HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

NGÔN NGỮ PYTHON

Giảng viên hướng dẫn : KIM NGỌC BÁCH
Họ và tên sinh viên : TRẦN ĐÌNH HIẾU
Mã sinh viên : B23DCE034
Lớp : D23CQCE04-B

Hà Nội – 2023

LỜI NÓI ĐẦU

Báo cáo này được thực hiện trong khuôn khổ môn học Python do thầy Kim Ngọc Bách hướng dẫn. Nội dung báo cáo tập trung vào việc áp dụng các kiến thức đã học để giải quyết một bài toán cụ thể bằng ngôn ngữ lập trình Python.

Trong quá trình thực hiện, nhóm đã cố gắng vận dụng kiến thức lập trình, tư duy logic và kỹ năng xử lý dữ liệu để hoàn thành yêu cầu của bài. Nhóm xin cảm ơn thầy Kim Ngọc Bách đã hướng dẫn và hỗ trợ trong suốt quá trình học và làm báo cáo.

Nhóm cũng mong nhận được góp ý để hoàn thiện hơn trong các bài làm sau.

1. Import thư viện

```
# ==================== IMPORT THƯ VIỆN ====================
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import random_split, DataLoader
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np
import os
```

Chức năng:

- `torch`, `torch.nn`: Thư viện chính để xây dựng và huấn luyện mô hình deep learning.
- `torchvision`: Hỗ trợ tải và xử lý dữ liệu ảnh.
- `matplotlib.pyplot`: Vẽ biểu đồ (loss, accuracy).
- `sklearn.metrics`: Tính ma trận nhầm lẫn để đánh giá mô hình.
- `numpy`, `os`: Hỗ trợ tính toán số học và thao tác hệ thống.

---

2. Chuẩn bị dữ liệu

```
# ==================== CHUẨN BỊ DỮ LIỆU ====================
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    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))
])
```

Mục tiêu: Tiền xử lý dữ liệu đầu vào.

- `RandomHorizontalFlip()`: Lật ảnh ngẫu nhiên theo chiều ngang.
- `RandomCrop(32, padding=4)`: Cắt ảnh ngẫu nhiên với đệm để tăng độ đa dạng dữ liệu.
- `ToTensor()`: Chuyển ảnh từ [0,255] sang tensor có giá trị [0,1].
- `Normalize((0.5,...), (0.5,...))`: Chuẩn hóa ảnh về trung bình 0 và độ lệch chuẩn 1.

```
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)



train_size = int(0.8 * len(trainset))

val_size = len(trainset) - train_size

train_data, val_data = random_split(trainset, [train_size, val_size])



train_loader = DataLoader(train_data, batch_size=64, shuffle=True)

val_loader = DataLoader(val_data, batch_size=64, shuffle=False)

test_loader = DataLoader(testset, batch_size=64, shuffle=False)



classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
```

Ý nghĩa:

- Tải dữ liệu CIFAR-10 (gồm 60,000 ảnh 32x32 thuộc 10 lớp).
- Chia dữ liệu train thành 80% train và 20% validation.
- Tạo các `DataLoader` để nạp dữ liệu theo batch.

---

3. Mô hình MLP (Multi-Layer Perceptron)

```
# ==================== MÔ HÌNH MLP ====================
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32*32*3, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        return self.model(x)
```

Cấu trúc:

- `Flatten`: Biến ảnh 32x32x3 → vector 3072.
- 3 tầng `Linear`:
    - 3072 → 512
    - 512 → 256
    - 256 → 10 (số lớp)

- `BatchNorm`, `ReLU`, `Dropout`: Tăng độ ổn định, tránh overfitting.

---

4. Mô hình CNN (Convolutional Neural Network)

```
# ==================== MÔ HÌNH CNN ====================
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Flatten(),
            nn.Linear(128*4*4, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        return self.net(x)
```

Cấu trúc:

- 3 block convolution:

    - Conv2D → BatchNorm → ReLU → MaxPool2d
    - Tăng số kênh: 3→32→64→128
    - Giảm kích thước ảnh từ 32×32 → 4×4
- Flatten và 2 tầng `Linear`: 2048 → 256 → 10

Lý do: CNN khai thác cấu trúc không gian ảnh tốt hơn MLP.

---

5. Hàm huấn luyện mô hình

```
# ====================== HÀM TRAIN MODEL ======================
def train_model(model, train_loader, val_loader, epochs=50, lr=0.001, patience=5, model_name='model.pth'):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

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

    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []

    best_val_acc = 0
    patience_counter = 0

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct, total = 0, 0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
```

```
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == targets).sum().item()
            total += targets.size(0)

        train_losses.append(running_loss / len(train_loader))
        train_accuracies.append(correct / total)

        model.eval()
        val_loss = 0.0
        correct, total = 0, 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, targets)
                val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                correct += (predicted == targets).sum().item()
                total += targets.size(0)

        val_losses.append(val_loss / len(val_loader))
        val_accuracies.append(correct / total)

        print(f"Epoch {epoch+1}: Train Loss = {train_losses[-1]:.4f},
Train Acc = {train_accuracies[-1]*100:.2f}%, "
              f"Val Loss = {val_losses[-1]:.4f}, Val Acc =
{val_accuracies[-1]*100:.2f}%")

        # Early Stopping
        if val_accuracies[-1] > best_val_acc:
            best_val_acc = val_accuracies[-1]
            patience_counter = 0
            torch.save(model.state_dict(), model_name)
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered.")
                break

    # Load best model
    model.load_state_dict(torch.load(model_name))
    return train_losses, val_losses, train_accuracies, val_accuracies
```

Thành phần chính:

- **optimizer**: Dùng Adam với learning rate 0.001.
- **loss**: CrossEntropyLoss (dùng cho phân loại).
- **Vòng lặp epoch**:
    - Giai đoạn train: cập nhật trọng số bằng `loss.backward()` và `optimizer.step()`.
    - Giai đoạn validation: đánh giá hiệu suất mô hình không cập nhật trọng số.
- **Early Stopping**: Nếu accuracy validation không tăng sau `patience` lần thì dừng sớm để tránh overfitting.

Kết quả trả về:

- loss và accuracy theo từng epoch để vẽ biểu đồ.

---

6. Vẽ learning curves

```
# ==================== VẼ LEARNING CURVES ====================
def plot_learning_curves(train_losses, val_losses, train_accs,
val_accs, title):
    plt.figure(figsize=(12, 5))

    # Loss curve
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss', color='blue')
    plt.plot(val_losses, label='Val Loss', color='orange')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'{title} - Loss Curve')
    plt.legend()
    plt.grid(True)

    # Accuracy curve
```

```
    plt.subplot(1, 2, 2)
    plt.plot(train_accs, label='Train Accuracy', color='blue')
    plt.plot(val_accs, label='Val Accuracy', color='orange')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title(f'{title} - Accuracy Curve')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

- **Loss Curve**: Theo dõi độ hội tụ của mô hình.
- **Accuracy Curve**: Theo dõi khả năng học của mô hình.
- Dùng `plt.subplot` để hiển thị 2 biểu đồ cạnh nhau.

---

7. Vẽ confusion matrix

```
# ==================== VẼ CONFUSION MATRIX ====================
def plot_confusion(model, data_loader, classes):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model.eval()
    model = model.to(device)
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu())
            all_labels.extend(labels)

    cm = confusion_matrix(all_labels, all_preds)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=classes)
```

```
    disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()
```

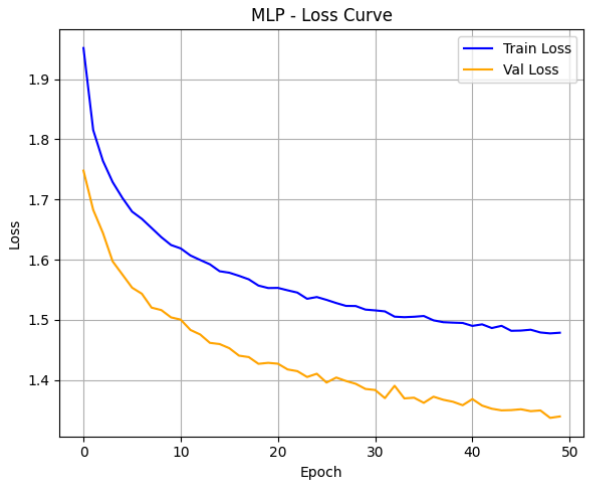
- Tính toán và hiển thị ma trận nhầm lẫn (Confusion Matrix).
- Cho biết mô hình dễ nhầm lẫn những lớp nào (ví dụ: cat vs dog).
- Dùng `sklearn.metrics.confusion_matrix` và `ConfusionMatrixDisplay`.

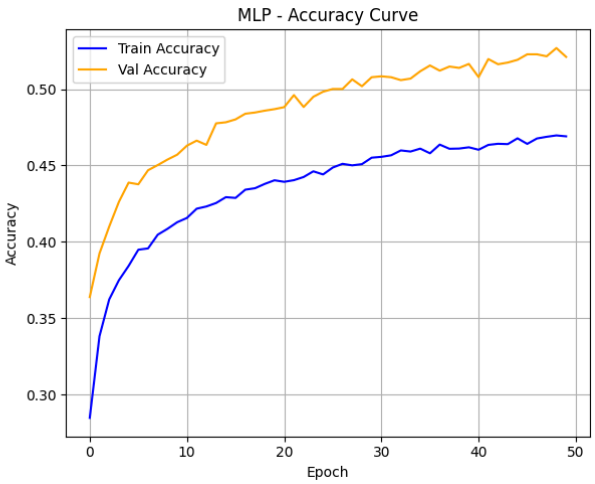
---

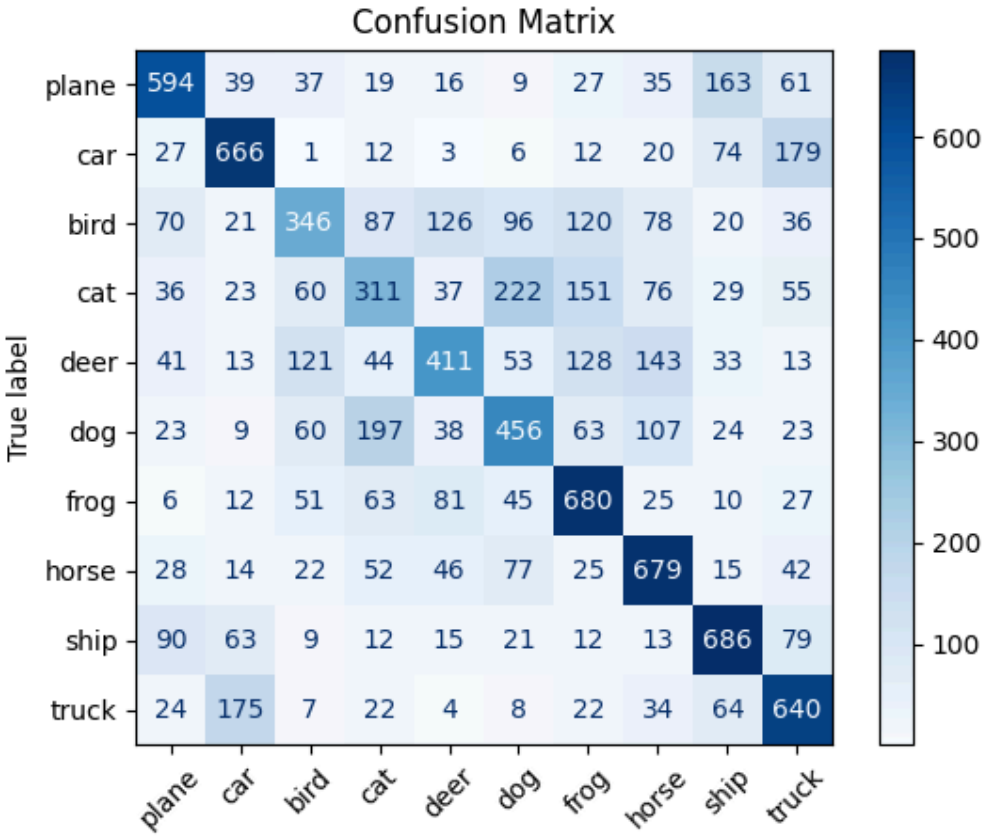
8. Chạy và đánh giá mô hình MLP

```
# ==================== CHẠY MLP ====================
print("\n--- Training MLP ---")
mlp = MLP()
mlp_train_losses, mlp_val_losses, mlp_train_accs, mlp_val_accs =
train_model(mlp, train_loader, val_loader, model_name='mlp_model.pth')
plot_learning_curves(mlp_train_losses, mlp_val_losses, mlp_train_accs,
mlp_val_accs, "MLP")
plot_confusion(mlp, test_loader, classes)
```

- Khởi tạo và huấn luyện MLP.
- Vẽ biểu đồ loss, accuracy qua các epoch.
- Đánh giá kết quả trên tập test.







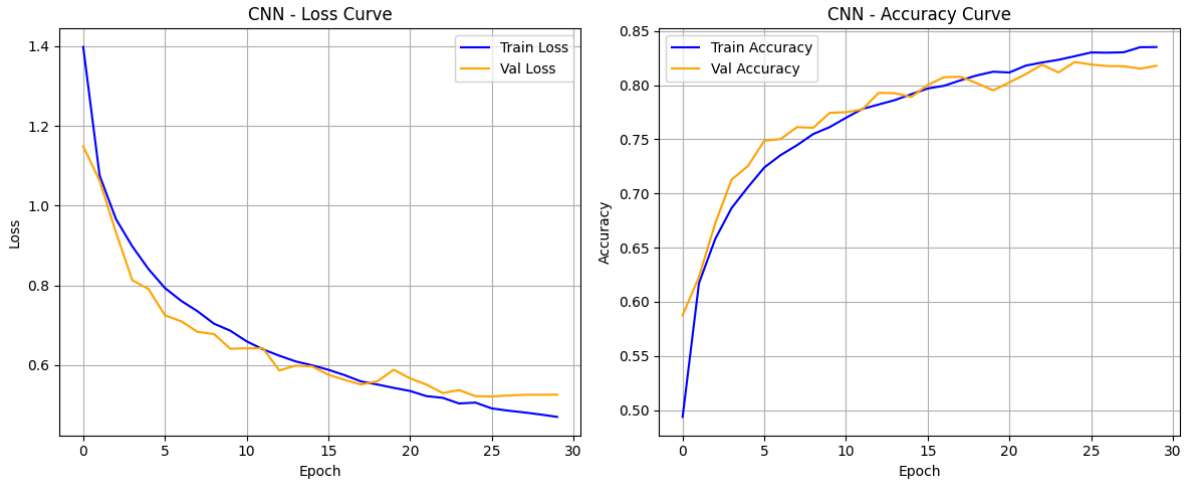
---

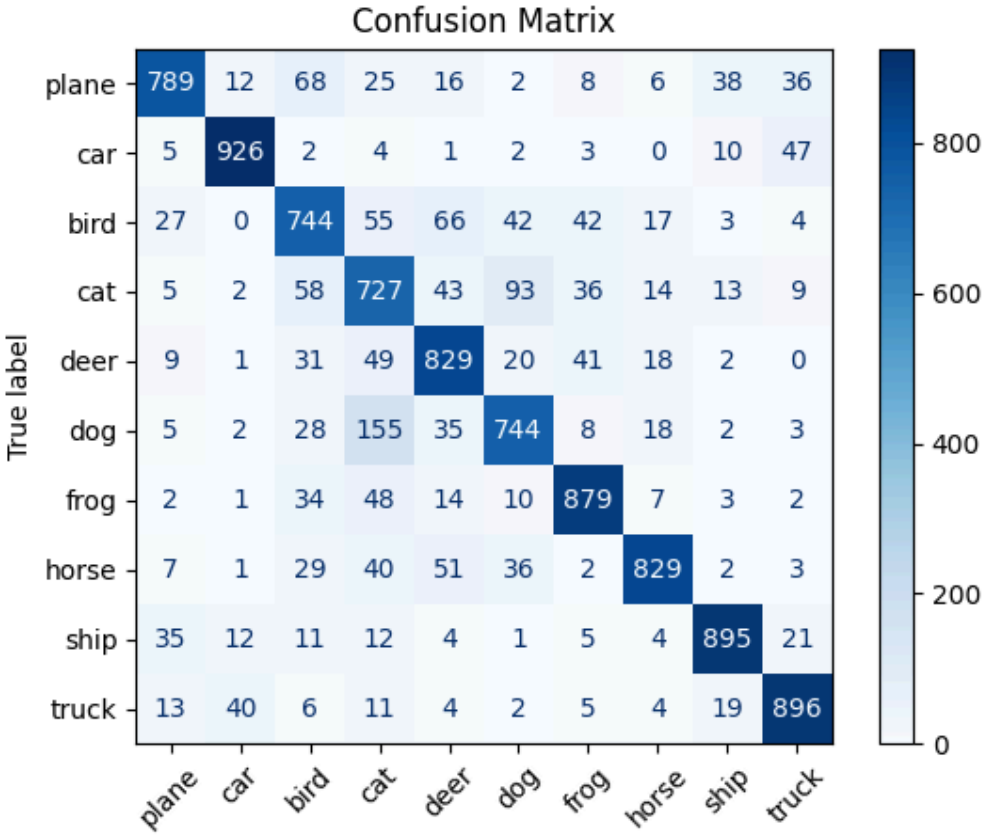
9. Chạy và đánh giá mô hình CNN

```
# ===================== CHẠY CNN =====================
print("\n--- Training CNN ---")
cnn = CNN()
```

```
cnn_train_losses, cnn_val_losses, cnn_train_accs, cnn_val_accs =
train_model(cnn, train_loader, val_loader, model_name='cnn_model.pth')
plot_learning_curves(cnn_train_losses, cnn_val_losses, cnn_train_accs,
cnn_val_accs, "CNN")
plot_confusion(cnn, test_loader, classes)
```

- Thực hiện tương tự như MLP nhưng với mô hình CNN.
- Kỳ vọng kết quả tốt hơn do CNN phù hợp với xử lý ảnh hơn.




Confusion Matrix
True label
plane 789 12 68 25 16 2 8 6 38 36
car 5 926 2 4 1 2 3 0 10 47
bird 27 0 744 55 66 42 42 17 3 4
cat 5 2 58 727 43 93 36 14 13 9
deer 9 1 31 49 829 20 41 18 2 0
dog 5 2 28 155 35 744 8 18 2 3
frog 2 1 34 48 14 10 879 7 3 2
horse 7 1 29 40 51 36 2 829 2 3
ship 35 12 11 12 4 1 5 4 895 21
truck 13 40 6 11 4 2 5 4 19 896
plane car bird cat deer dog frog horse ship truck
800
600
400
200
0