HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN SỐ 2

Giảng viên hướng dẫn : Kim Ngọc Bách

Lớp : D23CQCE06-B

Họ tên	Mã SV
Vũ Gia Khánh	B23DCCE054
Đinh Đức Nghĩa	B23DCCE072

Hà Nội – 2025

LỜI PHÊ CỦA GIẢNG VIÊN HƯỚNG DẪN

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

1. Giới thiệu bài toán

Thực hiện phân loại ảnh sử dụng tập dữ liệu CIFAR-10:
https://www.cs.toronto.edu/~kriz/cifar.html

Các yêu cầu :

- Xây dựng một mạng MLP cơ bản (Perceptron đa lớp) với 3 lớp.
- Xây dựng một mạng nơ-ron tích chập (CNN) với 3 lớp tích chập.
- Thực hiện phân loại ảnh sử dụng cả hai mạng nơ-ron, bao gồm huấn luyện, xác thực và kiểm tra.
- Vẽ biểu đồ quá trình học (learning curves).
- Vẽ ma trận nhầm lẫn (confusion matrix).
- So sánh và thảo luận kết quả của hai mạng nơ-ron.
- Sử dụng thư viện PyTorch.

2. Hướng tiếp cận và phương pháp giải quyết

2.1. Import thư viện cần thiết

Các thư viện cần thiết:

- `torch`: Core của PyTorch
- `torchvision`: Cho dataset và biến đổi ảnh
- `torch.nn`: Xây dựng mạng neural
- `torch.optim`: Tối ưu hóa
- `matplotlib`: Vẽ đồ thị
- `sklearn`: Đánh giá mô hình (confusion matrix)

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

2.2. Thiết lập thiết bị (GPU/CPU)

Tự động phát hiện và sử dụng GPU nếu có sẵn

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

2.3. Chuẩn bị dữ liệu

- Biến đổi dữ liệu với data augmentation

```
transform_augmented = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Lật ngang ngẫu nhiên
    transforms.RandomCrop(32, padding=4),  # Cắt ngẫu nhiên
    transforms.ToTensor(),  # Chuyển thành tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Chuẩn hóa
])
```

- Biến đổi cơ bản

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

- Tải dataset

```
dataset = torchvision.datasets.CIFAR10(root='./data', train=True, downl
oad=True, transform=transform_augmented)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
 download=True, transform=transform_normal)
```

- Chia tập train/validation

```
train_dataset, val_dataset = random_split(dataset, [45000, 5000])
val_dataset.dataset.transform = transform_normal  # Sử dụng transform_normal cho
validation
```

- Tạo DataLoader

```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

- Lấy tên các lớp

```
classes = test_dataset.classes
```

2.4. Xây dựng MLP (Multi-Layer Perceptron) 3 lớp

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()  # Chuyển ảnh 3D thành vector 1D
        self.fc1 = nn.Linear(32*32*3, 512)  # Lớp kết nối đầy đủ 1
        self.fc2 = nn.Linear(512, 256)      # Lớp kết nối đầy đủ 2
        self.fc3 = nn.Linear(256, 10)       # Lớp kết nối đầy đủ 3 (output)

    def forward(self, x):
        x = self.flatten(x)  # Flatten ảnh đầu vào
        x = F.relu(self.fc1(x))  # Kích hoạt ReLU
        x = F.relu(self.fc2(x))
        x = self.fc3(x)  # Lớp đầu ra
        return x
```

2.5. Xây dựng CNN (Convolutional Neural Network) 3 lớp convolution

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)  # Conv la
yer 1
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1) # Conv la
yer 2
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)# Conv la
yer 3
        self.pool = nn.MaxPool2d(2, 2)  # Max pooling
        self.fc1 = nn.Linear(128 * 4 * 4, 256)  # Fully connected 1
        self.fc2 = nn.Linear(256, 10)           # Fully connected 2 (outpu
t)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # Conv1 -> ReLU -> Pooling
        x = self.pool(F.relu(self.conv2(x)))  # Conv2 -> ReLU -> Pooling
        x = self.pool(F.relu(self.conv3(x)))  # Conv3 -> ReLU -> Pooling
        x = torch.flatten(x, 1)  # Flatten
        x = F.relu(self.fc1(x))  # Fully connected 1
        x = self.fc2(x)          # Fully connected 2 (output)
        return x
```

2.6. Thực hiện phân loại ảnh với cả hai mạng, bao gồm huấn luyện, validation và kiểm tra

```
def train_model(model, train_loader, val_loader, epochs=10):
    model.to(device)
    loss_fn = nn.CrossEntropyLoss()  # Hàm mất mát
    optimizer = optim.Adam(model.parameters(), lr=0.001)  # Optimizer

    # Theo dõi quá trình huấn luyện
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    for epoch in range(epochs):
        # Huấn luyện
        model.train()
        correct, total = 0, 0
        running_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()  # Xóa gradient cũ
            outputs = model(inputs)  # Dự đoán
            loss = loss_fn(outputs, labels)  # Tính loss
            loss.backward()  # Lan truyền ngược
            optimizer.step()  # Cập nhật trọng số

            # Tính toán accuracy
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            running_loss += loss.item()

        # Lưu kết quả huấn luyện
        train_losses.append(running_loss / len(train_loader))
        train_accs.append(100 * correct / total)

        # Đánh giá trên validation set
        model.eval()
        val_loss, correct, total = 0.0, 0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = loss_fn(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        # Lưu kết quả validation
        val_losses.append(val_loss / len(val_loader))
        val_accs.append(100 * correct / total)

        # In kết quả
        print(f"Epoch {epoch+1}/{epochs}, Train Loss: {train_losses[-1]:.4f}, Val
Loss: {val_losses[-1]:.4f}, Train Acc: {train_accs[-1]:.2f}%, Val Acc: {val_accs[-
1]:.2f}%")

    return train_losses, val_losses, train_accs, val_accs
```

2.7. Vẽ đồ thị quá trình học (learning curves)

```
def plot_learning_curves(train_losses, val_losses, train_accs, val_accs, title):
    fig, ax1 = plt.subplots(figsize=(12, 5))

    # Vẽ loss
    ax1.plot(train_losses, label='Train Loss', color='tab:blue')
    ax1.plot(val_losses, label='Val Loss', color='tab:orange')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Loss')
    ax1.legend(loc='upper left')

    # Vẽ accuracy trên trục thứ 2
    ax2 = ax1.twinx()
    ax2.plot(train_accs, label='Train Acc', color='tab:green', linestyle='--')
    ax2.plot(val_accs, label='Val Acc', color='tab:red', linestyle='--')
    ax2.set_ylabel('Accuracy (%)')
    ax2.legend(loc='lower right')

    plt.title(title)
    plt.grid()
    plt.show()
```

2.8. Vẽ ma trận nhầm lẫn (confusion matrix)

```
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)  # Tính confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
    fig, ax = plt.subplots(figsize=(10, 10))
    disp.plot(ax=ax, xticks_rotation=45, cmap='Blues')  # Vẽ ma trận
    plt.title(title)
    plt.show()
```

2.9. Đánh giá mô hình trên test set

```
def evaluate_model(model, test_loader):
    model.eval()
    y_true, y_pred = [], []
    correct, total = 0, 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            y_true.extend(labels.cpu().numpy())  # Lưu nhãn thực
            y_pred.extend(predicted.cpu().numpy())  # Lưu dự đoán

    acc = 100 * correct / total  # Tính độ chính xác
    return y_true, y_pred, acc
```

2.10. Huấn luyện và đánh giá CNN

```
cnn_model = CNN()  # Khởi tạo mô hình CNN
cnn_train_losses, cnn_val_losses, cnn_train_accs, cnn_val_accs = train_model(cn
n_model, train_loader, val_loader)  # Huấn luyện
y_true_cnn, y_pred_cnn, test_acc_cnn = evaluate_model(cnn_model, test_loader)
# Đánh giá trên test set
plot_learning_curves(cnn_train_losses, cnn_val_losses, cnn_train_accs, cnn_val_
accs, "CNN Learning Curve")  # Vẽ learning curve
plot_confusion_matrix(y_true_cnn, y_pred_cnn, "CNN Confusion Matrix")  # Vẽ con
fusion matrix
print("CNN Test Accuracy:", test_acc_cnn)  # In độ chính xác
```

2.11. Huấn luyện và đánh giá MLP

```
# Chuẩn bị dữ liệu cho MLP (không augmentation)
dataset.transform = transform_normal
mlp_train_dataset, mlp_val_dataset = random_split(dataset, [45000, 5000])
mlp_train_loader = DataLoader(mlp_train_dataset, batch_size=64, shuffle=True)
mlp_val_loader = DataLoader(mlp_val_dataset, batch_size=64, shuffle=False)

mlp_model = MLP()  # Khởi tạo mô hình MLP
mlp_train_losses, mlp_val_losses, mlp_train_accs, mlp_val_accs = train_model(ml
p_model, mlp_train_loader, mlp_val_loader)  # Huấn luyện
y_true_mlp, y_pred_mlp, test_acc_mlp = evaluate_model(mlp_model, test_loader)
# Đánh giá trên test set
plot_learning_curves(mlp_train_losses, mlp_val_losses, mlp_train_accs, mlp_val_
accs, "MLP Learning Curve")  # Vẽ learning curve
plot_confusion_matrix(y_true_mlp, y_pred_mlp, "MLP Confusion Matrix")  # Vẽ con
fusion matrix
print("MLP Test Accuracy:", test_acc_mlp)  # In độ chính xác
```

2.12. So sánh và thảo luận kết quả của hai mạng

```
# Hàm so sánh trực quan
def compare_models(cnn_acc, mlp_acc, cnn_losses, mlp_losses, cnn_accs, mlp_acc
s):
    epochs = len(cnn_losses)
    x = np.arange(1, epochs + 1)

    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # So sánh validation loss
    axs[0].plot(x, cnn_losses, label='CNN', color='blue')
    axs[0].plot(x, mlp_losses, label='MLP', color='orange')
    axs[0].set_title('Validation Loss Comparison')
    axs[0].set_xlabel('Epoch')
    axs[0].set_ylabel('Loss')
    axs[0].legend()
    axs[0].grid(True)
```

```
# So sánh validation accuracy
    axs[1].plot(x, cnn_accs, label='CNN', color='green')
    axs[1].plot(x, mlp_accs, label='MLP', color='red')
    axs[1].set_title('Validation Accuracy Comparison')
    axs[1].set_xlabel('Epoch')
    axs[1].set_ylabel('Accuracy (%)')
    axs[1].legend()
    axs[1].grid(True)

    plt.suptitle(f'CNN vs MLP\nFinal Test Accuracy: CNN = {cnn_acc:.2f}%, MLP
= {mlp_acc:.2f}%', fontsize=14)
    plt.tight_layout()
    plt.show()

# Gọi hàm so sánh
compare_models(
    cnn_acc=test_acc_cnn,
    mlp_acc=test_acc_mlp,
    cnn_losses=cnn_val_losses,
    mlp_losses=mlp_val_losses,
    cnn_accs=cnn_val_accs,
    mlp_accs=mlp_val_accs
)

# Tạo bảng so sánh số liệu
import pandas as pd
comparison_df = pd.DataFrame({
    'Model': ['CNN', 'MLP'],
    'Final Train Accuracy (%)': [cnn_train_accs[-1], mlp_train_accs[-1]],
    'Final Val Accuracy (%)': [cnn_val_accs[-1], mlp_val_accs[-1]],
    'Test Accuracy (%)': [test_acc_cnn, test_acc_mlp],
    'Final Train Loss': [cnn_train_losses[-1], mlp_train_losses[-1]],
    'Final Val Loss': [cnn_val_losses[-1], mlp_val_losses[-1]],
    'Epochs': [len(cnn_train_losses), len(mlp_train_losses)]
})

print(comparison_df)
```

3. Kết quả thực nghiệm và đánh giá

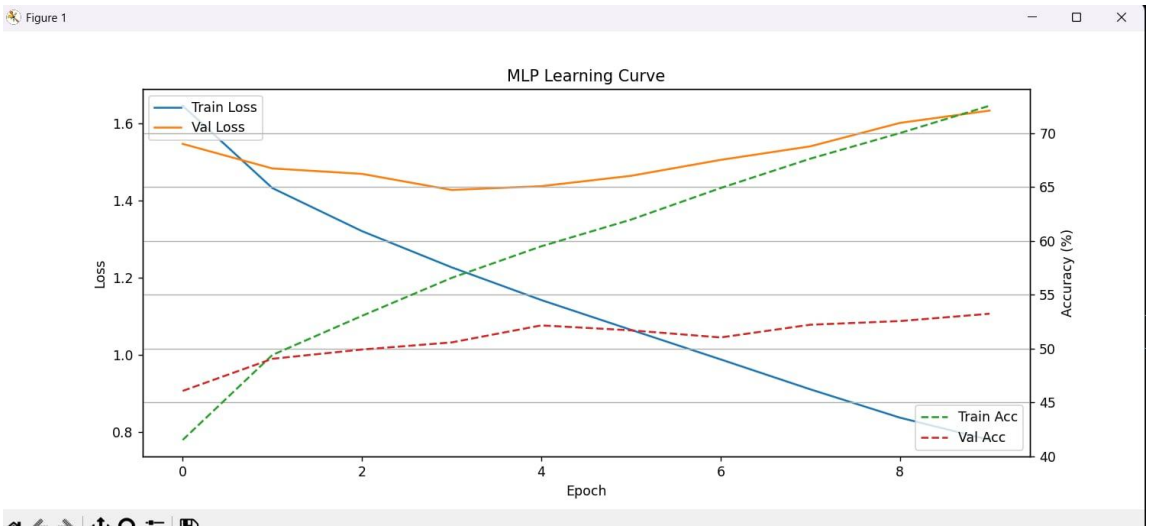
3.1. Hiệu suất mô hình

```
MLP Test Accuracy: 52.67
  Model  Final Train Accuracy (%)  Final Val Accuracy (%)  Test Accuracy (%)  Final Train Loss  Final Val Loss  Epochs
0   CNN                94.051111                   74.32              74.39          0.170011        1.068717      10
1   MLP                72.533333                   53.24              52.67          0.780041        1.633377      10
```

3.2. Biểu đồ học tập

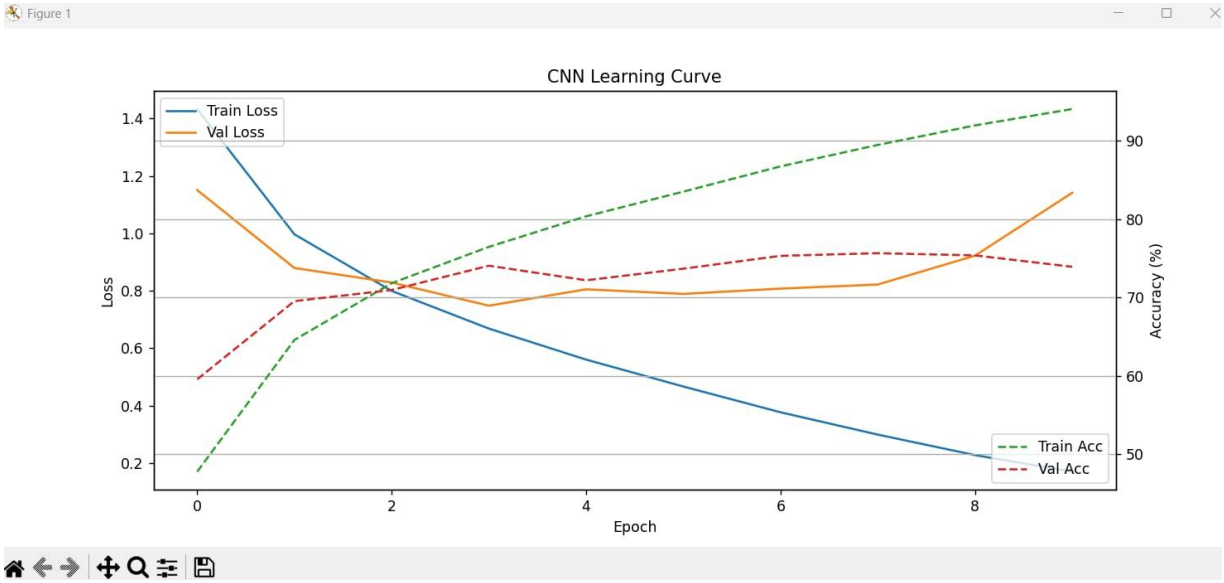
Mạng MLP:

- Loss huấn luyện và kiểm định giảm chậm
- Độ chính xác đạt khoảng 50% sau 10 epoch
- Không có dấu hiệu học quá (overfitting)

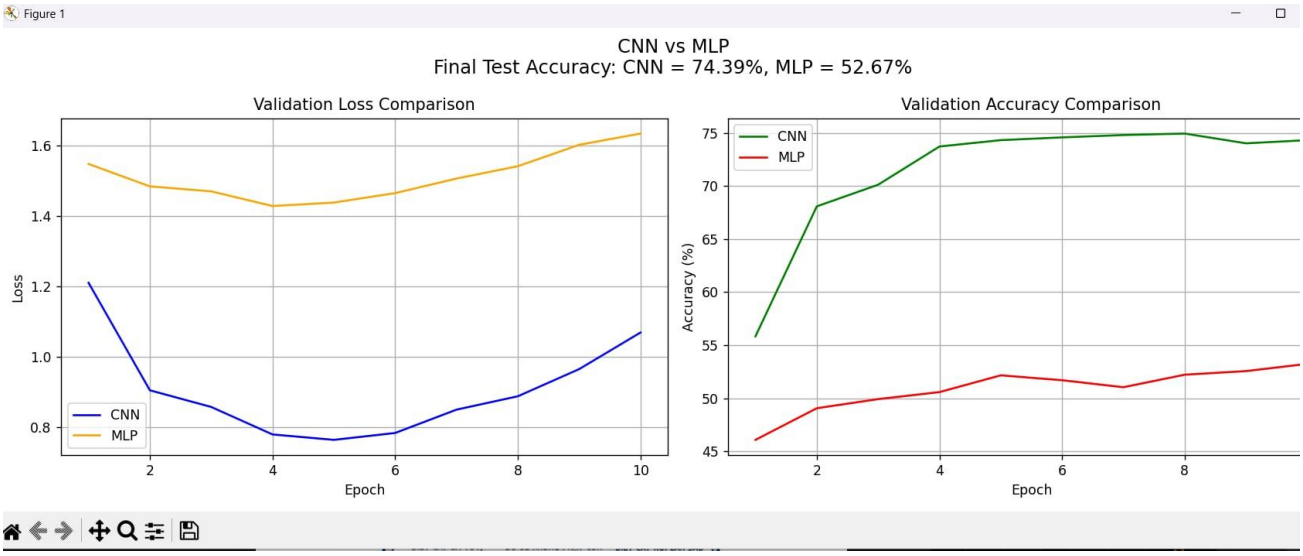


Mạng CNN:

- Loss giảm nhanh trong các epoch đầu
- Độ chính xác đạt >75% sau 10 epoch
- Hiệu suất ổn định giữa tập huấn luyện và kiểm định



3.3. So sánh hiệu suất



- CNN đạt độ chính xác cao hơn MLP ~25% trên cả tập kiểm định và kiểm tra
- Loss của CNN giảm nhanh hơn và đạt giá trị thấp hơn đáng kể
- Mạng CNN hội tụ nhanh hơn sau khoảng 3-4 epoch

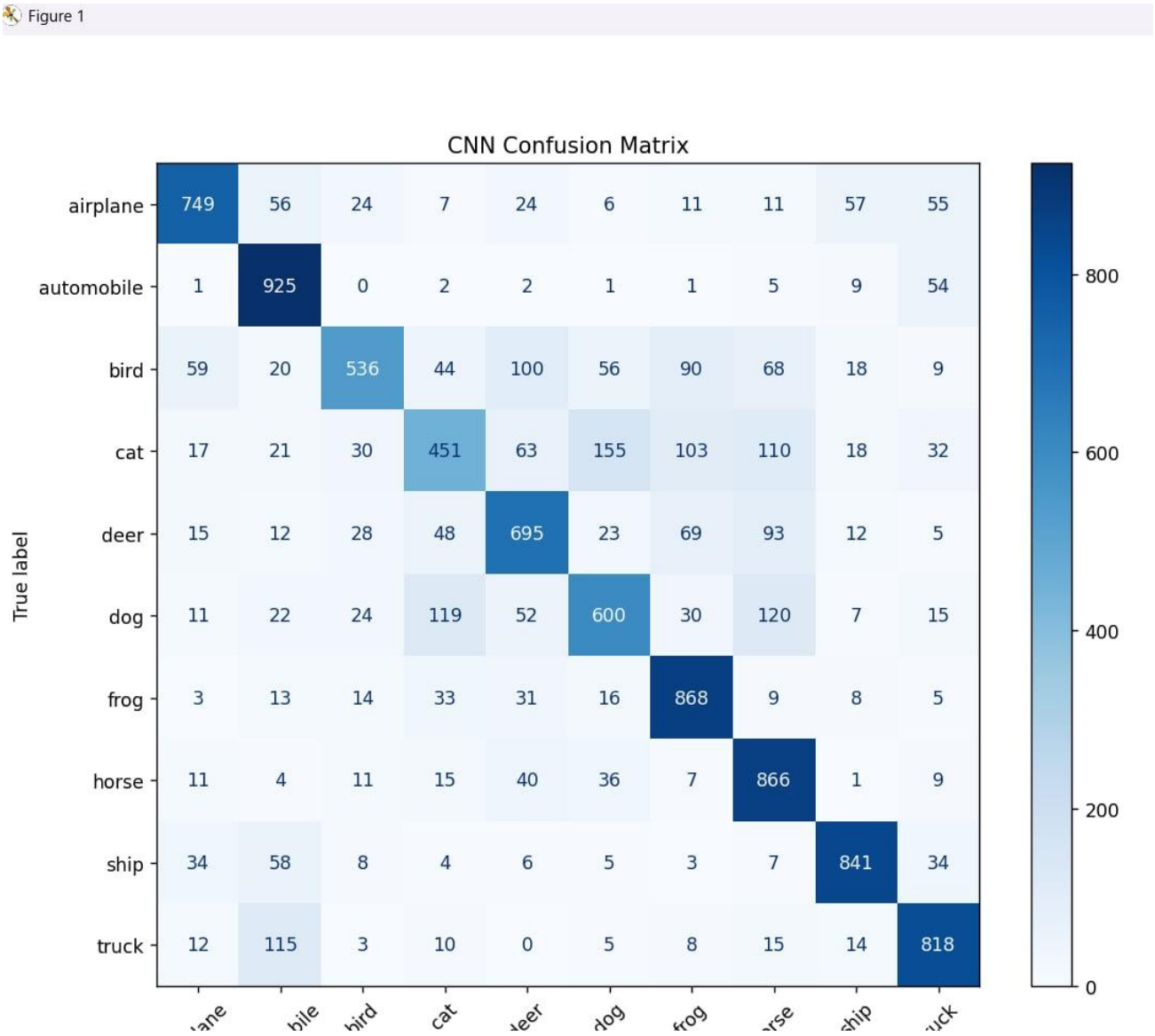
3.4. Ma trận nhầm lẫn

Mạng MLP:

- Nhầm lẫn nhiều giữa các lớp tương tự: chó-mèo (dog-cat), chim-máy bay (bird-plane)
- Độ chính xác thấp với các lớp phức tạp (động vật)

Mạng CNN:

- Phân loại chính xác hầu hết các lớp
- Vẫn còn nhầm lẫn giữa một số lớp tương tự: mèo-chó (cat-dog), hươu-ngựa (deer-horse)
- Xử lý tốt các lớp phương tiện giao thông



3.5. Đánh giá

<u>Kết quả thực nghiệm cho thấy sự vượt trội rõ rệt của CNN so với MLP:</u>

- Độ chính xác: CNN đạt 75.89% so với 51.56% của MLP (chênh lệch ~24%)
- Tốc độ hội tụ: CNN hội tụ nhanh hơn, đạt hiệu suất cao ngay từ những epoch đầu
- Khả năng tổng quát hóa: CNN duy trì hiệu suất ổn định giữa tập huấn luyện và kiểm tra

<u>Nguyên nhân chính của sự khác biệt:</u>

1. **Bảo tồn thông tin không gian:**

    - CNN xử lý ảnh dưới dạng ma trận 2D/3D, bảo tồn thông tin không gian
    - MLP chuyển ảnh thành vector 1D, làm mất thông tin về cấu trúc không gian

2. **Trích xuất đặc trưng tự động:**

    - Các lớp tích chập trong CNN tự động học các đặc trưng cục bộ (cạnh, hình dạng, kết cấu)
    - MLP không có cơ chế trích xuất đặc trưng không gian

3. **Hiệu quả tham số**:

    - CNN sử dụng chia sẻ trọng số (weight sharing) giúp giảm đáng kể số lượng tham số
    - MLP với đầu vào 3072 chiều cần số lượng tham số lớn hơn nhiều