=True, transform=transform)
test_data = datasets.CIFAR10('data', train=False,
                             download=True, transform=transform)

# obtain training indices that will be used for validation
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]
```

Giải thích:

- `datasets.CIFAR10`: Tải tập dữ liệu CIFAR-10 từ thư viện torchvision.
  - `'data'`: Thư mục lưu trữ dữ liệu.
  - `train=True/False`: Quyết định tải tập huấn luyện hay tập kiểm tra.

- `download=True`: Tự động tải nếu dữ liệu chưa có.
- `transform=transform`: Áp dụng các biến đổi đã định nghĩa ở trên.

- `num_train = len(train_data)`: Lấy số lượng mẫu trong tập huấn luyện (50,000 mẫu trong CIFAR-10).
- `indices = list(range(num_train))`: Tạo danh sách chỉ số từ 0 đến 49,999.
- `np.random.shuffle(indices)`: Xáo trộn ngẫu nhiên các chỉ số để chia dữ liệu một cách ngẫu nhiên.
- `split = int(np.floor(valid_size * num_train))`: Tính điểm chia (20% của 50,000 = 10,000 mẫu cho validation).
- `train_idx, valid_idx`: Chia danh sách chỉ số thành tập huấn luyện (40,000 mẫu) và validation (10,000 mẫu).

1.2.4 Tạo sampler và data loader

Đoạn mã sau tạo các sampler và data loader cho tập huấn luyện, validation và kiểm tra:

```
# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders (combine dataset and sampler)
train_loader = torch.utils.data.DataLoader(train_data,
    batch_size=batch_size,
    sampler=train_sampler, num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(train_data,
    batch_size=batch_size,
    sampler=valid_sampler, num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data,
    batch_size=batch_size,
    num_workers=num_workers)
```

Giải thích:

- `SubsetRandomSampler`: Tạo bộ lấy mẫu ngẫu nhiên từ tập chỉ số `train_idx` và `valid_idx`.
- `torch.utils.data.DataLoader`: Tạo bộ tải dữ liệu để cung cấp dữ liệu theo batch:
  - `train_data/test_data`: Tập dữ liệu nguồn.
  - `batch_size=batch_size`: Kích thước mỗi batch (20 mẫu).
  - `sampler=train_sampler/valid_sampler`: Chỉ định tập chỉ số để lấy dữ liệu.
  - `num_workers=num_workers`: Số tiến trình phụ để tải dữ liệu.

1.2.5 Định nghĩa danh sách lớp

Đoạn mã sau định nghĩa danh sách các lớp trong tập dữ liệu CIFAR-10:

```
# specify the image classes
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']
```

Giải thích:

- Đây là danh sách 10 lớp của tập dữ liệu CIFAR-10, dùng để ánh xạ nhãn số (0-9) thành tên lớp tương ứng.

2 Phần 2: Hiển thị hình ảnh

2.1 Giới thiệu

Phần này trình bày cách hiển thị một batch hình ảnh từ tập huấn luyện cùng với nhãn tương ứng, tương tự như trong mã Python gốc. Trong LaTeX, chúng ta sử dụng gói `subcaption` để tạo các subfigure trong một figure chính, sắp xếp theo lưới 2x10.

2.2 Mã Python gốc

Đoạn mã Python sau được sử dụng để hiển thị hình ảnh trong Jupyter Notebook:

```
import matplotlib.pyplot as plt
# %matplotlib inline

# helper function to un-normalize and display an image
def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    plt.imshow(np.transpose(img, (1, 2, 0)))  # convert from
        Tensor image

# obtain one batch of training images
dataiter = iter(train_loader)
images, labels = next(dataiter)
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding
    labels
fig = plt.figure(figsize=(25, 4))
# display 20 images
for idx in np.arange(20):
    ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])
```

Giải thích:

- `import matplotlib.pyplot as plt`: Thư viện để vẽ hình ảnh.
- `def imshow(img)`: Hàm trợ giúp để hiển thị hình ảnh:

  - `img / 2 + 0.5`: Đảo ngược chuẩn hóa (từ [-1, 1] về [0, 1]).
  - `np.transpose(img, (1, 2, 0))`: Chuyển đổi thứ tự chiều của tensor từ (C, H, W) sang (H, W, C) để phù hợp với `plt.imshow`.

- `dataiter = iter(train_loader)`: Tạo iterator từ `train_loader`.
- `images, labels = next(dataiter)`: Lấy một batch gồm 20 hình ảnh và nhãn tương ứng.
- `images = images.numpy()`: Chuyển tensor sang mảng NumPy để hiển thị.
- Vẽ hình ảnh:
  - `fig = plt.figure(figsize=(25, 4))`: Tạo figure kích thước 25x4.
  - `for idx in np.arange(20)`: Lặp qua 20 hình ảnh trong batch.
  - `add_subplot(2, 10, idx+1)`: Tạo lưới 2 hàng, 10 cột để hiển thị 20 ảnh.
  - `imshow(images[idx])`: Hiển thị từng ảnh.
  - `ax.set_title(classes[labels[idx]])`: Đặt tiêu đề là tên lớp tương ứng.

2.3 Hiển thị hình ảnh trong LaTeX

Để hiển thị tương tự trong LaTeX, chúng ta sử dụng gói `subcaption` để tạo các subfigure trong một figure chính. Dưới đây là mã LaTeX để hiển thị 20 hình ảnh placeholder với nhãn lớp:

Hình 1: Một batch gồm 20 hình ảnh huấn luyện cùng với nhãn tương ứng

Lưu ý:

- Trong mã Python gốc, hình ảnh được hiển thị bằng cách sử dụng `matplotlib` và hàm `imshow` sau khi được chuẩn hóa ngược. Trong LaTeX, chúng ta giả định rằng các hình ảnh đã được lưu dưới dạng file và có thể được bao gồm trực tiếp.
- Do không có hình ảnh thực tế, tất cả các subfigure đều sử dụng `placeholder.png`. Trong thực tế, bạn cần thay thế bằng các file hình ảnh thực tế (ví dụ: `image1.png`, `image2.png`, v.v.) và điều chỉnh nhãn lớp cho phù hợp.
- Nhãn lớp được gán theo chu kỳ từ danh sách `classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']` để minh họa.

3 Phần 3: Định nghĩa các hàm huấn luyện và đánh giá

3.1 Giới thiệu

Phần này trình bày phân tích chi tiết về các hàm được định nghĩa để huấn luyện và đánh giá mô hình trong mã Python. Các hàm này bao gồm hàm huấn luyện một epoch, hàm đánh giá, hàm vẽ đồ thị loss và accuracy, hàm kiểm tra mô hình, hàm huấn luyện mô hình và hàm tìm lớp dự đoán tốt nhất.

3.2 Phân tích mã code

3.2.1 Hàm huấn luyện một epoch

Đoạn mã sau định nghĩa hàm `train_one_epoch`, dùng để huấn luyện mô hình trong một epoch:

```
def train_one_epoch(model, loader, optimizer, criterion, device):
    model.train()
    running_loss, correct, total = 0.0, 0, 0
    for data, target in loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * data.size(0)
        _, preds = torch.max(output, 1)
        correct += (preds == target).sum().item()
        total += target.size(0)
    avg_loss = running_loss / len(loader.dataset)
    accuracy = 100. * correct / total
    return avg_loss, accuracy
```

Giải thích:

- `model.train()`: Đặt mô hình vào chế độ huấn luyện (kích hoạt dropout, batch normalization nếu có).
- `running_loss, correct, total`: Biến tích lũy loss, số dự đoán đúng và tổng số mẫu.
- Vòng lặp qua batch:
  - `data.to(device), target.to(device)`: Chuyển dữ liệu sang GPU/CPU.
  - `optimizer.zero_grad()`: Xóa gradient cũ.
  - `output = model(data)`: Tính đầu ra của mô hình.
  - `loss = criterion(output, target)`: Tính hàm mất mát.
  - `loss.backward()`: Lan truyền ngược để tính gradient.

- `optimizer.step()`: Cập nhật tham số mô hình.
- `running_loss += loss.item() * data.size(0)`: Tích lũy loss.
- `torch.max(output, 1)`: Lấy dự đoán lớp có xác suất cao nhất.
- `correct += (preds == target).sum().item()`: Đếm số dự đoán đúng.
- `total += target.size(0)`: Cộng số mẫu trong batch.

- `avg_loss`: Loss trung bình trên toàn bộ tập dữ liệu.
- `accuracy`: Độ chính xác (%).

3.2.2 Hàm đánh giá

Đoạn mã sau định nghĩa hàm `evaluate`, dùng để đánh giá mô hình trên tập validation hoặc test:

```
def evaluate(model, loader, criterion, device):
    model.eval()
    running_loss, correct, total = 0.0, 0, 0
    all_preds, all_labels = [], []
    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output, target)
            running_loss += loss.item() * data.size(0)
            _, preds = torch.max(output, 1)
            correct += (preds == target).sum().item()
            total += target.size(0)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(target.cpu().numpy())
    avg_loss = running_loss / len(loader.dataset)
    accuracy = 100. * correct / total
    return avg_loss, accuracy, all_preds, all_labels
```

Giải thích:

- `model.eval()`: Đặt mô hình vào chế độ đánh giá (tắt dropout, batch normalization).
- `with torch.no_grad()`: Tắt tính toán gradient để tiết kiệm bộ nhớ và tăng tốc.
- Vòng lặp qua batch: Tương tự như huấn luyện nhưng không cập nhật tham số.
- `all_preds, all_labels`: Lưu trữ dự đoán và nhãn thực tế để phân tích sau.
- Trả về: Loss trung bình, độ chính xác, danh sách dự đoán và nhãn.

3.2.3 Hàm vẽ đồ thị loss và accuracy

Đoạn mã sau định nghĩa hàm `plot_loss_accuracy`, dùng để trực quan hóa loss và accuracy qua các epoch:

```
def plot_loss_accuracy(train_losses, valid_losses, train_accs,
    valid_accs):
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(valid_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss per Epoch')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(train_accs, label='Train Accuracy')
    plt.plot(valid_accs, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.title('Accuracy per Epoch')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Giải thích:

- `plt.figure(figsize=(10, 4))`: Tạo figure kích thước 10x4.
- Biểu đồ Loss:
  - `subplot(1, 2, 1)`: Tạo biểu đồ đầu tiên trong lưới 1x2.
  - `plt.plot`: Vẽ đường loss cho huấn luyện và validation.
  - Thêm nhãn, tiêu đề, lưới và chú thích.
- Biểu đồ Accuracy: Tương tự, vẽ độ chính xác (%).
- `plt.tight_layout()`: Điều chỉnh khoảng cách giữa các biểu đồ.

3.2.4 Hàm kiểm tra mô hình

Đoạn mã sau định nghĩa hàm `test_model`, dùng để kiểm tra mô hình trên tập test và hiển thị ma trận nhầm lẫn:

```
def test_model(model, loader, criterion, classes, device):
    model.eval()
    test_loss = 0.0
    class_correct = [0 for _ in range(len(classes))]
    class_total = [0 for _ in range(len(classes))]
    all_preds, all_labels = [], []

    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(device), target.to(device)
```

```
            output = model(data)
            loss = criterion(output, target)
            test_loss += loss.item() * data.size(0)
            _, preds = torch.max(output, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(target.cpu().numpy())
            correct_tensor = preds.eq(target.data.view_as(preds))
            correct = correct_tensor.cpu().numpy()
            for i in range(len(data)):
                label = target[i]
                class_correct[label] += correct[i]
                class_total[label] += 1

    test_loss /= len(loader.dataset)
    overall_acc = 100. * sum(class_correct) / sum(class_total)
    print(f'\nTest Loss: {test_loss:.4f}')
    for i in range(len(classes)):
        if class_total[i] > 0:
            acc = 100. * class_correct[i] / class_total[i]
            print(f'Test Accuracy of {classes[i]:5s}: {acc:.2f}\%
               ({class_correct[i]}/{class_total[i]})')
    print(f'\nOverall Test Accuracy: {overall_acc:.2f}\%')

    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
    return overall_acc, class_correct, class_total
```

Giải thích:

- Tính toán loss và accuracy từng lớp:
  - `class_correct, class_total`: Lưu số dự đoán đúng và tổng số mẫu cho từng lớp.
  - `correct_tensor = preds.eq(target)`: So sánh dự đoán với nhãn thực tế.
  - Vòng lặp để cập nhật `class_correct` và `class_total` cho từng mẫu.
- `test_loss`: Loss trung bình trên tập kiểm tra.
- `overall_acc`: Độ chính xác tổng thể (%).
- In kết quả: Loss và accuracy cho từng lớp và tổng thể.
- Vẽ ma trận nhầm lẫn:
  - `confusion_matrix`: Tạo ma trận nhầm lẫn từ dự đoán và nhãn thực tế.
  - `sns.heatmap`: Vẽ heatmap với chú thích số, dùng màu xanh (Blues).

3.2.5 Hàm huấn luyện mô hình

Đoạn mã sau định nghĩa hàm `train_model`, dùng để huấn luyện mô hình qua nhiều epoch và theo dõi tiến trình:

```
def train_model(model, train_loader, valid_loader, criterion,
    optimizer, device, classes, n_epochs=20):
    train_losses, valid_losses = [], []
    train_accuracies, valid_accuracies = [], []

    for epoch in range(1, n_epochs+1):
        train_loss, train_acc = train_one_epoch(model,
            train_loader, optimizer, criterion, device)
        valid_loss, valid_acc, _, _ = evaluate(model,
            valid_loader, criterion, device)

        train_losses.append(train_loss)
        valid_losses.append(valid_loss)
        train_accuracies.append(train_acc)
        valid_accuracies.append(valid_acc)

        print(f"Epoch {epoch}: "
              f"Train Loss = {train_loss:.4f}, Valid Loss =
                  {valid_loss:.4f}, "
              f"Train Acc = {train_acc:.2f}\%, Valid Acc =
                  {valid_acc:.2f}\%")

    plot_loss_accuracy(train_losses, valid_losses,
        train_accuracies, valid_accuracies)

    avg_train_loss = np.mean(train_losses)
    avg_train_acc = np.mean(train_accuracies)
    print(f"\n---  T ng  k ít  h u n  l u y n  cho
        {model.__class__.__name__} ---")
    print(f"Trung b nh Train Loss: {avg_train_loss:.4f}")
    print(f"Trung b nh Train Accuracy: {avg_train_acc:.2f}\%")

    return avg_train_loss, avg_train_acc, train_losses,
        valid_losses, train_accuracies, valid_accuracies
```

Giải thích:

- Vòng lặp qua epoch:
  - Gọi `train_one_epoch` để huấn luyện một epoch.
  - Gọi `evaluate` để đánh giá trên tập validation.
  - Lưu trữ loss và accuracy qua từng epoch.
- In tiến trình: Loss và accuracy sau mỗi epoch.
- Vẽ đồ thị: Gọi `plot_loss_accuracy` để trực quan hóa kết quả.
- Tổng kết: Tính trung bình loss và accuracy huấn luyện.

3.2.6 Hàm tìm lớp dự đoán tốt nhất

Đoạn mã sau định nghĩa hàm `find_best_performing_class`, dùng để xác định lớp có độ chính xác cao nhất trên tập test:

```
def find_best_performing_class(class_correct, class_total,
    classes, model_name):
    accuracies = []
    for i in range(len(classes)):
        if class_total[i] > 0:
            acc = 100. * class_correct[i] / class_total[i]
            accuracies.append((acc, classes[i]))

    highest_accuracy = max(accuracies, key=lambda x: x[0])
    print(f"Lop co phan tram du doan cao nhat cho {model_name}:
        '{highest_accuracy[1]}' v i {highest_accuracy[0]:.2f}\%")
```

Giải thích:

- `accuracies`: Danh sách chứa độ chính xác và tên lớp.
- `max(accuracies, key=lambda x: x[0])`: Tm lp có chnh xác cao nht.
- In kết quả: Lớp có hiệu suất tốt nhất cho mô hình.

4 Phần 4: Định nghĩa và huấn luyện mô hình CNN

4.1 Giới thiệu

Phần này trình bày phân tích chi tiết về việc định nghĩa và huấn luyện mô hình mạng nơ-ron tích chập (CNN) trong mã Python. Nội dung bao gồm định nghĩa kiến trúc CNN, khởi tạo mô hình, thiết lập hàm mất mát, tối ưu hóa và huấn luyện mô hình trên tập dữ liệu CIFAR-10.

4.2 Phân tích mã code

4.2.1 Định nghĩa kiến trúc CNN

Đoạn mã sau định nghĩa lớp `CNN`, xác định kiến trúc của mạng nơ-ron tích chập:

```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # convolutional layer (sees 32x32x3 image tensor)
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        # convolutional layer (sees 16x16x16 tensor)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        # convolutional layer (sees 8x8x32 tensor)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # linear layer (64 * 4 * 4 -> 500)
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        # linear layer (500 -> 10)
        self.fc2 = nn.Linear(500, 10)
        # dropout layer (p=0.25)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        # flatten image input
        x = x.view(-1, 64 * 4 * 4)
        # add dropout layer
        x = self.dropout(x)
        # add 1st hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        # add dropout layer
        x = self.dropout(x)
        # add 2nd hidden layer
        x = self.fc2(x)
```

```
38         return x
```

Giải thích:

- `nn.Module`: Lớp cơ sở cho tất cả mô hình trong PyTorch, cung cấp các phương thức để quản lý tham số và lan truyền thuận.
- `__init__`: Khởi tạo các tầng của mạng:
  - `conv1`: Tầng tích chập đầu tiên, nhận đầu vào 3 kênh (RGB) và tạo 16 kênh đầu ra, sử dụng kernel 3x3 với padding=1 để giữ kích thước không gian.
  - `conv2`: Tầng tích chập thứ hai, nhận 16 kênh và tạo 32 kênh đầu ra.
  - `conv3`: Tầng tích chập thứ ba, nhận 32 kênh và tạo 64 kênh đầu ra.
  - `pool`: Tầng max pooling 2x2, giảm kích thước không gian xuống một nửa sau mỗi lần áp dụng.
  - `fc1`: Tầng fully connected, chuyển từ 64 * 4 * 4 (kích thước sau tích chập và pooling) sang 500 đơn vị.
  - `fc2`: Tầng fully connected cuối, chuyển từ 500 đơn vị sang 10 đơn vị (tương ứng với 10 lớp của CIFAR-10).
  - `dropout`: Tầng dropout với xác suất 0.25, giúp ngăn chặn overfitting bằng cách ngẫu nhiên vô hiệu hóa các nơ-ron trong quá trình huấn luyện.
- `forward`: Định nghĩa luồng dữ liệu qua mạng:
  - Áp dụng lần lượt ba tầng tích chập, mỗi tầng kèm theo hàm kích hoạt ReLU và max pooling.
  - Làm phẳng tensor đầu ra bằng `x.view(-1, 64 * 4 * 4)` để chuẩn bị cho tầng fully connected.
  - Áp dụng dropout, tầng fully connected đầu tiên với ReLU, dropout lần nữa, và cuối cùng là tầng fully connected thứ hai.
  - Trả về đầu ra với 10 giá trị, tương ứng với xác suất dự đoán cho mỗi lớp.

4.2.2 Khởi tạo và huấn luyện CNN

Đoạn mã sau khởi tạo mô hình CNN, thiết lập hàm mất mát, tối ưu hóa và huấn luyện mô hình:

```
device = torch.device("cuda" if torch.cuda.is_available() else
    "cpu")
model_1 = CNN()
model_1.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_1.parameters(), lr=0.001)

# Training
cnn_avg_train_loss, cnn_avg_train_acc, cnn_train_losses,
    cnn_valid_losses, cnn_train_accuracies, cnn_valid_accuracies =
    train_model(model_1, train_loader, valid_loader, criterion,
    optimizer, device, classes, n_epochs=30)
```

```
10
11 # Test
12 cnn_test_acc, cnn_class_correct, cnn_class_total =
      test_model(model_1, test_loader, criterion, classes, device)
```

Giải thích:

- `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`: Kiểm tra xem có GPU khả dụng không; nếu có, sử dụng GPU, nếu không, sử dụng CPU.
- `model_1 = CNN()`: Khởi tạo một thể hiện của lớp `CNN`.
- `model_1.to(device)`: Chuyển mô hình sang thiết bị được chọn (GPU hoặc CPU).
- `criterion = nn.CrossEntropyLoss()`: Định nghĩa hàm mất mát CrossEntropy, phù hợp cho bài toán phân loại đa lớp.
- `optimizer = torch.optim.Adam(model_1.parameters(), lr=0.001)`: Sử dụng thuật toán tối ưu hóa Adam với tốc độ học (learning rate) là 0.001 để cập nhật các tham số của mô hình.
- `train_model(...)`: Gọi hàm `train_model` để huấn luyện mô hình qua 30 epoch, sử dụng các data loader đã chuẩn bị (`train_loader`, `valid_loader`), hàm mất mát, tối ưu hóa, thiết bị, danh sách lớp và số epoch. Hàm trả về các giá trị loss và accuracy trung bình, cũng như danh sách loss và accuracy qua từng epoch.
- `test_model(...)`: Gọi hàm `test_model` để đánh giá mô hình trên tập kiểm tra (`test_loader`), trả về độ chính xác tổng thể và độ chính xác cho từng lớp.

4.3 Kết quả huấn luyện

4.3.1 Loss và Accuracy qua từng Epoch

Bảng sau đây liệt kê các giá trị Train Loss, Validation Loss, Train Accuracy và Validation Accuracy qua 30 epoch:

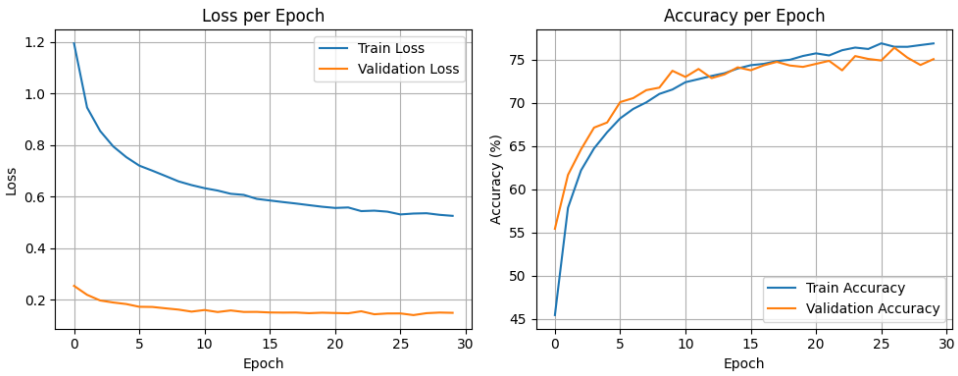
Bảng 1: Loss và Accuracy qua từng Epoch của mô hình CNN

Epoch	Train Loss	Validation Loss	Train Accuracy (%)	Validation Accuracy (%)
1	1.1949	0.2533	45.46	55.44
2	0.9456	0.2186	57.87	61.67
3	0.8547	0.1966	62.19	64.62
4	0.7947	0.1887	64.74	67.14
5	0.7532	0.1827	66.60	67.73
6	0.7201	0.1721	68.21	70.09
7	0.7005	0.1714	69.30	70.55
8	0.6800	0.1661	70.06	71.47
9	0.6591	0.1612	71.03	71.76
10	0.6445	0.1533	71.54	73.71
11	0.6326	0.1595	72.39	73.00
12	0.6233	0.1517	72.74	73.92
13	0.6108	0.1579	73.11	72.86
14	0.6065	0.1520	73.43	73.30
15	0.5910	0.1522	73.97	74.11
16	0.5848	0.1500	74.36	73.77
17	0.5787	0.1495	74.51	74.34
18	0.5730	0.1498	74.83	74.74
19	0.5666	0.1472	74.98	74.32
20	0.5605	0.1492	75.42	74.16
21	0.5558	0.1479	75.72	74.51
22	0.5577	0.1468	75.48	74.86
23	0.5435	0.1546	76.09	73.76
24	0.5453	0.1430	76.39	75.41
25	0.5414	0.1461	76.23	75.08
26	0.5306	0.1463	76.88	74.90
27	0.5341	0.1398	76.49	76.38
28	0.5353	0.1473	76.48	75.23
29	0.5291	0.1497	76.68	74.38
30	0.5252	0.1486	76.87	75.06

Phân tích:

- **Train Loss**: Giảm dần từ 1.1949 ở epoch 1 xuống 0.5252 ở epoch 30, cho thấy mô hình học tốt trên tập huấn luyện.
- **Validation Loss**: Giảm từ 0.2533 xuống mức thấp nhất là 0.1398 (epoch 27), nhưng có dao động nhẹ, cho thấy mô hình ổn định nhưng có thể hơi overfitting.
- **Train Accuracy**: Tăng từ 45.46% lên 76.87%, thể hiện khả năng phân loại cải thiện đáng kể.

- **Validation Accuracy**: Tăng từ 55.44% lên 76.38% (cao nhất ở epoch 27), cho thấy mô hình tổng quát hóa tốt trên tập validation.



Hình 2: Biểu đồ Train Loss và Validation Loss qua các epoch của mô hình CNN.

4.3.2 Tổng kết huấn luyện

Kết quả tổng kết huấn luyện được in như sau:

```
--- Tong ket huan luyen cho CNN ---
Trung binh Train Loss: 0.6424
Trung binh Train Accuracy: 71.67%
```

Giải thích:

- `Trung binh Train Loss: 0.6424`: Loss trung bình qua 30 epoch trên tập huấn luyện, phản ánh mức độ sai lệch trung bình của mô hình.
- `Trung binh Train Accuracy: 71.67%`: Độ chính xác trung bình trên tập huấn luyện, cho thấy hiệu suất tổng thể của mô hình trong quá trình huấn luyện.

4.4 Kết quả kiểm tra

Kết quả kiểm tra trên tập test được in như sau:

```
Test Loss: 0.7446
Test Accuracy of airplane: 80.60% (806/1000)
Test Accuracy of automobile: 83.10% (831/1000)
Test Accuracy of bird: 61.10% (611/1000)
Test Accuracy of cat: 64.40% (644/1000)
Test Accuracy of deer: 74.60% (746/1000)
Test Accuracy of dog: 61.90% (619/1000)
Test Accuracy of frog: 73.30% (733/1000)
Test Accuracy of horse: 79.80% (798/1000)
Test Accuracy of ship: 91.60% (916/1000)
Test Accuracy of truck: 78.60% (786/1000)

Overall Test Accuracy: 74.90%
```

Để trình bày kết quả kiểm tra một cách rõ ràng, bảng sau liệt kê độ chính xác của từng lớp:

Bảng 2: Độ chính xác kiểm tra trên tập test của mô hình CNN

Lớp	Độ chính xác (%)	Số mẫu đúng/Tổng số mẫu
airplane	80.60	806/1000
automobile	83.10	831/1000
bird	61.10	611/1000
cat	64.40	644/1000
deer	74.60	746/1000
dog	61.90	619/1000
frog	73.30	733/1000
horse	79.80	798/1000
ship	91.60	916/1000
truck	78.60	786/1000
Overall	**74.90**	**7490/10000**

Phân tích:

- **Test Loss: 0.7446**: Loss trên tập test, cao hơn một chút so với validation loss, cho thấy mô hình có thể gặp khó khăn nhẹ khi tổng quát hóa trên dữ liệu mới.
- **Độ chính xác từng lớp**:
  - Lớp `ship` đạt độ chính xác cao nhất (91.60%), với 916/1000 mẫu được phân loại đúng, cho thấy mô hình học tốt các đặc trưng của lớp này.
  - Lớp `bird` và `dog` có độ chính xác thấp nhất (61.10% và 61.90%), có thể do các đặc trưng của các lớp này dễ bị nhầm lẫn.
  - Các lớp khác như `automobile` (83.10%) và `airplane` (80.60%) cũng đạt hiệu suất tốt.
- **Overall Test Accuracy: 74.90%**: Độ chính xác tổng thể trên tập test, với 7490/10000 mẫu được phân loại đúng, cho thấy mô hình CNN hoạt động khá tốt trên tập dữ liệu CIFAR-10.

Hình 3: Biểu đồ Train Accuracy và Validation Accuracy qua các epoch của mô hình CNN.

5 Phần 5: Định nghĩa và huấn luyện mô hình MLP

5.1 Giới thiệu

Phần này trình bày phân tích chi tiết về việc định nghĩa và huấn luyện mô hình mạng nơ-ron đa tầng (MLP - Multi-Layer Perceptron) trong mã Python. Nội dung bao gồm định nghĩa kiến trúc MLP, khởi tạo mô hình, thiết lập hàm mất mát, tối ưu hóa và huấn luyện mô hình trên tập dữ liệu CIFAR-10.

5.2 Phân tích mã code

5.2.1 Định nghĩa kiến trúc MLP

Đoạn mã sau định nghĩa lớp `MLP3Layer`, xác định kiến trúc của mạng nơ-ron đa tầng:

```
class MLP3Layer(nn.Module):
    def __init__(self, input_size=3*32*32, hidden1=512,
       hidden2=256, num_classes=10):
        super(MLP3Layer, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, num_classes)
        self.dropout = nn.Dropout(0.3)
```

```
8
9       def forward(self, x):
10          x = x.view(x.size(0), -1)  # Flatten  nh
11          x = torch.relu(self.fc1(x))
12          x = self.dropout(x)
13          x = torch.relu(self.fc2(x))
14          x = self.dropout(x)
15          x = self.fc3(x)
16          return x
```

Giải thích:

- `nn.Module`: Lớp cơ sở cho tất cả mô hình trong PyTorch, cung cấp các phương thức để quản lý tham số và lan truyền thuận.
- `__init__`: Khởi tạo các tầng của mạng với các tham số:
  - `input_size=3*32*32`: Kích thước đầu vào là 3072, tương ứng với ảnh CIFAR-10 (3 kênh RGB, 32x32 pixel) được làm phẳng.
  - `hidden1=512`: Kích thước của tầng ẩn thứ nhất.
  - `hidden2=256`: Kích thước của tầng ẩn thứ hai.
  - `num_classes=10`: Số lớp đầu ra, tương ứng với 10 lớp của CIFAR-10.
  - `fc1`: Tầng fully connected đầu tiên, chuyển từ 3072 đơn vị sang 512 đơn vị.
  - `fc2`: Tầng fully connected thứ hai, chuyển từ 512 đơn vị sang 256 đơn vị.
  - `fc3`: Tầng fully connected cuối, chuyển từ 256 đơn vị sang 10 đơn vị.
  - `dropout`: Tầng dropout với xác suất 0.3, giúp ngăn chặn overfitting bằng cách ngẫu nhiên vô hiệu hóa các nơ-ron trong quá trình huấn luyện.
- `forward`: Định nghĩa luồng dữ liệu qua mạng:
  - `x.view(x.size(0), -1)`: Làm phẳng ảnh đầu vào từ kích thước (batch, 3, 32, 32) thành (batch, 3072).
  - Áp dụng tầng `fc1` với hàm kích hoạt ReLU, sau đó là dropout.
  - Áp dụng tầng `fc2` với ReLU và dropout lần nữa.
  - Áp dụng tầng `fc3` để tạo đầu ra với 10 giá trị, tương ứng với xác suất dự đoán cho mỗi lớp.

5.2.2 Khởi tạo và huấn luyện MLP

Đoạn mã sau khởi tạo mô hình MLP, thiết lập hàm mất mát, tối ưu hóa và huấn luyện mô hình:

```
1 device = torch.device("cuda" if torch.cuda.is_available() else
     "cpu")
2 model_2 = MLP3Layer()
3 model_2.to(device)
4
5 criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(model_2.parameters(), lr=0.001)

# Training
mlp_avg_train_loss, mlp_avg_train_acc, mlp_train_losses,
    mlp_valid_losses, mlp_train_accuracies, mlp_valid_accuracies =
    train_model(model_2, train_loader, valid_loader, criterion,
    optimizer, device, classes, n_epochs=30)

# Test
mlp_test_acc, mlp_class_correct, mlp_class_total =
    test_model(model_2, test_loader, criterion, classes, device)
```

Giải thích:

- `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`: Kiểm tra xem có GPU khả dụng không; nếu có, sử dụng GPU, nếu không, sử dụng CPU.
- `model_2 = MLP3Layer()`: Khởi tạo một thể hiện của lớp `MLP3Layer` với các tham số mặc định.
- `model_2.to(device)`: Chuyển mô hình sang thiết bị được chọn (GPU hoặc CPU).
- `criterion = nn.CrossEntropyLoss()`: Định nghĩa hàm mất mát CrossEntropy, phù hợp cho bài toán phân loại đa lớp.
- `optimizer = torch.optim.Adam(model_2.parameters(), lr=0.001)`: Sử dụng thuật toán tối ưu hóa Adam với tốc độ học (learning rate) là 0.001 để cập nhật các tham số của mô hình.
- `train_model(...)`: Gọi hàm `train_model` để huấn luyện mô hình qua 30 epoch, sử dụng các data loader đã chuẩn bị (`train_loader`, `valid_loader`), hàm mất mát, tối ưu hóa, thiết bị, danh sách lớp và số epoch. Hàm trả về các giá trị loss và accuracy trung bình, cũng như danh sách loss và accuracy qua từng epoch.
- `test_model(...)`: Gọi hàm `test_model` để đánh giá mô hình trên tập kiểm tra (`test_loader`), trả về độ chính xác tổng thể và độ chính xác cho từng lớp.

5.3 Kết quả huấn luyện

5.3.1 Loss và Accuracy qua từng Epoch

Bảng sau đây liệt kê các giá trị Train Loss, Validation Loss, Train Accuracy và Validation Accuracy qua 30 epoch:

Bảng 3: Loss và Accuracy qua từng Epoch của mô hình MLP

Epoch	Train Loss	Validation Loss	Train Accuracy (%)	Validation Accuracy (%)
1	1.4706	0.3389	34.16	39.75
2	1.3981	0.3356	37.84	39.74
3	1.3768	0.3221	39.22	43.34
4	1.3530	0.3199	40.23	42.96
5	1.3438	0.3205	40.57	43.39
6	1.3335	0.3186	41.16	43.90
7	1.3236	0.3147	41.57	44.58
8	1.3162	0.3111	41.71	45.08
9	1.3192	0.3095	41.51	44.69
10	1.2994	0.3097	42.48	45.60
11	1.3038	0.3116	42.08	44.08
12	1.3035	0.3092	42.65	45.29
13	1.2834	0.3059	43.13	46.47
14	1.2896	0.3122	43.10	44.30
15	1.2853	0.3036	43.42	46.06
16	1.2812	0.3039	43.41	45.70
17	1.2777	0.3014	43.59	46.28
18	1.2784	0.3058	43.63	46.02
19	1.2784	0.3036	43.81	46.28
20	1.2702	0.3004	43.68	47.03
21	1.2730	0.3020	44.15	44.86
22	1.2655	0.3001	44.06	46.72
23	1.2644	0.3009	44.16	46.87
24	1.2654	0.3044	43.80	46.29
25	1.2659	0.3024	44.00	46.72
26	1.2609	0.3056	44.41	44.55
27	1.2599	0.2986	44.60	46.69
28	1.2565	0.3031	44.77	46.15
29	1.2491	0.3013	44.92	46.35
30	1.2561	0.2963	44.40	46.73

Phân tích:

- **Train Loss**: Giảm từ 1.4706 ở epoch 1 xuống 1.2561 ở epoch 30, cho thấy mô hình học được một phần trên tập huấn luyện, nhưng tiến trình cải thiện chậm và loss vẫn ở mức cao.
- **Validation Loss**: Giảm từ 0.3389 xuống mức thấp nhất là 0.2963 (epoch 30), nhưng dao động nhẹ quanh mức 0.30, cho thấy mô hình không cải thiện nhiều trên tập validation.
- **Train Accuracy**: Tăng từ 34.16% lên 44.92% (cao nhất ở epoch 29), nhưng độ chính xác vẫn thấp, cho thấy mô hình MLP gặp khó khăn trong việc học các đặc trưng phức tạp từ dữ liệu hình ảnh.
- **Validation Accuracy**: Tăng từ 39.75% lên 47.03% (cao nhất ở epoch 20), nhưng không vượt quá 50%, cho thấy mô hình tổng quát hóa kém trên tập validation.

5.3.2 Tổng kết huấn luyện

Kết quả tổng kết huấn luyện được in như sau:

```
--- Tong ket huan luyen cho MLP3Layer ---
Trung binh Train Loss: 1.3001
Trung binh Train Accuracy: 42.54%
```

Giải thích:

- `Trung binh Train Loss: 1.3001`: Loss trung bình qua 30 epoch trên tập huấn luyện, khá cao, cho thấy mô hình chưa học tốt.
- `Trung binh Train Accuracy: 42.54%`: Độ chính xác trung bình trên tập huấn luyện, khá thấp, phản ánh hiệu suất hạn chế của mô hình MLP trên tập dữ liệu CIFAR-10.

5.4 Kết quả kiểm tra

Kết quả kiểm tra trên tập test được in như sau:

```
Test Loss: 1.4661
Test Accuracy of airplane: 55.30% (553/1000)
Test Accuracy of automobile: 65.90% (659/1000)
Test Accuracy of bird: 35.10% (351/1000)
Test Accuracy of cat: 46.20% (462/1000)
Test Accuracy of deer: 32.50% (325/1000)
Test Accuracy of dog: 22.30% (223/1000)
Test Accuracy of frog: 57.00% (570/1000)
Test Accuracy of horse: 55.70% (557/1000)
Test Accuracy of ship: 59.40% (594/1000)
Test Accuracy of truck: 49.70% (497/1000)

Overall Test Accuracy: 47.91%
```

Bảng sau liệt kê độ chính xác của từng lớp trên tập test:
Phân tích:

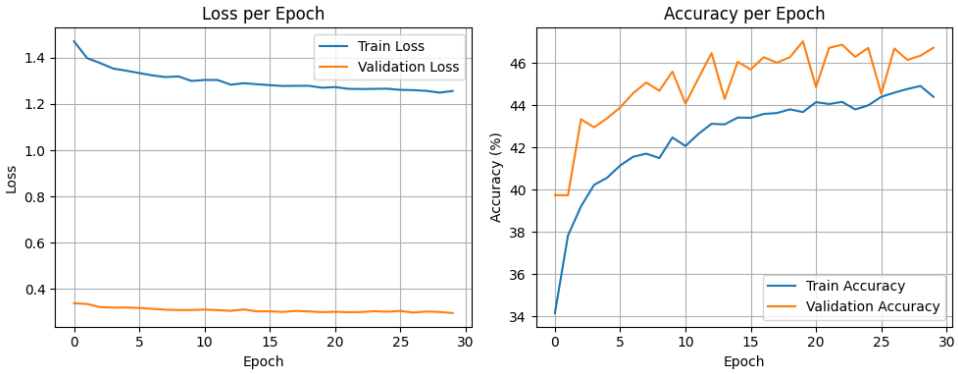
- **Test Loss: 1.4661**: Loss trên tập test, rất cao, cho thấy mô hình không tổng quát hóa tốt trên dữ liệu mới.
- **Độ chính xác từng lớp**:
  - Lớp `automobile` đạt độ chính xác cao nhất (65.90%), với 659/1000 mẫu được phân loại đúng.
  - Lớp `dog` có độ chính xác thấp nhất (22.30%), với chỉ 223/1000 mẫu đúng, cho thấy mô hình gặp khó khăn lớn trong việc phân loại lớp này.
  - Các lớp như `bird` (35.10%) và `deer` (32.50%) cũng có độ chính xác thấp, có thể do các đặc trưng của chúng bị nhầm lẫn với các lớp khác.
  - Các lớp như `ship` (59.40%) và `frog` (57.00%) đạt hiệu suất trung bình.
- **Overall Test Accuracy: 47.91%**: Độ chính xác tổng thể trên tập test, với 4791/10000 mẫu được phân loại đúng, cho thấy mô hình MLP hoạt động kém trên tập dữ liệu CIFAR-10 so với mô hình CNN (74.90% ở phần 4).

Bảng 4: Độ chính xác kiểm tra trên tập test của mô hình MLP

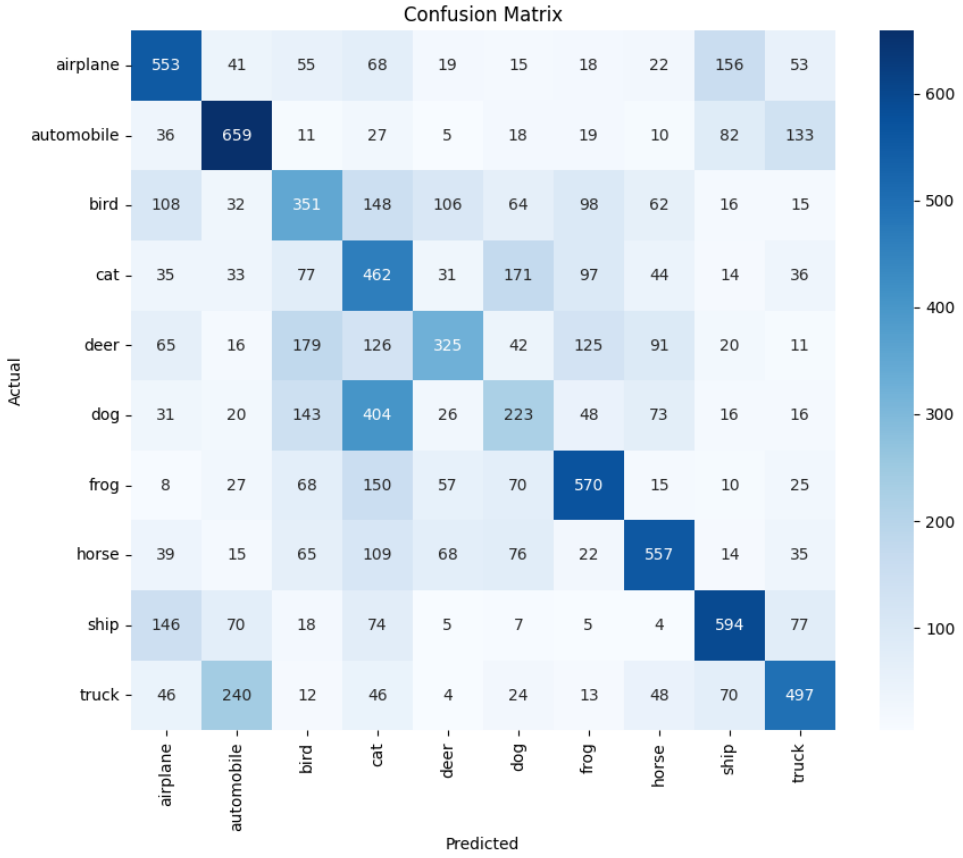
Lớp	Độ chính xác (%)	Số mẫu đúng/Tổng số mẫu
airplane	55.30	553/1000
automobile	65.90	659/1000
bird	35.10	351/1000
cat	46.20	462/1000
deer	32.50	325/1000
dog	22.30	223/1000
frog	57.00	570/1000
horse	55.70	557/1000
ship	59.40	594/1000
truck	49.70	497/1000
Overall	**47.91**	**4791/10000**

5.5 Biểu đồ Loss và Accuracy qua các Epoch

Hai biểu đồ dưới đây được tạo bởi hàm `plot_loss_accuracy`, minh họa sự thay đổi của Loss và Accuracy qua 30 epoch trong quá trình huấn luyện mô hình MLP:



Hình 4: Biểu đồ Train Loss và Validation Loss qua các epoch của mô hình MLP.



Hình 5: Biểu đồ Train Accuracy và Validation Accuracy qua các epoch của mô hình MLP.

6 Phần 6: So sánh và kết luận

6.1 Giới thiệu

Phần này trình bày phân tích chi tiết về đoạn mã Python thực hiện so sánh hiệu suất của hai mô hình: mạng nơ-ron tích chập (CNN) và mạng nơ-ron đa tầng (MLP). Mã in kết quả huấn luyện trung bình, so sánh độ chính xác trên tập kiểm tra, đưa ra kết luận về mô hình nào tốt hơn, và xác định lớp có độ chính xác dự đoán cao nhất.

6.2 Phân tích mã code

6.2.1 Mã so sánh và kết quả

Đoạn mã sau thực hiện so sánh và in kết quả cuối của hai mô hình:

```
print("\n============================================")
print("============= BAO CAO SO SANH HAI MO HINH =============")
print("============================================\n")

print("--- Ket qua Huan luyen Trung binh ---")
print(f"CNN: Trung binh Train Loss = {cnn_avg_train_loss:.4f},
   Trung binh Train Accuracy = {cnn_avg_train_acc:.2f}%")
print(f"MLP: Trung binh Train Loss = {mlp_avg_train_loss:.4f},
   Trung binh Train Accuracy = {mlp_avg_train_acc:.2f}%\n")

print("--- So sanh Test Accuracy cua 2 mo hinh ---")
print(f"Test Accuracy cua CNN: {cnn_test_acc:.2f}%")
print(f"Test Accuracy cua MLP: {mlp_test_acc:.2f}%")

if cnn_test_acc > mlp_test_acc:
    print("=> Ket luan: Mo hinh **CNN** cho hieu suat tong the
       tren tap test tot hon.")
elif mlp_test_acc > cnn_test_acc:
    print("=> Ket luan: Mo hinh **MLP** cho hieu suat tong the
       tren tap test tot hon.")
else:
    print("=> Ket luan: Ca hai mo hinh co Test Accuracy tuong
       duong.")

print("\n--- Phan tich lop du doan tot nhat ---")
find_best_performing_class(cnn_class_correct, cnn_class_total,
   classes, "CNN")
find_best_performing_class(mlp_class_correct, mlp_class_total,
   classes, "MLP")
```

Giải thích:

- `print("============================================")`: In các dòng phân cách và tiêu đề báo cáo để làm rõ phần so sánh.
- In kết quả huấn luyện trung bình:
  - `cnn_avg_train_loss, cnn_avg_train_acc`: Loss và độ chính xác trung bình của CNN trên tập huấn luyện.
  - `mlp_avg_train_loss, mlp_avg_train_acc`: Tương tự cho MLP.
  - Sử dụng f-string để in giá trị với độ chính xác 4 chữ số thập phân cho loss và 2 chữ số thập phân cho accuracy.
- So sánh độ chính xác trên tập kiểm tra:
  - `cnn_test_acc`: Độ chính xác của CNN trên tập kiểm tra.
  - `mlp_test_acc`: Độ chính xác của MLP trên tập kiểm tra.
  - In cả hai giá trị với định dạng 2 chữ số thập phân.
- Đưa ra kết luận:

- Sử dụng cấu trúc `if-elif-else` để so sánh `cnn_test_acc` và `mlp_test_acc`.
- Nếu `cnn_test_acc` lớn hơn, kết luận CNN tốt hơn.
- Nếu `mlp_test_acc` lớn hơn, kết luận MLP tốt hơn.
- Nếu bằng nhau, kết luận cả hai mô hình có hiệu suất tương đương.

- Phân tích lớp dự đoán tốt nhất:
  - Gọi hàm `find_best_performing_class` hai lần, cho CNN và MLP.
  - Hàm xác định lớp có độ chính xác dự đoán cao nhất trên tập kiểm tra, dựa trên `cnn_class_correct`, `cnn_class_total` (cho CNN) và `mlp_class_correct`, `mlp_class_total` (cho MLP).
  - In tên lớp và độ chính xác cao nhất cho mỗi mô hình.

6.3 Kết quả thu được

Kết quả Huấn luyện Trung bình
CNN: Trung bình Train Loss = 0.6424, Trung bình Train Accuracy = 71.67%
MLP: Trung bình Train Loss = 1.3001, Trung bình Train Accuracy = 42.54%

So sánh Test Accuracy của 2 mô hình
Test Accuracy của CNN: 74.90%
Test Accuracy của MLP: 47.91%
⇒ **Kết luận**: Mô hình **CNN** cho hiệu suất tổng thể trên tập test tốt hơn.

Phân tích lớp dự đoán tốt nhất
Lớp có phần trăm dự đoán cao nhất cho CNN: 'ship' với 91.60%
Lớp có phần trăm dự đoán cao nhất cho MLP: 'automobile' với 65.90%