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

BÀI BÁO CÁO

BÀI TẬP LỚN: PYTHON

Giảng viên: KIM NGỌC BÁCH

Nhóm sinh viên thực hiện: Nguyễn Việt Anh - B23DCCE009

Lê Huy Hoàng - B23DCVT171

Hà Nội, Tháng 6/2025

Mục lục

1 Lời mở đầu

Thị giác máy tính là một lĩnh vực quan trọng trong trí tuệ nhân tạo, và bộ dữ liệu CIFAR-10 là một tiêu chuẩn phổ biến để đánh giá hiệu năng của các mô hình học máy. Bài tập này tập trung vào việc thiết kế, huấn luyện và đánh giá một Mạng Nơ-ron Tích chập (Convolutional Neural Network - CNN) sử dụng Python và thư viện học sâu PyTorch để giải quyết bài toán phân loại ảnh trên bộ dữ liệu CIFAR-10.

Các giai đoạn chính bao gồm:

- Chuẩn bị dữ liệu, kết hợp kỹ thuật tăng cường dữ liệu (Data Augmentation).
- Xây dựng kiến trúc mạng CNN với các tầng tích chập và kỹ thuật Dropout để hạn chế hiện tượng học quá khớp (overfitting).
- Huấn luyện mô hình với các kỹ thuật tối ưu hóa như Weight Decay và Early Stopping.
- Trực quan hóa kết quả thông qua biểu đồ đường cong học tập (Learning Curves) và ma trận nhầm lẫn (Confusion Matrix) để cung cấp cái nhìn sâu sắc về hiệu suất của mô hình.

Mục tiêu của bài tập là xây dựng một quy trình hoàn chỉnh cho bài toán phân loại ảnh, đồng thời áp dụng các kỹ thuật hiện đại nhằm nâng cao độ chính xác và khả năng tổng quát hóa của mô hình học sâu.

2 Chuẩn bị dữ liệu và Xây dựng mô hình

2.1 Tổng quan về mã nguồn và ý tưởng chung

Đoạn mã nguồn được phát triển nhằm thực hiện một quy trình học máy hoàn chỉnh để phân loại ảnh từ bộ dữ liệu CIFAR-10. Mô hình được sử dụng là Mạng Nơ-ron Tích chập (CNN) và được triển khai bằng thư viện PyTorch. Các bước chính trong quy trình bao gồm:

1. **Chuẩn bị dữ liệu**: Tải bộ dữ liệu CIFAR-10, áp dụng các phép biến đổi tăng cường dữ liệu (Data Augmentation) cho tập huấn luyện, chuẩn hóa dữ liệu và nạp vào các đối tượng `DataLoader` cho quá trình huấn luyện, kiểm định và kiểm thử.
2. **Xây dựng mô hình CNN**: Định nghĩa một kiến trúc mạng nơ-ron tích chập bao gồm ba tầng tích chập, các tầng gộp (pooling), hàm kích hoạt ReLU, và các lớp Dropout nhằm giảm thiểu hiện tượng overfitting.
3. **Huấn luyện mô hình**: Phát triển hàm huấn luyện để theo dõi giá trị hàm mất mát (loss) và độ chính xác (accuracy) trên cả ba tập dữ liệu qua từng epoch, tích hợp Weight Decay trong bộ tối ưu Adam và kỹ thuật Early Stopping.
4. **Đánh giá mô hình**: Tính toán loss, accuracy và thu thập nhãn dự đoán, nhãn thực tế để vẽ ma trận nhầm lẫn.

5. **Trực quan hóa**: Vẽ biểu đồ Learning Curves và Confusion Matrix để đánh giá trực quan hiệu suất của mô hình.

2.2 Thiết lập môi trường và thư viện

Chương trình bắt đầu bằng việc nhập các thư viện cần thiết:

```
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

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Giải thích các thư viện:

- `torch, torch.nn, torch.optim`: Các module cốt lõi của PyTorch, cung cấp cấu trúc dữ liệu tensor, các lớp xây dựng mạng nơ-ron, và thuật toán tối ưu hóa.
- `torchvision, torchvision.transforms`: Cung cấp bộ dữ liệu CIFAR-10 và các hàm biến đổi ảnh.
- `matplotlib.pyplot`: Thư viện để vẽ biểu đồ.
- `numpy`: Thư viện cho tính toán số học.
- `sklearn.metrics.confusion_matrix`: Hàm tính toán ma trận nhầm lẫn.
- `seaborn`: Thư viện trực quan hóa dữ liệu, hỗ trợ tạo heatmap.
- `device`: Chọn thiết bị tính toán (GPU nếu có, CPU nếu không).

2.3 Chuẩn bị dữ liệu CIFAR-10

2.3.1 Tải và tiền xử lý dữ liệu

Bộ dữ liệu CIFAR-10 được tải và tiền xử lý với các phép biến đổi:

```
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
```

```
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
    download=True, transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
    download=True, transform=transform_test)
```

Giải thích các phép biến đổi:

- `transforms.Compose`: Kết hợp nhiều phép biến đổi thành một chuỗi xử lý.
- `transform_train`: Áp dụng cho tập huấn luyện:
  - `RandomHorizontalFlip`: Lật ngang ảnh ngẫu nhiên.
  - `RandomRotation(10)`: Xoay ảnh ngẫu nhiên trong khoảng [-10, 10] độ.
  - `ToTensor`: Chuyển ảnh thành tensor PyTorch.
  - `Normalize`: Chuẩn hóa giá trị pixel về [-1.0, 1.0].
- `transform_test`: Không bao gồm Data Augmentation để đảm bảo đánh giá công bằng.
- `trainset, testset`: Tải dữ liệu CIFAR-10 với `train=True` cho huấn luyện và `train=False` cho kiểm thử.

2.3.2 Phân chia dữ liệu và tạo DataLoader

Tập huấn luyện được chia thành 80% dữ liệu huấn luyện và 20% dữ liệu kiểm định:

```
train_size = int(0.8 * len(trainset))
val_size = len(trainset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(trainset, [
    train_size, val_size])

trainloader = torch.utils.data.DataLoader(train_dataset, batch_size
    =100, shuffle=True)
valloader = torch.utils.data.DataLoader(val_dataset, batch_size=100,
    shuffle=False)
testloader = torch.utils.data.DataLoader(testset, batch_size=100,
    shuffle=False)
```

```

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', '
    horse', 'ship', 'truck')
```

Giải thích:

- `random_split`: Chia ngẫu nhiên tập huấn luyện thành hai tập con.
- `DataLoader`: Cung cấp dữ liệu theo batch với `batch_size=100`.
- `shuffle=True`: Xáo trộn dữ liệu huấn luyện để tăng tính ngẫu nhiên.
- `classes`: Tuple chứa tên 10 lớp đối tượng trong CIFAR-10.

2.4 Xây dựng mô hình Mạng Nơ-ron Tích chập (CNN)

2.4.1 Định nghĩa kiến trúc mô hình

Lớp `CNN` kế thừa từ `nn.Module`:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)  #
            Input: 3x32x32, Output: 32x32x32
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1) #
            Output: 64x16x16 (after pooling)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1) #
            Output: 128x8x8 (after pooling)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout = nn.Dropout(0.25)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 10)
        self.dropout_fc = nn.Dropout(0.5)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)  # 32x16x16
        x = self.dropout(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)  # 64x8x8
        x = self.dropout(x)
        x = self.relu(self.conv3(x))
```

```
        x = self.pool(x)  # 128x4x4
        x = self.dropout(x)
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.dropout_fc(x)
        x = self.fc2(x)
        return x
```

Giải thích:

- **Tầng tích chập**:
  - `conv1`: Nhận đầu vào 3x32x32, xuất ra 32x32x32 với 32 bộ lọc.
  - `conv2`: Xuất ra 64x16x16 sau pooling.
  - `conv3`: Xuất ra 128x8x8 sau pooling.
  - `padding=1`: Giữ kích thước không gian sau tích chập.
- `pool`: MaxPooling giảm kích thước không gian.
- `ReLU`: Hàm kích hoạt áp dụng sau mỗi tầng tích chập và fully connected.
- `Dropout`: Tỷ lệ 0.25 cho tầng tích chập và 0.5 cho fully connected.
- `fc1, fc2`: Tầng fully connected với đầu ra 10 lớp.

2.4.2 Vai trò của Dropout

Dropout ngẫu nhiên tắt một số nơ-ron trong quá trình huấn luyện, giúp mô hình học các đặc trưng mạnh mẽ hơn và giảm hiện tượng overfitting.

3 Huấn luyện và Đánh giá mô hình

3.1 Hàm huấn luyện mô hình (train_model)

3.1.1 Mục đích

- Huấn luyện mô hình CNN sử dụng `trainloader`.
- Đánh giá hiệu suất trên `valloader` và `testloader` sau mỗi epoch.
- Lưu trữ lịch sử loss và accuracy.
- Áp dụng Early Stopping để ngăn chặn overfitting.
- Trả về mô hình với trọng số tốt nhất và lịch sử huấn luyện.

3.1.2 Tham số đầu vào

```
def train_model(model, trainloader, valloader, testloader, epochs=20,
    patience=3)
```

- `model`: Mô hình CNN.
- `trainloader, valloader, testloader`: Các DataLoader.
- `epochs=20`: Số epoch tối đa.
- `patience=3`: Số epoch chờ trước khi dừng sớm.

3.1.3 Cơ chế hoạt động

```
def train_model(model, trainloader, valloader, testloader, epochs=20,
    patience=3):
     model = model.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay
        =1e-4)

     train_losses, val_losses, test_losses = [], [], []
     train_accs, val_accs, test_accs = [], [], []

     best_val_loss = float('inf')
     epochs_no_improve = 0
     best_model_state = None

     for epoch in range(epochs):
         # Training
         model.train()
         running_loss, correct, total = 0.0, 0, 0
         for inputs, labels in trainloader:
             inputs, labels = inputs.to(device), labels.to(device)
             optimizer.zero_grad()
             outputs = model(inputs)
             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_losses.append(running_loss / len(trainloader))
        train_accs.append(100 * correct / total)

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

        val_losses.append(val_loss / len(valloader))
        val_accs.append(100 * correct / total)

        # Test
        test_loss, correct, total = 0.0, 0, 0
        with torch.no_grad():
            for inputs, labels in testloader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        test_losses.append(test_loss / len(testloader))
        test_accs.append(100 * correct / total)

        # Print the results
        print(f'Epoch {epoch+1}/{epochs}, Train Loss: {train_losses
            [-1]:.4f}, Train Acc: {train_accs[-1]:.2f}%, '
              f'Val Loss: {val_losses[-1]:.4f}, Val Acc: {val_accs
                  [-1]:.2f}%, '
```

```
67                 f'Test Loss: {test_losses[-1]:.4f}, Test Acc: {test_accs
                       [-1]:.2f}%')
68
69         # Early Stopping
70         if val_losses[-1] < best_val_loss:
71             best_val_loss = val_losses[-1]
72             epochs_no_improve = 0
73             best_model_state = model.state_dict()
74         else:
75             epochs_no_improve += 1
76
77         if epochs_no_improve >= patience:
78             print(f'Early stopping at epoch {epoch+1} due to no
                       improvement in Val Loss.')
79             model.load_state_dict(best_model_state)
80             break
81
82     return train_losses, val_losses, test_losses, train_accs, val_accs
           , test_accs
```

Khởi tạo:

- Mô hình được chuyển lên thiết bị tính toán (GPU/CPU).
- Hàm mất mát `nn.CrossEntropyLoss` cho bài toán phân loại đa lớp.
- Thuật toán tối ưu Adam với `lr=0.001`, `weight_decay=1e-4`.
- Khởi tạo danh sách lưu trữ loss và accuracy.
- Biến cho Early Stopping: `best_val_loss`, `epochs_no_improve`, `best_model_state`.

Vòng lặp epoch:

- **Huấn luyện**: Chuyển mô hình sang chế độ `model.train()`, thực hiện forward pass, tính loss, lan truyền ngược, cập nhật trọng số, tích lũy loss và accuracy.
- **Kiểm định**: Chuyển sang `model.eval()`, vô hiệu hóa gradient, tính loss và accuracy trên `valloader`.
- **Kiểm thử**: Tương tự kiểm định, trên `testloader`.
- **In kết quả**: Hiển thị loss và accuracy sau mỗi epoch.
- **Early Stopping**: So sánh `val_loss` với `best_val_loss`, lưu trạng thái mô hình tốt nhất, dừng sớm nếu không cải thiện.

3.1.4 Tối ưu hóa và Regularization

- **Optimizer**: Thuật toán Adam với `lr=0.001`, `weight_decay=1e-4`.
- **Dropout**: Tích hợp trong kiến trúc CNN.
- **Data Augmentation**: Lật ngang, xoay ngẫu nhiên cho tập huấn luyện.

3.1.5 Kỹ thuật Early Stopping

Theo dõi validation loss, dừng huấn luyện nếu không cải thiện trong `patience` epoch, chọn mô hình với `best_val_loss`.

3.1.6 Kết quả trả về

Hàm trả về 6 danh sách chứa lịch sử loss và accuracy của ba tập qua các epoch.

3.2 Hàm đánh giá mô hình (evaluate_model)

3.2.1 Mục đích

- Đánh giá hiệu suất mô hình trên một `DataLoader` cụ thể.
- Tính loss trung bình, accuracy tổng thể, thu thập nhãn thực tế/dự đoán.

3.2.2 Tham số đầu vào

```
def evaluate_model(model, dataloader, set_name="Test")
```

- `model`: Mô hình CNN đã huấn luyện.
- `dataloader`: DataLoader cung cấp dữ liệu.
- `set_name`: Tên tập dữ liệu ("Train", "Validation", "Test").

3.2.3 Cơ chế hoạt động

```
def evaluate_model(model, dataloader, set_name="Test"):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    model.eval()
    running_loss, correct, total = 0.0, 0, 0
    y_true, y_pred = [], []

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
```

```
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(predicted.cpu().numpy())

    avg_loss = running_loss / len(dataloader)
    accuracy = 100 * correct / total
    print(f'{set_name} Loss: {avg_loss:.4f}, {set_name} Accuracy: {
        accuracy:.2f}%')

    return y_true, y_pred, avg_loss, accuracy
```

Cơ chế hoạt động:

1. Chuyển mô hình lên thiết bị tính toán và đặt chế độ `model.eval()`.
2. Khởi tạo danh sách `y_true`, `y_pred` và biến tích lũy loss, accuracy.
3. Vô hiệu hóa gradient với `torch.no_grad()`.
4. Duyệt qua từng batch:
   - Thực hiện forward pass, tính loss.
   - Lưu nhãn thực tế và dự đoán.
   - Tích lũy loss và số dự đoán đúng.
5. Tính loss trung bình và accuracy tổng thể.
6. In kết quả và trả về `y_true`, `y_pred`, `avg_loss`, `accuracy`.

3.2.4 Kết quả trả về

- `y_true`: Danh sách nhãn thực tế.
- `y_pred`: Danh sách nhãn dự đoán.
- `avg_loss`: Loss trung bình.
- `accuracy`: Độ chính xác (%).

4 Trực quan hóa kết quả

4.1 Hàm vẽ biểu đồ Learning Curves (plot_learning_curves)

4.1.1 Mục đích

Vẽ biểu đồ thể hiện sự thay đổi của loss và accuracy trên các tập train, validation, test qua các epoch.

4.1.2 Tham số đầu vào

```
def plot_learning_curves(train_losses, val_losses, test_losses,
    train_accs, val_accs, test_accs, title)
```

- `train_losses, val_losses, test_losses`: Danh sách loss qua các epoch.
- `train_accs, val_accs, test_accs`: Danh sách accuracy qua các epoch.
- `title`: Tiêu đề biểu đồ.

4.1.3 Cơ chế hoạt động

1. Tạo figure với hai subplot (loss và accuracy).
2. Vẽ đường cong loss (train: xanh dương, validation: cam, test: xanh lá).
3. Vẽ đường cong accuracy tương tự.
4. Đặt tiêu đề, nhãn trục, chú thích.
5. Lưu figure vào `cnn_learning_curves.png`.

4.2 Hàm vẽ Confusion Matrix (plot_confusion_matrix)

4.2.1 Mục đích

Vẽ ma trận nhầm lẫn để đánh giá hiệu suất phân loại.

4.2.2 Tham số đầu vào

```
def plot_confusion_matrix(y_true, y_pred, title, filename)
```

- `y_true`: Nhãn thực tế.
- `y_pred`: Nhãn dự đoán.
- `title`: Tiêu đề biểu đồ.
- `filename`: Tên file lưu ảnh.

4.2.3 Cơ chế hoạt động

1. Tính ma trận nhầm lẫn bằng `confusion_matrix`.
2. Tạo figure và vẽ heatmap bằng `seaborn.heatmap`.
3. Đặt nhãn trục bằng `classes`.
4. Lưu biểu đồ vào file `filename`.

5 Thực thi và Kết quả

5.1 Khởi tạo và huấn luyện mô hình CNN

Đầu tiên, một đối tượng của lớp `CNN` được khởi tạo. Sau đó, hàm `train_model` được gọi để huấn luyện mô hình này. Quá trình huấn luyện được thiết lập để chạy tối đa 20 epoch, với `patience` (số epoch chờ đợi sự cải thiện trên tập kiểm định trước khi dừng sớm) là 3.

```
cnn = CNN()
cnn_train_losses, cnn_val_losses, cnn_test_losses, cnn_train_accs,
    cnn_val_accs, cnn_test_accs = train_model(cnn, trainloader,
    valloader, testloader, epochs=20, patience=3)
```

Lịch sử của giá trị hàm mất mát (loss) và độ chính xác (accuracy) qua các epoch được lưu trữ lại cho cả ba tập dữ liệu: huấn luyện, kiểm định và kiểm thử.

5.2 Đánh giá mô hình

Sau khi quá trình huấn luyện hoàn tất (có thể do đạt đủ số epoch hoặc do Early Stopping), mô hình thu được sẽ được đánh giá một cách cẩn thận trên cả ba tập dữ liệu: huấn luyện, kiểm định và kiểm thử.

```
cnn_train_y_true, cnn_train_y_pred, cnn_train_loss, cnn_train_acc =
    evaluate_model(cnn, trainloader, "Train")
cnn_val_y_true, cnn_val_y_pred, cnn_val_loss, cnn_val_acc =
    evaluate_model(cnn, valloader, "Validation")
cnn_test_y_true, cnn_test_y_pred, cnn_test_loss, cnn_test_acc =
    evaluate_model(cnn, testloader, "Test")
print(f"\nFinal CNN Results:")
print(f"Train Loss: {cnn_train_loss:.4f}, Train Accuracy: {
    cnn_train_acc:.2f}%")
print(f"Validation Loss: {cnn_val_loss:.4f}, Validation Accuracy: {
    cnn_val_acc:.2f}%")
print(f"Test Loss: {cnn_test_loss:.4f}, Test Accuracy: {cnn_test_acc
    :.2f}%")
```

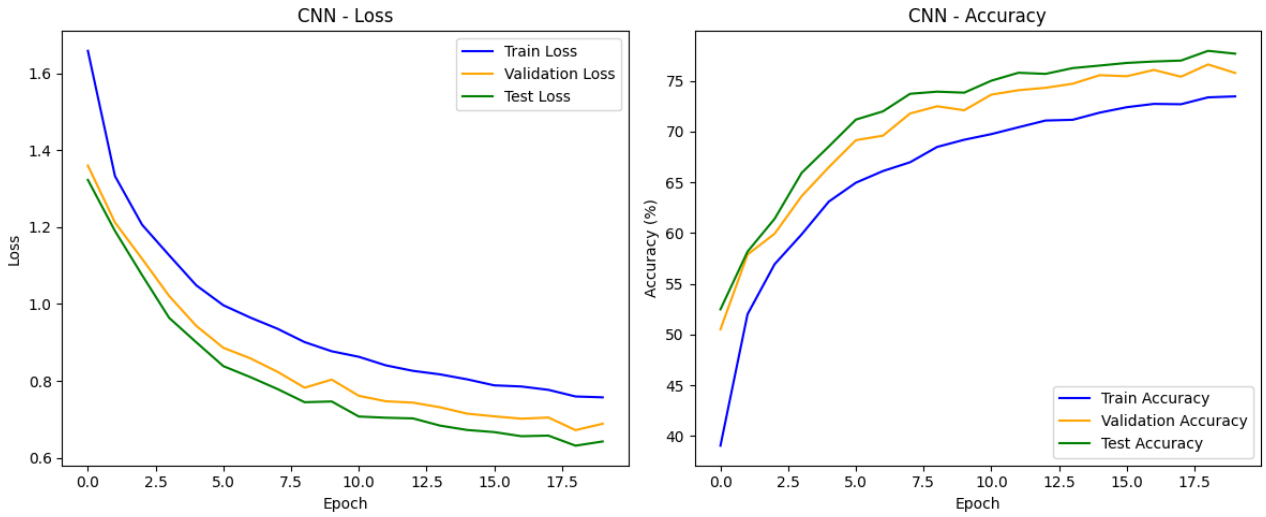
Hàm `evaluate_model` được sử dụng để tính toán loss và accuracy trên từng tập dữ liệu. Các kết quả cuối cùng về loss và accuracy của mô hình trên ba tập này được in ra màn hình, cung cấp một cái nhìn tổng quan và định lượng về hiệu suất của mô hình.

5.3 Trực quan hóa Learning Curves

Để trực quan hóa quá trình huấn luyện và đánh giá sự hội tụ của mô hình, biểu đồ Learning Curves được tạo ra bằng cách gọi hàm `plot_learning_curves`.

```
plot_learning_curves(cnn_train_losses, cnn_val_losses, cnn_test_losses
    , cnn_train_accs, cnn_val_accs, cnn_test_accs, 'CNN')
```

Hàm này sẽ tạo ra một file ảnh có tên `cnn_learning_curves.png`, chứa hai biểu đồ con.



Hình 1: cnn_learning_curves.png

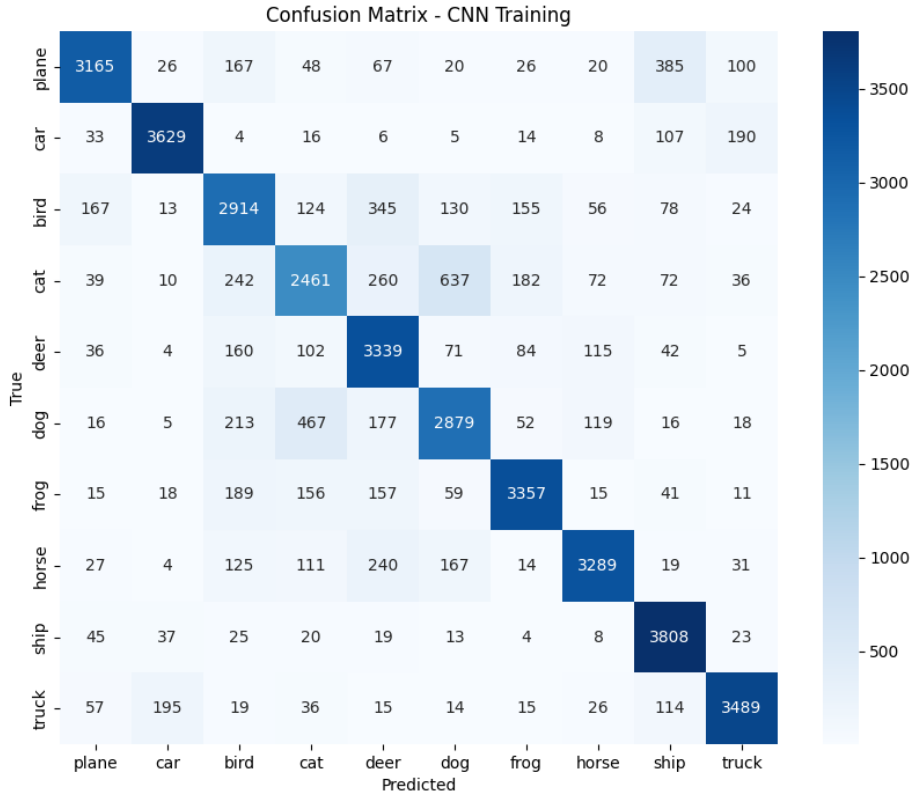
5.4 Trực quan hóa Confusion Matrices

Để phân tích chi tiết hơn về khả năng phân loại của mô hình đối với từng lớp cụ thể, ma trận nhầm lẫn (Confusion Matrix) được vẽ cho mỗi tập dữ liệu (huấn luyện, kiểm định, và kiểm thử) bằng cách sử dụng hàm `plot_confusion_matrix`.

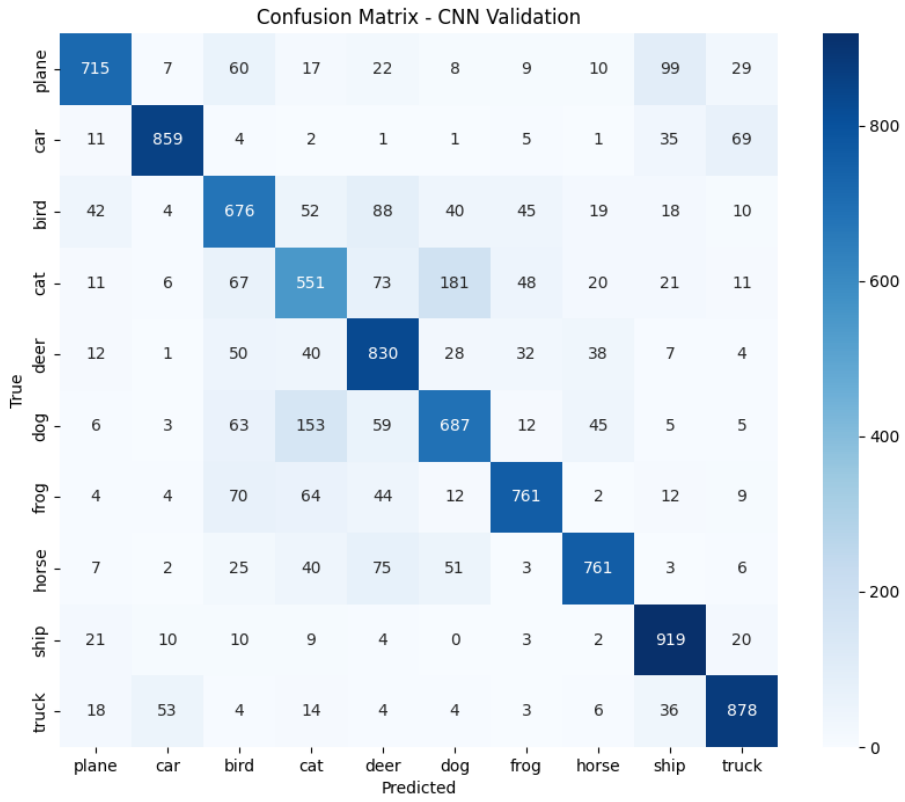
```
plot_confusion_matrix(cnn_train_y_true, cnn_train_y_pred, 'CNN
    Training', 'cnn_confusion_matrix_train.png')
plot_confusion_matrix(cnn_val_y_true, cnn_val_y_pred, 'CNN Validation'
    , 'cnn_confusion_matrix_val.png')
plot_confusion_matrix(cnn_test_y_true, cnn_test_y_pred, 'CNN Test', '
    cnn_confusion_matrix_test.png')
```

Thao tác này sẽ tạo ra ba file ảnh riêng biệt: Mỗi ma trận nhầm lẫn sẽ hiển thị một cách trực quan số lượng các mẫu được dự đoán đúng (nằm trên đường chéo chính) và số lượng các mẫu bị dự đoán sai

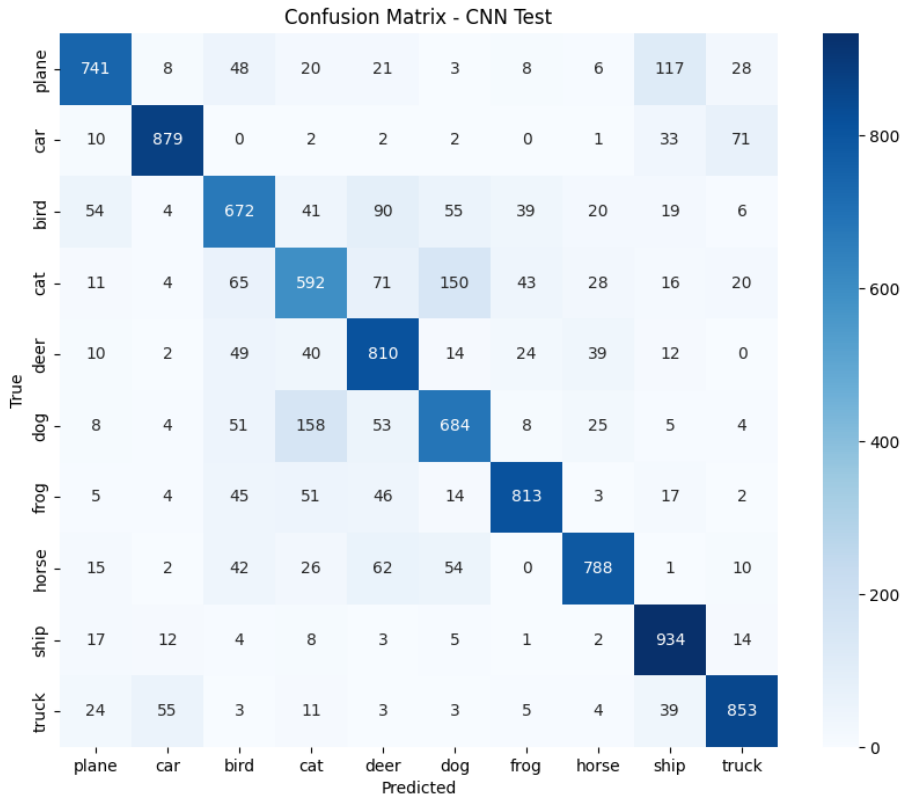
(nằm ngoài đường chéo chính) cho từng lớp đối tượng.



Hình 2: cnn_confusion_matrix_train.png



Hình 3: cnn_confusion_matrix_val.png



Hình 4: cnn_confusion_matrix_test.png

5.5 Phân tích kết quả

Việc phân tích kết quả dựa trên các thông tin thu thập được từ log huấn luyện, các chỉ số hiệu suất cuối cùng, biểu đồ Learning Curves và các ma trận nhầm lẫn:

1. **Log huấn luyện**: Quan sát sự thay đổi của loss và accuracy qua các epoch để đánh giá quá trình hội tụ và hiệu quả của Early Stopping.
2. **Kết quả cuối cùng**: So sánh loss và accuracy trên ba tập. Độ chính xác trên tập test là thước đo quan trọng nhất về khả năng tổng quát hóa.
3. **Learning Curves**:
   - Kiểm tra sự hội tụ: Loss giảm và accuracy tăng đều đặn.
   - Phát hiện overfitting: Nếu train accuracy cao hơn nhiều so với val/test accuracy, hoặc val loss tăng trong khi train loss giảm, đó là dấu hiệu overfitting.
   - Đánh giá Early Stopping: Xác định xem việc dừng sớm có phù hợp không.
4. **Confusion Matrices**:
   - Xác định các lớp được phân loại tốt (giá trị lớn trên đường chéo).
   - Phát hiện các cặp lớp hay bị nhầm lẫn (giá trị lớn ngoài đường chéo).

Phân tích này cung cấp cái nhìn toàn diện về hiệu suất của mô hình CNN, giúp xác định điểm mạnh và điểm yếu trong việc phân loại các lớp của CIFAR-10.

6 Kết luận

Bài tập này đã xây dựng và triển khai một mô hình CNN cho bài toán phân loại ảnh trên bộ dữ liệu CIFAR-10 sử dụng PyTorch. Các điểm nổi bật:

1. **Xử lý dữ liệu**: Áp dụng Data Augmentation hiệu quả để tăng tính đa dạng của tập huấn luyện.
2. **Kiến trúc CNN**: Xây dựng mô hình với ba tầng tích chập, tích hợp Dropout để chống overfitting.
3. **Huấn luyện nâng cao**: Sử dụng Adam optimizer, Weight Decay và Early Stopping để tối ưu hóa hiệu suất.
4. **Đánh giá toàn diện**: Đánh giá mô hình trên các tập train, validation và test, cung cấp số liệu loss và accuracy.
5. **Trực quan hóa**: Tạo Learning Curves và Confusion Matrices để phân tích chi tiết hiệu suất.

Thông qua bài tập này, các kỹ thuật học sâu hiện đại đã được áp dụng một cách thành công, từ đó xây dựng nên một quy trình học máy hoàn chỉnh và hiệu quả cho bài toán phân loại ảnh. Các kết quả phân

tích thu được không chỉ đánh giá được hiệu năng của mô hình mà còn giúp hiểu rõ hơn về khả năng của mô hình CNN trong bối cảnh cụ thể của bài toán này.