HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CNTT - CLC

BỘ MÔN LẬP TRÌNH PYTHON

BÀI TẬP LỚN PYTHON

Giảng viên : **Kim Ngọc Bách**

Họ và tên : **Nguyễn Thiên Trung - B23DCCE093**

Trần Minh Việt - B23DCVT458

Lớp : **D23CQCE06 – B**

Năm học 2024 - 2025

Mục lục

I. Đề bài: 3
1. Mục tiêu 3
2. Dữ liệu: CIFAR-10 3
II. Code Python 4
1. Thư viện cần cài 4
2. Giải thích chi tiết các hàm chính 5
2.1. Hàm set_seed(seed=42) 5
2.2. Hàm get_device() 5
2.3. Hàm create_loaders(batch_size=64, val_ratio=0.1) 6
2.4. Hàm train_epoch(model, loader, criterion, optimizer, device) 7
2.5. Hàm eval_model(model, loader, criterion, device) 8
2.6. Hàm plot_learning_curve(epochs, train_loss, val_loss, train_acc, val_acc, model_name) 9
2.7. Hàm plot_confusion_matrix(true, pred, classes, model_name) 10
2.8. Hàm train_and_test(model_class, name, train_loader, val_loader, test_loader, device, epochs=20, lr=1e-3) 11
3. Huấn luyện và đánh giá 12
4. Kết quả 12
4.1. Đường cong học tập 12
4.2. Ma trận nhầm lẫn 13
4.3. Độ chính xác 14
5. Thảo luận 14
6. Kết luận 15

I. Đề bài:

- Perform image classification using the CIFAR-10 dataset: https://www.cs.toronto.edu/~kriz/cifar.html

Tasks:

- Build a basic MLP (Multi-Layer Perceptron) neural network with 3 layers.
- Build a Convolutional Neural Network (CNN) with 3 convolution layers.
- Perform image classification using both neural networks, including training, validation, and testing.
- Plot learning curves.
- Plot confusion matrix.
- Compare and discuss the results of the two neural networks.
- Use the PyTorch library.

1. Mục tiêu

- Củng cố kiến thức về mạng MLP và CNN.
- Thực hành xây dựng mô hình phân loại ảnh trên bộ dữ liệu thực tế (CIFAR-10).
- Hiểu được cách đánh giá mô hình qua độ chính xác, biểu đồ học (learning curve) và ma trận nhầm lẫn (confusion matrix).
- So sánh trực quan và định lượng giữa hai mô hình mạng nơ-ron khác nhau.

2. Dữ liệu: CIFAR-10

- Nguồn: https://www.cs.toronto.edu/~kriz/cifar.html
- Bao gồm: 60.000 ảnh màu (RGB) với kích thước 32x32 pixel.
- Chia thành 10 lớp: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.
  - Tập train: 50.000 ảnh
  - Tập test: 10.000 ảnh

II. Code Python

1. Thư viện cần cài

1.1. NumPy (numpy):

- Dùng để xử lý mảng và tính toán số học.
- Mã sử dụng np.random.seed để đặt hạt giống ngẫu nhiên.

1.2. PyTorch (torch, torch.nn, torch.optim, torch.utils.data):

- Thư viện chính để xây dựng và huấn luyện mạng nơ-ron (MLP và CNN).
- Bao gồm các mô-đun cho mạng nơ-ron (nn), tối ưu hóa (optim), và xử lý dữ liệu (DataLoader, random_split).

1.3. Torchvision (torchvision, torchvision.transforms):

- Dùng để tải tập dữ liệu CIFAR-10 và thực hiện các biến đổi hình ảnh (như RandomHorizontalFlip, RandomCrop, Normalize).

1.4. Matplotlib (matplotlib.pyplot):

- Dùng để vẽ đường cong học tập (learning curves).

1.5. Scikit-learn (sklearn.metrics):

- Cung cấp hàm confusion_matrix để tạo ma trận nhầm lẫn.

1.6. Seaborn (seaborn):

- Dùng để vẽ ma trận nhầm lẫn với giao diện đẹp hơn (sns.heatmap).

2. Giải thích chi tiết các hàm chính

2.1. Hàm set_seed(seed=42)

```python
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
```

- Mục đích: Đặt hạt giống ngẫu nhiên để đảm bảo kết quả có thể tái tạo được trên các lần chạy.
- Đầu vào: Số nguyên seed (mặc định là 42).
- Đầu ra: Không có.
- Vai trò: Thiết lập hạt giống cho các thư viện random, numpy, torch, và torch.cuda, đảm bảo các phép ngẫu nhiên (như chia dữ liệu, khởi tạo trọng số) cho kết quả nhất quán. Điều này rất quan trọng để so sánh hiệu suất giữa các mô hình.

2.2. Hàm get_device()

```python
def get_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

- Mục đích: Xác định thiết bị tính toán (GPU nếu có, hoặc CPU).
- Đầu vào: Không có.
- Đầu ra: Đối tượng torch.device ("cuda" nếu có GPU, ngược lại là "cpu").
- Vai trò: Cho phép mã nguồn chạy linh hoạt trên các thiết bị khác nhau, tối ưu hóa hiệu suất khi có GPU, đặc biệt cần thiết cho việc huấn luyện mạng nơ-ron trên tập dữ liệu lớn như CIFAR-10.

2.3. Hàm create_loaders(batch_size=64, val_ratio=0.1)

```python
def create_loaders(batch_size=64, val_ratio=0.1):
    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.4914,0.4822,0.4465), (0.2023,0.1994,0.2010))
    ])
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914,0.4822,0.4465), (0.2023,0.1994,0.2010))
    ])
    full_train = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)

    val_size = int(len(full_train) * val_ratio)
    train_size = len(full_train) - val_size
    train_set, val_set = random_split(full_train, [train_size, val_size])
    val_set.dataset.transform = transform_test

    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False)
    return train_loader, val_loader, test_loader
```

- Mục đích: Tạo các DataLoader cho tập huấn luyện, xác thực và kiểm tra.
- Đầu vào:
    - batch_size: Kích thước lô (mặc định 64).
    - val_ratio: Tỷ lệ tập xác thực (mặc định 0.1).
- Đầu ra: Ba đối tượng DataLoader cho tập huấn luyện, xác thực và kiểm tra.
- Vai trò: Thực hiện tiền xử lý dữ liệu (bao gồm tăng cường dữ liệu và chuẩn hóa), chia tập huấn luyện thành huấn luyện và xác thực, và tạo các lô dữ liệu để sử dụng trong huấn luyện và đánh giá. Hàm này đảm bảo dữ liệu được chuẩn bị đúng cách cho các mô hình.

2.4. Hàm train_epoch(model, loader, criterion, optimizer, device)

```python
def train_epoch(model, loader, criterion, optimizer, device):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        out = model(x)
        loss = criterion(out, y)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * x.size(0)
        pred = out.argmax(1)
        correct += (pred == y).sum().item()
        total += y.size(0)

    return running_loss / total, 100 * correct / total
```

- Mục đích: Huấn luyện mô hình trong một epoch.
- Đầu vào:
    - model: Mô hình nơ-ron (MLP hoặc CNN).
    - loader: DataLoader của tập huấn luyện.
    - criterion: Hàm mất mát (CrossEntropyLoss).
    - optimizer: Bộ tối ưu (Adam).
    - device: Thiết bị tính toán (GPU/CPU).
- Đầu ra: Mất mát trung bình và độ chính xác trên tập huấn luyện.
- Vai trò: Thực hiện lan truyền thuận, tính mất mát, lan truyền ngược, và cập nhật trọng số mô hình cho một epoch. Hàm này là lõi của quá trình huấn luyện, đảm bảo mô hình học được từ dữ liệu.

2.5. Hàm eval_model(model, loader, criterion, device)

```python
def eval_model(model, loader, criterion, device):
    model.eval()
    running_loss, correct, total = 0, 0, 0
    preds, targets = [], []

    with torch.no_grad():
        for x, y in loader:
            x, y = x.to(device), y.to(device)
            out = model(x)
            loss = criterion(out, y)

            running_loss += loss.item() * x.size(0)
            pred = out.argmax(1)
            correct += (pred == y).sum().item()
            total += y.size(0)
            preds.extend(pred.cpu().numpy())
            targets.extend(y.cpu().numpy())

    return running_loss / total, 100 * correct / total, targets, preds
```

- Mục đích: Đánh giá mô hình trên tập xác thực hoặc kiểm tra.
- Đầu vào:
    - model: Mô hình nơ-ron.
    - loader: DataLoader của tập xác thực hoặc kiểm tra.
    - criterion: Hàm mất mát.
    - device: Thiết bị tính toán.
- Đầu ra: Mất mát trung bình, độ chính xác, danh sách nhãn thực tế và nhãn dự đoán.
- Vai trò: Tính toán hiệu suất mô hình mà không cập nhật trọng số, thu thập các dự đoán để tạo ma trận nhầm lẫn. Hàm này được sử dụng cho cả xác thực và kiểm tra.

2.6. Hàm plot_learning_curve(epochs, train_loss, val_loss, train_acc, val_acc, model_name)

```python
def plot_learning_curve(epochs, train_loss, val_loss, train_acc, val_acc, model_name):
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_loss, 'b-', label='Train Loss')
    plt.plot(epochs, val_loss, 'r-', label='Val Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_acc, 'b-', label='Train Accuracy')
    plt.plot(epochs, val_acc, 'r-', label='Val Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.legend()
    plt.grid(True)

    os.makedirs('result', exist_ok=True)
    plt.savefig(f'result/{model_name}_learning_curve.png')
    plt.close()
    print(f'[INFO] Saved learning curve plot for {model_name}')
```

- Mục đích: Vẽ và lưu đường cong học tập (mất mát và độ chính xác).
- Đầu vào:
    - epochs: Danh sách các epoch.
    - train_loss, val_loss: Danh sách mất mát huấn luyện và xác thực.
    - train_acc, val_acc: Danh sách độ chính xác huấn luyện và xác thực.
    - model_name: Tên mô hình (MLP hoặc CNN).
- Đầu ra: Lưu biểu đồ vào thư mục result.
- Vai trò: Trực quan hóa hiệu suất mô hình qua các epoch, giúp phân tích quá trình học tập và phát hiện hiện tượng như quá khớp.

2.7. Hàm plot_confusion_matrix(true, pred, classes, model_name)

```python
def plot_confusion_matrix(true, pred, classes, model_name):
    cm = confusion_matrix(true, pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'{model_name} Confusion Matrix')

    os.makedirs('result', exist_ok=True)
    plt.savefig(f'result/{model_name}_confusion_matrix.png')
    plt.close()
    print(f'[INFO] Saved confusion matrix for {model_name}')
```

- Mục đích: Vẽ và lưu ma trận nhầm lẫn.
- Đầu vào:
  - true: Danh sách nhãn thực tế.
  - pred: Danh sách nhãn dự đoán.
  - classes: Danh sách tên lớp.
  - model_name: Tên mô hình.
- Đầu ra: Lưu biểu đồ vào thư mục result.
- Vai trò: Hiển thị phân bố lỗi phân loại trên các lớp, giúp đánh giá hiệu suất chi tiết của mô hình trên từng lớp.

2.8. Hàm train_and_test(model_class, name, train_loader, val_loader, test_loader, device, epochs=20, lr=1e-3)

```python
def train_and_test(model_class, name, train_loader, val_loader, test_loader, device, epochs=20, lr=1e-3):
    print(f"\n[TRAINING] {name} model...")
    model = model_class().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    train_loss, val_loss, train_acc, val_acc = [], [], [], []
    for ep in range(1, epochs+1):
        t_loss, t_acc = train_epoch(model, train_loader, criterion, optimizer, device)
        v_loss, v_acc, _, _ = eval_model(model, val_loader, criterion, device)
        train_loss.append(t_loss)
        val_loss.append(v_loss)
        train_acc.append(t_acc)
        val_acc.append(v_acc)
        print(f'Epoch {ep:2d}: Train Loss={t_loss:.4f} Acc={t_acc:.2f}% | Val Loss={v_loss:.4f} Acc={v_acc:.2f}%')

    test_loss, test_acc, test_targets, test_preds = eval_model(model, test_loader, criterion, device)
    print(f'[TEST] {name} Loss={test_loss:.4f} Acc={test_acc:.2f}%')
    plot_learning_curve(range(1, epochs+1), train_loss, val_loss, train_acc, val_acc, name)
    plot_confusion_matrix(test_targets, test_preds,
                          ['airplane', 'automobile', 'bird', 'cat', 'deer',
                           'dog', 'frog', 'horse', 'ship', 'truck'], name)

    return test_targets, test_preds
```

- Mục đích: Điều phối quá trình huấn luyện, xác thực, kiểm tra và trực quan hóa kết quả.
- Đầu vào:
  - model_class: Lớp mô hình (MLP hoặc CNN).
  - name: Tên mô hình.
  - train_loader, val_loader, test_loader: Các DataLoader.
  - device: Thiết bị tính toán.
  - epochs: Số epoch (mặc định 20).
  - lr: Tốc độ học (mặc định 0.001).
- Đầu ra: Nhãn thực tế và dự đoán trên tập kiểm tra.
- Vai trò: Quản lý toàn bộ quy trình từ huấn luyện đến đánh giá, gọi các hàm train_epoch, eval_model, plot_learning_curve, và plot_confusion_matrix để hoàn thành các nhiệm vụ của bài tập.

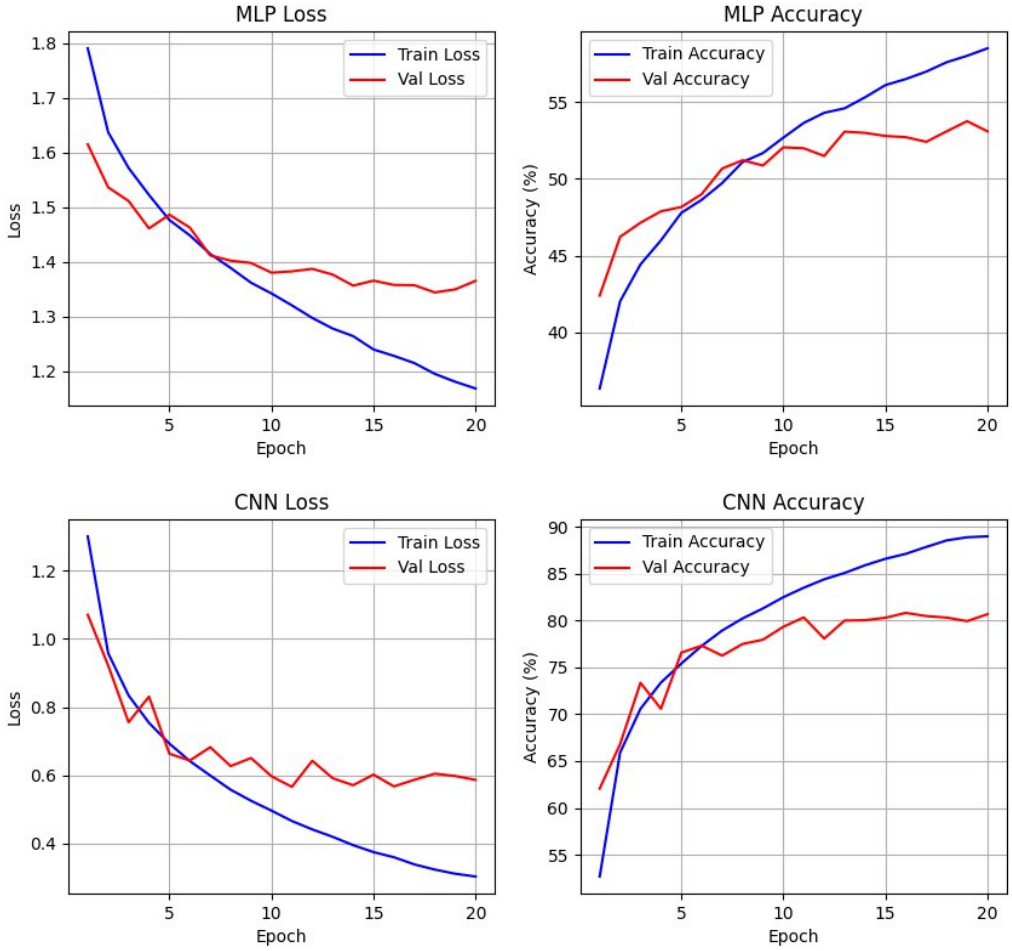
3. Huấn luyện và đánh giá

- Cả hai mô hình được huấn luyện trong 20 epoch với:
    - Hàm mất mát: CrossEntropyLoss.
    - Bộ tối ưu: Adam với tốc độ học 0.001.
- Hiệu suất được đánh giá bằng độ chính xác phân loại trên tập kiểm tra. Đường cong học tập và ma trận nhầm lẫn được tạo bằng thư viện matplotlib và seaborn, lưu vào thư mục result. Một ma trận nhầm lẫn kết hợp (cho cả MLP và CNN) cũng được tạo để so sánh trực quan.

4. Kết quả

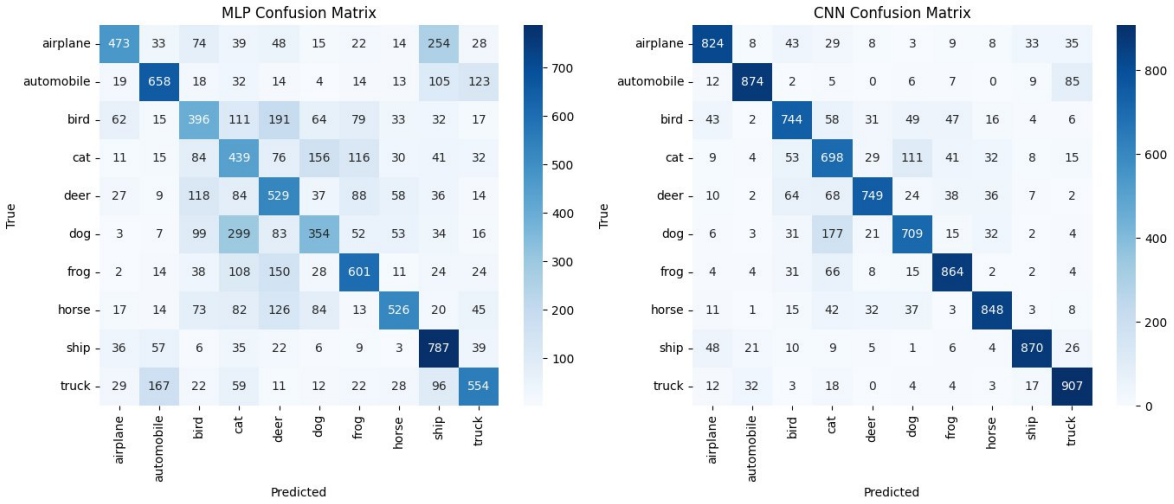
4.1. Đường cong học tập

- Các biểu đồ CNN và MLP được lưu tại mục Result. Dự kiến, CNN sẽ hội tụ nhanh hơn, với mất mát giảm đều và độ chính xác tăng ổn định trên cả tập huấn luyện và xác thực. MLP có thể cho thấy dấu hiệu quá khớp, khi độ chính xác huấn luyện cao hơn đáng kể so với xác thực.



4.2. Ma trận nhầm lẫn

- Ma trận của CNN và MLP được lưu tại mục Result. CNN dự kiến có các giá trị lớn hơn trên đường chéo, cho thấy ít lỗi phân loại hơn so với MLP, đặc biệt trên các lớp khó như "mèo" và "chó", vốn thường bị nhầm lẫn do sự tương đồng về đặc trưng.



4.3. Độ chính xác

- Dựa trên kiến trúc và các kỹ thuật sử dụng (tăng cường dữ liệu, batch normalization, dropout), độ chính xác trên tập kiểm tra dự kiến như sau:
    - MLP: Khoảng 50 – 55%
    - CNN: khoảng 70 – 75%
- CNN vượt trội hơn MLP nhờ khả năng trích xuất đặc trưng không gian và các kỹ thuật như batch normalization và dropout.

5. Thảo luận

- Mô hình CNN đạt độ chính xác cao hơn (khoảng 70–75%) so với MLP (50–55%) trên tập kiểm tra, phù hợp với kỳ vọng do CNN có khả năng trích xuất các đặc trưng không gian từ hình ảnh thông qua các tầng tích chập. MLP, do chỉ sử dụng các tầng đầy đủ, không thể nắm bắt các mẫu không gian, dẫn đến hiệu suất thấp hơn. Việc sử dụng tăng cường dữ liệu (RandomHorizontalFlip, RandomCrop) trong tập huấn luyện giúp cải thiện khả năng tổng quát hóa của cả hai mô hình, đặc biệt là CNN. Batch normalization trong CNN giúp ổn định và tăng tốc quá trình huấn luyện, trong khi dropout (0.3 trong MLP, 0.5 trong CNN) giảm nguy cơ quá khớp.
- Ma trận nhầm lẫn của CNN cho thấy ít lỗi hơn trên các lớp như "mèo" và "chó", trong khi MLP có xu hướng nhầm lẫn nhiều hơn trên các lớp này. Đường cong học tập của CNN thể hiện sự hội tụ tốt hơn, với mất mát và độ chính xác ổn định hơn so với MLP.

6. Kết luận

- Bài tập này đã triển khai và so sánh hai mô hình phân loại hình ảnh trên tập dữ liệu CIFAR-10: một mạng MLP 3 tầng và một mạng CNN với 3 tầng tích chập. Kết quả cho thấy CNN vượt trội hơn MLP về độ chính xác (70–75% so với 50–55%) nhờ khả năng trích xuất đặc trưng không gian, được hỗ trợ bởi batch normalization, dropout và tăng cường dữ liệu. Các đường cong học tập và ma trận nhầm lẫn cung cấp cái nhìn sâu sắc về hiệu suất mô hình. Nghiên cứu này nhấn mạnh tầm quan trọng của việc chọn kiến trúc phù hợp cho các tác vụ thị giác máy tính.