HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN 2

MÔN HỌC: NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn	**: Kim Ngọc Bách**
Lớp	**: D23CQCE06-B**
Thành viên	**:**
Lê Minh Đức	**- B23DCVT095**
Nguyễn Phạm Anh Phúc	**- B23DCCE078**

Hà Nội – 2025

MỤC LỤC

MỞ ĐẦU

Trong kỷ nguyên số hóa hiện nay, thị giác máy tính đóng vai trò ngày càng quan trọng trong nhiều lĩnh vực, từ y tế, sản xuất đến an ninh và giải trí. Khả năng nhận diện, phân tích và hiểu nội dung hình ảnh của máy tính đã mở ra vô vàn tiềm năng ứng dụng. Một trong những bài toán cơ bản nhưng không kém phần thách thức trong thị giác máy tính là phân loại hình ảnh – nhiệm vụ gán nhãn chính xác cho một hình ảnh dựa trên nội dung của nó. Với sự bùng nổ của dữ liệu và sự phát triển vượt bậc của các thuật toán học sâu, đặc biệt là mạng nơ-ron tích chập (Convolutional Neural Networks - CNNs), hiệu suất của các hệ thống phân loại hình ảnh đã được cải thiện đáng kể.

Bài tập này tập trung vào việc thực hiện phân loại hình ảnh trên tập dữ liệu CIFAR-10, một bộ dữ liệu chuẩn mực và được sử dụng rộng rãi trong cộng đồng nghiên cứu thị giác máy tính. CIFAR-10 bao gồm 60.000 hình ảnh màu 32x32 pixel, được chia thành 10 lớp khác nhau (ví dụ: máy bay, ô tô, chim, mèo, chó, v.v.), với 6.000 hình ảnh cho mỗi lớp. Độ đa dạng và độ phức tạp của dữ liệu này cung cấp một môi trường lý tưởng để kiểm tra và đánh giá hiệu quả của các kiến trúc mạng nơ-ron khác nhau.

Mục tiêu chính của bài tập này là xây dựng, huấn luyện và đánh giá hiệu suất của hai kiến trúc mạng nơ-ron tiêu biểu: Mạng truyền thẳng đa lớp (Multi-Layer Perceptron - MLP) và Mạng tích chập (Convolutional Neural Network - CNN). Bằng cách so sánh hai kiến trúc này, chúng tôi sẽ khám phá cách chúng xử lý dữ liệu hình ảnh, khả năng học các đặc trưng phức tạp từ pixel thô, và những ưu nhược điểm riêng biệt của từng loại mạng trong bối cảnh phân loại hình ảnh. Toàn bộ quá trình triển khai và thực hiện sẽ được tiến hành sử dụng thư viện PyTorch, một framework mã nguồn mở mạnh mẽ và linh hoạt cho học sâu, nổi bật với khả năng tính toán trên GPU và giao diện lập trình trực quan.

Để có cái nhìn toàn diện về quá trình huấn luyện và hiệu suất mô hình, chúng tôi sẽ thực hiện một loạt các phân tích chi tiết. Cụ thể, các đường cong học tập (learning curves) sẽ được vẽ để theo dõi sự thay đổi của độ chính xác và hàm mất mát trên cả tập huấn luyện và tập kiểm tra, giúp chúng ta hiểu rõ hơn về quá trình hội tụ và khả năng tổng quát hóa của mô hình. Hơn nữa, ma trận nhầm lẫn (confusion matrix) sẽ được xây dựng để cung cấp cái nhìn định lượng về hiệu suất phân loại của từng lớp, từ đó phát hiện các lớp bị nhầm lẫn thường xuyên. Cuối cùng, chúng tôi sẽ tiến hành so sánh và thảo luận sâu sắc về kết quả đạt được từ hai kiến trúc MLP và CNN, rút ra những nhận xét về ưu điểm, hạn chế và gợi ý cho các hướng phát triển trong tương lai.

I. Yêu cầu

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

Các nhiệm vụ:

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

Một thành viên trong nhóm phải nộp bài tập qua GitHub cá nhân của họ.

Nộp bài bao gồm:

- Một tệp báo cáo (định dạng PDF)
- Mã nguồn Python

II. Các bước tiến hành

1. Chuẩn bị dữ liệu: Tải và tiền xử lý bộ dữ liệu CIFAR-10, chia thành các tập huấn luyện (train), kiểm định (validation) và kiểm tra (test).

2. Xây dựng mô hình: Định nghĩa kiến trúc cho một mạng MLP cơ bản và một mạng CNN.

3. Huấn luyện: Viết hàm để huấn luyện các mô hình trên tập huấn luyện và đánh giá trên tập kiểm định sau mỗi epoch.

4. Đánh giá: Viết hàm để đánh giá độ chính xác cuối cùng của mô hình trên tập kiểm tra và thu thập các dự đoán.

5. Trực quan hóa: Viết các hàm để vẽ biểu đồ đường cong học tập (learning curves) và ma trận nhầm lẫn (confusion matrix).

6. Thực thi: Khởi tạo, huấn luyện, đánh giá và trực quan hóa kết quả cho cả hai mô hình MLP và CNN, cuối cùng so sánh độ chính xác của chúng.

III. Mô tả chi tiết

III.1. Thư viện.

```
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, ConfusionMatrixDisplay
import seaborn as sns
```

Imports: Nhập các thư viện cần thiết. “torch” là thư viện chính của PyTorch.nn dùng để xây dựng mạng nơ-ron. optim cho các thuật toán tối ưu. “torchvision” cung cấp bộ dữ liệu CIFAR-10 và các công cụ biến đổi ảnh. “matplotlib.pyplot” và “sklearn.metrics” dùng để trực quan hóa kết quả.

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

Device: Xác định xem có GPU (cuda) khả dụng hay không. Nếu có, các tensor và mô hình sẽ được chuyển sang GPU để tăng tốc tính toán; nếu không, CPU sẽ được sử dụng.

III.2. Chuẩn bị dữ liệu CIFAR-10.

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

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=64, shuffle=True)
valloader = torch.utils.data.DataLoader(val_dataset, batch_size=64, shuffle=False)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

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

Transform: Định nghĩa một chuỗi các phép tiền xử lý ảnh.

- Transforms.ToTensor(): Chuyển đổi ảnh từ định dạng PIL Image hoặc mảng NumPy (chiều cao x chiều rộng x số kênh) thành tensor PyTorch (số kênh x chiều cao x chiều rộng) và chuẩn hóa giá trị pixel từ [0, 255] về [0.0, 1.0].
- Transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)): Chuẩn hóa giá trị pixel của tensor ảnh. Với mỗi kênh màu, nó trừ đi 0.5 (mean) và chia cho 0.5 (std). Điều này đưa giá trị pixel về khoảng [-1.0, 1.0].

Trainset, testset: Tải bộ dữ liệu CIFAR-10. root='./data' chỉ định thư mục lưu dữ liệu. train=True để tải tập huấn luyện, train=False cho tập kiểm tra. download=True tự động tải nếu chưa có. transform=transform áp dụng các phép biến đổi đã định nghĩa.

Chia tập train/validation: Tập huấn luyện gốc (trainset) được chia thành một tập huấn luyện mới (train_dataset, 80%) và một tập kiểm định (val_dataset, 20%) bằng torch.utils.data.random_split.

DataLoader: Tạo các đối tượng DataLoader cho mỗi tập dữ liệu. DataLoader giúp nạp dữ liệu theo từng batch (lô), giúp quản lý bộ nhớ hiệu quả và cần thiết cho việc huấn luyện. batch_size=64 nghĩa là mỗi lần nạp 64 ảnh. shuffle=True cho trainloader để xáo trộn dữ liệu ở mỗi epoch, giúp mô hình học tốt hơn.

Classes: Một tuple chứa tên của 10 lớp đối tượng trong CIFAR-10, dùng để hiển thị kết quả sau này.

III.3. Xây dựng mô hình MLP.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(32 * 32 * 3, 512)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(512, 256)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x
```

Class MLP(nn.Module): Định nghĩa kiến trúc của mạng MLP.

• __init__(self): Hàm khởi tạo, nơi khai báo các lớp (layers) của mạng.

○ self.flatten = nn.Flatten(): Chuyển tensor ảnh đầu vào (ví dụ: 64x3x32x32 - batch_size x channels x height x width) thành một vector phẳng (ví dụ: 64x3072).

○ self.fc1 = nn.Linear(32*32*3, 512): Lớp kết nối đầy đủ (fully connected) đầu tiên. Ảnh CIFAR-10 có kích thước 32x32 pixel và 3 kênh màu (RGB), nên sau khi làm phẳng sẽ có 32*32*3 = 3072 đặc trưng đầu vào. Lớp này có 512 nơ-ron.

○ self.relu1 = nn.ReLU(): Hàm kích hoạt ReLU (Rectified Linear Unit), một hàm phi tuyến phổ biến.

○ self.fc2 = nn.Linear(512, 256): Lớp kết nối đầy đủ thứ hai, với 256 nơ-ron.

○ self.fc3 = nn.Linear(256, 10): Lớp kết nối đầy đủ cuối cùng (lớp đầu ra), có 10 nơ-ron, tương ứng với 10 lớp của CIFAR-10.

• forward(self, x): Định nghĩa cách dữ liệu (x) truyền qua các lớp đã khai báo trong __init__.

III.4. Hàm huấn luyện mô hình.

```
def train_model(model, trainloader, valloader, epochs=10):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

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

    for epoch in range(epochs):
        # Huấn luyện
        model.train()
        running_loss = 0.0
        correct, total = 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)

        # Đánh giá trên tập validation
        model.eval()
        val_loss = 0.0
        correct, total = 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)

        print(f'Epoch {epoch+1}, Train Loss: {train_losses[-1]:.3f}, Train Acc: {train_accs[-1]:.2f}%, Val Loss: {val_losses[-1]:.3f}, Val Acc: {val_accs[-1]:.2f}%')

    return train_losses, val_losses, train_accs, val_accs
```

- **criterion = nn.CrossEntropyLoss():** Chọn hàm mất mát là CrossEntropyLoss, phù hợp cho bài toán phân loại đa lớp. Hàm này đã bao gồm cả LogSoftmax và NLLLoss.

- **optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9):** Chọn thuật toán tối ưu là SGD (Stochastic Gradient Descent) với tốc độ học (lr) là 0.001 và momentum là 0.9. model.parameters() cung cấp các tham số (trọng số) của mô hình cho optimizer để cập nhật.

- **Vòng lặp for epoch in range(epochs):** Huấn luyện qua nhiều epoch.

  - Giai đoạn Huấn luyện (model.train()):

▪ model.train(): Chuyển mô hình sang chế độ huấn luyện. Điều này quan trọng nếu mô hình có các lớp như Dropout hoặc BatchNorm, vì chúng hoạt động khác nhau giữa lúc huấn luyện và đánh giá.

▪ Lặp qua từng batch (inputs, labels) trong trainloader.

▪ inputs, labels = inputs.to(device), labels.to(device): Chuyển dữ liệu lên device.

▪ optimizer.zero_grad(): Đặt lại gradient của tất cả các tham số về 0 trước mỗi lần tính gradient mới.

▪ outputs = model(inputs): Cho dữ liệu đầu vào đi qua mô hình để nhận được dự đoán.

▪ loss = criterion(outputs, labels): Tính toán sự khác biệt (loss) giữa dự đoán và nhãn thực tế.

▪ loss.backward(): Tính toán gradient của loss theo từng tham số của mô hình (lan truyền ngược).

▪ optimizer.step(): Cập nhật các tham số của mô hình dựa trên gradient đã tính.

▪ Tính toán running_loss (tổng loss) và accuracy (độ chính xác) trên tập huấn luyện của epoch hiện tại.

○ Giai đoạn Kiểm định (model.eval()):

▪ model.eval(): Chuyển mô hình sang chế độ đánh giá.

▪ with torch.no_grad(): Tắt việc tính toán gradient, vì không cần thiết cho việc đánh giá và giúp tiết kiệm bộ nhớ, tăng tốc độ.

▪ Lặp qua từng batch trong valloader.

▪ Tính toán val_loss và val_accs trên tập kiểm định.

○ In ra loss và accuracy của cả tập huấn luyện và kiểm định sau mỗi epoch.

• Trả về lịch sử loss và accuracy của tập huấn luyện và kiểm định.

III.5. Hàm đánh giá trên tập test.

```python
def test_model(model, testloader):
    model.eval()
    correct, total = 0, 0
    all_preds, all_labels = [], []
    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    print(f'Test Accuracy: {100 * correct / total:.2f}%')
    return all_preds, all_labels
```

Hàm để đánh giá hiệu suất cuối cùng của mô hình trên tập kiểm tra (testloader).

- Tương tự như phần kiểm định trong train_model, nhưng hàm này chỉ chạy một lần sau khi mô hình đã được huấn luyện xong.
- Tính toán độ chính xác tổng thể trên tập test.
- Trả về danh sách tất cả các dự đoán (all_preds) và nhãn thực tế (all_labels) để sử dụng cho việc vẽ ma trận nhầm lẫn.

III.6. Hàm vẽ learning curves và confusion matrix

```python
def plot_learning_curves(train_losses, val_losses, train_accs, val_accs, model_name):
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.title(f'{model_name} - Loss Curve')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(train_accs, label='Train Accuracy')
    plt.plot(val_accs, label='Validation Accuracy')
    plt.title(f'{model_name} - Accuracy Curve')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.legend()

    plt.tight_layout()
    plt.savefig(f'{model_name}_learning_curves.png')
    plt.show()

def plot_confusion_matrix(labels, preds, model_name):
    cm = confusion_matrix(labels, preds)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f'{model_name} - Confusion Matrix')
    plt.savefig(f'{model_name}_confusion_matrix.png')
    plt.show()
```

plot_learning_curves(...):

- Sử dụng matplotlib.pyplot để vẽ hai biểu đồ:
  - Biểu đồ loss (Train Loss và Validation Loss) qua các epoch.
  - Biểu đồ accuracy (Train Accuracy và Validation Accuracy) qua các epoch.
- Lưu biểu đồ thành file ảnh và hiển thị.

plot_confusion_matrix(...):

- Sử dụng sklearn.metrics.confusion_matrix để tính toán ma trận nhầm lẫn dựa trên nhãn thực tế (labels) và dự đoán của mô hình (preds).
- Sử dụng sklearn.metrics.ConfusionMatrixDisplay để vẽ ma trận nhầm lẫn một cách trực quan.
- Lưu biểu đồ thành file ảnh và hiển thị.

III.7. Huấn luyện, đánh giá và so sánh kết quả

```
# Huấn luyện và đánh giá MLP
mlp = MLP().to(device)
mlp_train_losses, mlp_val_losses, mlp_train_accs, mlp_val_accs = train_model(mlp, trainloader, valloader)
mlp_preds, mlp_labels = test_model(mlp, testloader)
plot_learning_curves(mlp_train_losses, mlp_val_losses, mlp_train_accs, mlp_val_accs, 'MLP')
plot_confusion_matrix(mlp_labels, mlp_preds, 'MLP')

# Huấn luyện và đánh giá CNN
cnn = CNN().to(device)
cnn_train_losses, cnn_val_losses, cnn_train_accs, cnn_val_accs = train_model(cnn, trainloader, valloader)
cnn_preds, cnn_labels = test_model(cnn, testloader)
plot_learning_curves(cnn_train_losses, cnn_val_losses, cnn_train_accs, cnn_val_accs, 'CNN')
plot_confusion_matrix(cnn_labels, cnn_preds, 'CNN')

# So sánh kết quả
print("\nSo sánh kết quả:")
print(f"MLP Test Accuracy: {mlp_val_accs[-1]:.2f}%")
print(f"CNN Test Accuracy: {cnn_val_accs[-1]:.2f}%")
```

- **MLP:**
  - **mlp = MLP().to(device): Khởi tạo một đối tượng từ lớp MLP và chuyển nó lên device.**
  - **Gọi train_model để huấn luyện MLP (ví dụ: 10 epochs).**
  - **Gọi test_model để đánh giá MLP trên tập test.**
  - **Gọi plot_learning_curves và plot_confusion_matrix cho MLP.**
- **CNN:**
  - **Tương tự như MLP, khởi tạo, huấn luyện, đánh giá và trực quan hóa cho mô hình CNN.**
- **So sánh kết quả: In ra độ chính xác trên tập kiểm định (validation accuracy) của epoch cuối cùng cho cả MLP và CNN để so sánh.**
- ***Lưu ý: Để so sánh hiệu suất cuối cùng một cách chính xác nhất, nên so sánh độ chính xác trên tập kiểm tra (test accuracy) mà hàm test_model đã in ra.***

KẾT LUẬN

Qua quá trình thực hiện bài tập này, chúng tôi đã hoàn thành các mục tiêu đặt ra, bao gồm việc xây dựng, huấn luyện và đánh giá hai kiến trúc mạng nơ-ron cơ bản là MLP và CNN trên tập dữ liệu phân loại hình ảnh CIFAR-10 sử dụng thư viện PyTorch. Các kết quả thu được đã cung cấp một cái nhìn sâu sắc về khả năng và hạn chế của từng loại mạng trong việc xử lý dữ liệu hình ảnh phức tạp.

Cụ thể, khi phân tích hiệu suất của Mạng truyền thẳng đa lớp (MLP), chúng tôi nhận thấy rằng mặc dù MLP có khả năng học được các mối quan hệ tuyến tính và phi tuyến tính đơn giản, nhưng với dữ liệu hình ảnh như CIFAR-10 – vốn có cấu trúc không gian phức tạp và các đặc trưng cục bộ quan trọng – MLP đã gặp phải những thách thức đáng kể. Do MLP xử lý hình ảnh như một vector phẳng, nó bỏ qua các mối quan hệ không gian giữa các pixel, dẫn đến hiệu suất phân loại tương đối thấp và khả năng tổng quát hóa còn hạn chế. Các đường cong học tập thường cho thấy dấu hiệu của overfitting hoặc underfitting, và ma trận nhầm lẫn chỉ ra sự nhầm lẫn cao giữa các lớp có đặc trưng hình ảnh tương tự.

Ngược lại, Mạng tích chập (CNN) đã chứng minh được sự vượt trội rõ rệt. Với các lớp tích chập và pooling, CNN có khả năng tự động trích xuất các đặc trưng phân cấp từ hình ảnh, từ các đường nét cơ bản đến các hình dạng phức tạp hơn. Điều này cho phép CNN hiểu được cấu trúc không gian của hình ảnh một cách hiệu quả hơn nhiều so với MLP. Các đường cong học tập của CNN thường cho thấy sự hội tụ tốt hơn, với độ chính xác cao hơn trên cả tập huấn luyện và tập kiểm tra, chứng tỏ khả năng tổng quát hóa mạnh mẽ. Ma trận nhầm lẫn cũng xác nhận rằng CNN giảm thiểu đáng kể sự nhầm lẫn giữa các lớp, đạt được độ chính xác phân loại cao hơn đáng kể.

Việc sử dụng thư viện PyTorch trong suốt quá trình triển khai đã mang lại nhiều lợi ích. PyTorch cung cấp một môi trường lập trình linh hoạt, cho phép dễ dàng xây dựng và thử nghiệm các kiến trúc mạng nơ-ron khác nhau, quản lý dữ liệu hiệu quả và tận dụng sức mạnh của GPU để tăng tốc quá trình huấn luyện.

Tóm lại, bài tập này đã củng cố hiểu biết của chúng tôi về ứng dụng của học sâu trong thị giác máy tính và tầm quan trọng của việc lựa chọn kiến trúc mạng nơ-ron phù hợp. Sự khác biệt rõ rệt về hiệu suất giữa MLP và CNN trên tập dữ liệu CIFAR-10 đã làm nổi bật vai trò không thể thiếu của các lớp tích chập trong việc xử lý và hiểu dữ liệu hình ảnh. Mặc dù vẫn còn nhiều hướng phát triển tiềm năng như tinh chỉnh siêu tham số, sử dụng các kỹ thuật tăng cường dữ liệu (data augmentation), hoặc khám phá các kiến trúc CNN tiên tiến hơn (ví dụ: ResNet, VGG), những kết quả ban đầu này đã cung cấp một nền tảng vững chắc và định hướng cho các nghiên cứu sâu hơn trong tương lai. Bài tập này không chỉ là một cơ hội thực hành kỹ thuật mà còn là một minh chứng rõ ràng cho sức mạnh của học sâu trong việc giải quyết các bài toán phức tạp trong thế giới thực.