Phân tích Toàn diện Phân loại Hình ảnh CIFAR-10 bằng Mô hình MLP và CNN

Nguyễn Đức Hà Hùng, Trịnh Hoàng Việt

26/05/2025

Tóm tắt nội dung

Báo cáo này trình bày phân tích chi tiết về việc triển khai và đánh giá hiệu suất của hai mô hình mạng nơ-ron: Perceptron Đa lớp (MLP) và Mạng Nơ-ron Tích chập (CNN) để phân loại tập dữ liệu CIFAR-10, gồm 60,000 hình ảnh RGB 32x32 thuộc 10 lớp. Sử dụng PyTorch, nghiên cứu bao gồm tiền xử lý dữ liệu, thiết kế mô hình, huấn luyện, xác thực và kiểm tra. Mô hình CNN vượt trội hơn MLP, đạt độ chính xác kiểm tra 71.03% so với 52.03% của MLP. Kết quả huấn luyện và xác thực chi tiết, bao gồm mất mát và độ chính xác qua từng epoch, được phân tích kỹ lưỡng, cùng với thảo luận về hành vi của mô hình.

Mục lục

1 Giới thiệu

Tập dữ liệu CIFAR-10 là một chuẩn mực cho bài toán phân loại hình ảnh, gồm 60,000 hình ảnh RGB 32x32, chia thành 50,000 hình ảnh huấn luyện (trong đó 40,000 dùng để huấn luyện và 10,000 dùng để xác thực) và 10,000 hình ảnh kiểm tra thuộc 10 lớp: máy bay, ô tô, chim, mèo, hươu, chó, ếch, ngựa, tàu thủy và xe tải. Dự án này triển khai và so sánh hai kiến trúc mạng nơ-ron—Perceptron Đa lớp (MLP) và Mạng Nơ-ron Tích chập (CNN)—để phân loại các hình ảnh này bằng PyTorch. Mục tiêu là đánh giá hiệu suất thông qua mất mát huấn luyện, độ chính xác xác thực và kiểm tra, đồng thời phân tích hiệu quả của việc trích xuất đặc trưng không gian của CNN so với cách tiếp cận hoàn toàn kết nối của MLP. Kết quả, được triển khai một cách có thể tái lập, nhấn mạnh hiệu suất vượt trội của CNN.

2 Phương pháp

2.1 Tập dữ liệu và Tiền xử lý

Tập dữ liệu CIFAR-10 được tiền xử lý như sau:

- **Biến đổi**: Hình ảnh được chuyển thành tensor bằng `transforms.ToTensor()` và chuẩn hóa về $[-1, 1]$ với giá trị trung bình 0.5 và độ lệch chuẩn 0.5 cho mỗi kênh bằng `transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`.
- **Tải dữ liệu**: Tập dữ liệu được tải bằng `torchvision.datasets.CIFAR10`, với kích thước batch là 64. Tập huấn luyện (50,000 hình) được chia thành 40,000 hình huấn luyện và 10,000 hình xác thực, còn 10,000 hình còn lại dùng để kiểm tra. Dữ liệu được xáo trộn cho huấn luyện và sử dụng 2 luồng xử lý để tăng hiệu quả.

2.2 Kiến trúc Mô hình

Hai mô hình được triển khai:

- **MLP**: Mạng kết nối đầy đủ với:
  - Lớp đầu vào: 3072 nơ-ron (hình ảnh 32x32x3 được làm phẳng).
  - Lớp ẩn: 512 nơ-ron và 256 nơ-ron, mỗi lớp sử dụng hàm kích hoạt ReLU.
  - Lớp đầu ra: 10 nơ-ron cho xác suất của các lớp.

  Tổng số tham số: khoảng 1.8 triệu.
- **CNN**: Mạng tích chập với:
  - Lớp tích chập:
    * Lớp 1: 3 kênh đầu vào, 32 bộ lọc (3x3, padding=1), ReLU, max-pooling 2x2.
    * Lớp 2: 32 kênh đầu vào, 64 bộ lọc (3x3, padding=1), ReLU, max-pooling 2x2.
    * Lớp 3: 64 kênh đầu vào, 64 bộ lọc (3x3, padding=1), ReLU.

– Lớp kết nối đầy đủ: 4096 nơ-ron (làm phẳng) đến 512 nơ-ron (ReLU), sau đó đến 10 nơ-ron.

Tổng số tham số: khoảng 2.3 triệu.

2.3 Quy trình Huấn luyện

Cả hai mô hình được huấn luyện trong 10 epoch với:

- **Hàm mất mát**: Mất mát entropy chéo (`nn.CrossEntropyLoss`).
- **Trình tối ưu**: Adam (`optim.Adam`) với tốc độ học 0.001.
- **Thiết bị**: GPU CUDA nếu có, nếu không dùng CPU.

Quá trình huấn luyện bao gồm lan truyền xuôi, tính toán mất mát, lan truyền ngược và cập nhật tham số. Mất mát và độ chính xác được ghi nhận sau mỗi epoch trên cả tập huấn luyện và tập xác thực.

2.4 Chỉ số Đánh giá

Đánh giá bao gồm:

- **Độ chính xác huấn luyện, xác thực và kiểm tra**: Tỷ lệ hình ảnh được phân loại đúng trên từng tập.
- **Mất mát và Độ chính xác**: Ghi nhận mỗi epoch để phân tích hội tụ.

3 Triển khai

Triển khai được viết bằng Python sử dụng PyTorch, với các thành phần chính được trình bày dưới đây.

3.1 Tải và Chia Dữ liệu

```
import torch
import torchvision
import torchvision.transforms as transforms


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, num_workers=2)
valloader = torch.utils.data.DataLoader(val_dataset, batch_size
    =64, shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
    shuffle=False, num_workers=2)
```

Listing 1: Tải và Chia Dữ liệu CIFAR-10

3.2 Định nghĩa Mô hình

Kiến trúc MLP và CNN được định nghĩa như sau:

```
import torch
import torch.nn as nn

# MLP Model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.layers = nn.Sequential(
            nn.Linear(32 * 32 * 3, 512),  %   u    v  o: 3072 (32
               x32x3)   n   512 n  -ron
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)  %   u   ra: 10  l p
        )
    def forward(self, x):
        x = self.flatten(x)
        return self.layers(x)

# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),  % 3
               k nh   u   v o, 32  b   l c
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
```

```
            nn.ReLU(),
        )
        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 8 * 8, 512),  % 4096    n    512 n  -
               ron
            nn.ReLU(),
            nn.Linear(512, 10)
        )
    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x
```

Listing 2: Định nghĩa Mô hình MLP và CNN

3.3 Hàm Huấn luyện và Đánh giá

```
def train_model(model, trainloader, valloader, criterion,
   optimizer, num_epochs=10):
    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []

    for epoch in range(num_epochs):

        model.train()
        running_train_loss = 0.0
        correct_train = 0
        total_train = 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_train_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_train += labels.size(0)
            correct_train += (predicted == labels).sum().item()
        epoch_train_loss = running_train_loss / len(trainloader)
        epoch_train_acc = 100 * correct_train / total_train


        model.eval()
        running_val_loss = 0.0
        correct_val = 0
        total_val = 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)
                running_val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total_val += labels.size(0)
                correct_val += (predicted == labels).sum().item()
        epoch_val_loss = running_val_loss / len(valloader)
        epoch_val_acc = 100 * correct_val / total_val

        train_losses.append(epoch_train_loss)
        val_losses.append(epoch_val_loss)
        train_accuracies.append(epoch_train_acc)
        val_accuracies.append(epoch_val_acc)

        print(f'Epoch {epoch+1}, Train Loss: {epoch_train_loss:.3
            f}, Train Acc: {epoch_train_acc:.2f}%, Val Loss: {
            epoch_val_loss:.3f}, Val Acc: {epoch_val_acc:.2f}%')

    return train_losses, val_losses, train_accuracies,
        val_accuracies

def evaluate_model(model, testloader):
    model.eval()
    correct = 0
    total = 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())
    accuracy = 100 * correct / total
    return accuracy, all_preds, all_labels
```

Listing 3: Hàm Huấn luyện và Đánh giá

3.4 Chạy Mô hình

```
def main():

    mlp = MLP().to(device)
    cnn = CNN().to(device)
    criterion = nn.CrossEntropyLoss()
    mlp_optimizer = optim.Adam(mlp.parameters(), lr=0.001)
```

```
    cnn_optimizer = optim.Adam(cnn.parameters(), lr=0.001)


    print("Training MLP...")
    mlp_losses, mlp_val_losses, mlp_accs, mlp_val_accs =
       train_model(mlp, trainloader, valloader, criterion,
       mlp_optimizer)
    print("\nTraining CNN...")
    cnn_losses, cnn_val_losses, cnn_accs, cnn_val_accs =
       train_model(cnn, trainloader, valloader, criterion,
       cnn_optimizer)

    mlp_accuracy, mlp_preds, mlp_labels = evaluate_model(mlp,
       testloader)
    cnn_accuracy, cnn_preds, cnn_labels = evaluate_model(cnn,
       testloader)
    print(f'\nMLP Test Accuracy: {mlp_accuracy:.2f}%')
    print(f'CNN Test Accuracy: {cnn_accuracy:.2f}%')


    # plot_learning_curves(mlp_losses, mlp_val_losses, mlp_accs,
       mlp_val_accs, cnn_losses, cnn_val_losses, cnn_accs,
       cnn_val_accs)
    # plot_confusion_matrix(mlp_labels, mlp_preds, 'MLP Confusion
       Matrix', 'mlp_confusion_matrix.png')
    # plot_confusion_matrix(cnn_labels, cnn_preds, 'CNN Confusion
       Matrix', 'cnn_confusion_matrix.png')

if __name__ == '__main__':
    main()
```

Listing 4: Chạy Mô hình

4 Kết quả

Các mô hình được huấn luyện trong 10 epoch, và kết quả chi tiết được trình bày dưới đây.

4.1 Hiệu suất Huấn luyện và Xác thực

Mất mát và độ chính xác trên tập huấn luyện và xác thực cho mỗi mô hình được tóm tắt trong Bảng 1 và 2.

4.2 Hiệu suất Kiểm tra

Độ chính xác kiểm tra:

- **MLP**: 52.03%
- **CNN**: 71.03%

Bảng 1: Kết quả Huấn luyện và Xác thực Mô hình MLP

Epoch	Train Loss	Train Acc (%)	Val Acc (%)
1	1.660	41.23	45.78
2	1.449	48.76	48.37
3	1.341	52.63	50.18
4	1.241	56.16	49.70
5	1.154	58.98	52.36
6	1.071	62.16	51.76
7	0.993	64.69	51.50
8	0.906	67.68	52.61
9	0.833	70.06	51.44
10	0.761	72.83	51.40

Bảng 2: Kết quả Huấn luyện và Xác thực Mô hình CNN

Epoch	Train Loss	Train Acc (%)	Val Acc (%)
1	1.416	48.68	59.07
2	1.014	63.97	66.89
3	0.799	71.69	70.37
4	0.630	77.72	72.70
5	0.468	83.69	72.28
6	0.318	89.03	73.42
7	0.195	93.32	73.10
8	0.127	95.69	72.70
9	0.088	97.07	73.22
10	0.081	97.28	72.08

Mô hình CNN vượt trội hơn MLP 19.00%, cho thấy khả năng tổng quát hóa tốt hơn trên dữ liệu kiểm tra.

5 Thảo luận

Hiệu suất vượt trội của CNN (độ chính xác kiểm tra 71.03% so với 52.03% của MLP) là do các lớp tích chập, giúp trích xuất đặc trưng không gian hiệu quả, không giống cách tiếp cận làm phẳng đầu vào của MLP. Kết quả huấn luyện cho thấy CNN đạt độ chính xác huấn luyện cao 97.28% ở epoch 10, nhưng độ chính xác xác thực giảm từ 73.42% (epoch 6) xuống 72.08% (epoch 10), chỉ ra hiện tượng quá khớp nhẹ, có thể do thiếu điều chuẩn. MLP hội tụ chậm hơn, với độ chính xác xác thực dao động và đạt đỉnh 52.61% rồi giảm xuống 51.40% ở epoch 10, cho thấy khả năng tổng quát hóa hạn chế.

Hạn chế bao gồm:

- Kiến trúc nông: Các mạng sâu hơn (như ResNet) có thể cải thiện hiệu suất.
- Thiếu điều chuẩn: Dropout hoặc suy giảm trọng số có thể giảm quá khớp của CNN.
- Tiền xử lý hạn chế: Tăng cường dữ liệu (lật, xoay) có thể cải thiện tổng quát hóa của cả hai mô hình.

6 Kết luận

Nghiên cứu này chứng minh hiệu quả của CNN so với MLP trong phân loại CIFAR-10, với CNN đạt 71.03% độ chính xác kiểm tra so với 52.03% của MLP. Kết quả huấn luyện và xác thực nhấn mạnh ưu thế của CNN trong trích xuất đặc trưng, dù có dấu hiệu quá khớp. Công việc tương lai nên khám phá kiến trúc sâu hơn, kỹ thuật điều chuẩn, và học chuyển giao để nâng cao hiệu suất.

Tài liệu

[1] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical Report.

[2] Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS.

[3] LeCun, Y., et al. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE.