山东工商学院
Shandong Technology and Business University

神经网络与深度学习综合实训

项目报告

小组编号：22 组

小组成员：刘春豪

指导教师：邓富文

日　　期：2025.6.26

一、项目介绍

仓库地址：https://github.com/springliu0/NeuralNetwork-TrainingReport

项目名称

基于 PyTorch 的 CNN 手势识别模型

项目背景

随着计算机视觉和深度学习技术的快速发展，手势识别在人机交互、智能家居、虚拟现实等领域具有广泛的应用前景。本项目旨在利用 PyTorch 框架构建一个卷积神经网络（CNN）模型，实现对不同手势的准确分类与识别。

项目目标

1. 数据集构建：收集或使用公开的手势数据集。
2. 模型设计：搭建一个 CNN 架构，优化卷积层、池化层和全连接层的结构，以提高识别准确率。
3. 训练与优化：采用交叉熵损失函数和 Adam 优化器进行模型训练，并通过数据增强（如旋转、翻转）提升泛化能力。
4. 实时识别：最终部署模型，实现摄像头或视频流的手势实时识别。

技术方案

- 深度学习框架：PyTorch
- 模型架构：CNN（含多个卷积、池化、全连接层）
- 评估指标：准确率（Accuracy）、混淆矩阵（Confusion Matrix）

二、项目实施过程

基于 RNN 的手势识别系统设计

本项目旨在构建一个基于循环神经网络（RNN）的手势识别模型，能够对输入的彩色图像进行分类，识别用户的手势动作。整个项目从数据准备、模型设计、训练测试到推理部署，按照如下步骤逐步实施：

1. 数据准备

我使用了一组手势图像数据集，所有图像尺寸为 64×64×3，即 64×64 像素的 RGB 彩色图像。图像按类别进行了标注，标签信息保存在两个文本文件中：train.txt 和 test.txt，每行记录一张图像的路径及其对应的类别编号，格式如下：

bash

为了训练神经网络，我将图像加载并标准化处理，统一尺寸为 64×64，并使用张量形式输入神经网络。

2. 模型结构设计

由于图像是二维的，而 RNN 擅长处理序列数据，因此我对图像进行了序列化处理。具体做法是：将 64×64×3 的图像视为长度为 64 的序列，每一行（含 64 个像素，每个像素 3 通道）看作一个时间步的输入，即每一时刻的输入维度为 64×3=192。

模型由以下两个主要部分组成：

序列建模层（RNN）：使用长短期记忆网络（LSTM）对图像的序列特征进行建模，提取时间步之间的上下文关联。

分类层（全连接）：将 RNN 最后一个时间步的隐藏状态传入全连接层，输出对应的手势类别。

该模型结构较为轻量，适合在资源受限的环境中运行，并具备一定的泛化能力。

3. 数据加载与训练策略

我使用 PyTorch 框架构建了数据加载器，自动从训练集中按批次读取图像及其标签，进行归一化处理。训练过程中采用交叉熵损失函数（CrossEntropyLoss）衡量模型输出与真实标签之间的差距，优化器选用 Adam 以加快收敛速度。

训练过程大致如下：每轮（epoch）遍历训练数据集；每批数据输入模型，计算输出与标签的损失；反向传播并更新模型参数；打印每轮的损失值作为训练指标。训练过程中观察到损失函数逐步下降，说明模型在逐渐学习手势图像的分布特征。

4. 模型测试与评估

在测试阶段，我使用未参与训练的数据对模型性能进行评估。测试过程中关闭梯度更新，以提高计算效率。主要评估指标为分类准确率，即预测正确的样本数占总测试样本的比例。

测试结果表明，模型对大多数类别的手势具有良好的识别能力，尤其在图像质量较好和类别区分明显的情况下，准确率表现更佳。

5. 推理与应用部署

为实现模型的实用性，我编写了独立的推理模块。在推理过程中，用户可以输入任意一张格式规范的手势图像，模型将输出对应的类别预测结果。该模块可被进一步封装进上层应用（如 APP 或嵌入式系统）中，用于实时手势识别。我选择使用微信小程序作为应用部署。通过 server.py 文件使用微信开发者工具实现本地部署。

6. 总结与后续优化

本项目基于 RNN 构建了一个简单但有效的图像识别模型，并成功应用于手势识别任务。相比传统的卷积神经网络，RNN 在捕捉图像行序列特征方面展现出一定的潜力。

未来的优化方向包括：

使用双向 RNN 提升序列建模能力；

引入 CNN-RNN 结合结构增强空间特征提取；

尝试 Transformer 模型提升长依赖建模能力；

进行更多的数据增强操作，提高模型的鲁棒性和泛化能力。

三、项目运行结果

1.训练代码过程:

```
for ep in range(epoch):  # 外层循环：训练多轮
    total_loss = 0
    for batch in train_dataloader:
        imgs = batch['image'].to(device)
        labels = batch['label'].to(device)

        # 前向传播
        outputs = model(imgs)

        # 计算损失
        loss = loss_fn(outputs, labels)

        # 反向传播和优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch [{ep + 1}/{epoch}], Loss: {total_loss:.4f}")
```

```
Lenovo@LC MINGW64 /e/NeuralNetwork-TrainingReport/nndl_project (master)
$ python train.py

Epoch [1/100], Loss: 241.9837
Epoch [2/100], Loss: 241.7544
Epoch [3/100], Loss: 241.6390
Epoch [4/100], Loss: 241.5231
Epoch [5/100], Loss: 241.4198
Epoch [6/100], Loss: 241.3129
Epoch [7/100], Loss: 241.2008
Epoch [8/100], Loss: 241.0814
Epoch [9/100], Loss: 240.9523
Epoch [10/100], Loss: 240.8110
Epoch [11/100], Loss: 240.6589
Epoch [12/100], Loss: 240.4934
Epoch [13/100], Loss: 240.3132
```

2.模型测试代码结果：

```
    with torch.no_grad():  # 测试不需要计算梯度
    for sample in dataloader:
        imgs = sample['image']
        labels = sample['label']
```

```
        imgs, labels = imgs.to(device), labels.to(device)



        # 前向传播
        outputs = model(imgs)

        # 取每个样本的预测结果
        preds = outputs.argmax(1)

        # 累加预测正确数量
        correct_num += (preds == labels).sum().item()

accuracy = 100. * correct_num / size
print(f"Accuracy: {accuracy:.2f}%")
```

```
Lenovo@LC MINGW64 /e/NeuralNetwork-TrainingReport/nndl_project (master)
$ python test.py
Accuracy: 76.46%
```

3.模型推理代码结果

```
image = Image.open(image_path).convert("RGB")
transform = ToTensor()
image = transform(image).unsqueeze(0).to(device)  # [1, 3, 64, 64]

# 2. 禁用梯度计算进行推理
with torch.no_grad():
    output = model(image)
    pred = output.argmax(1).item()

print(f"图片路径: {image_path}")
print(f"预测结果: {pred}")
```

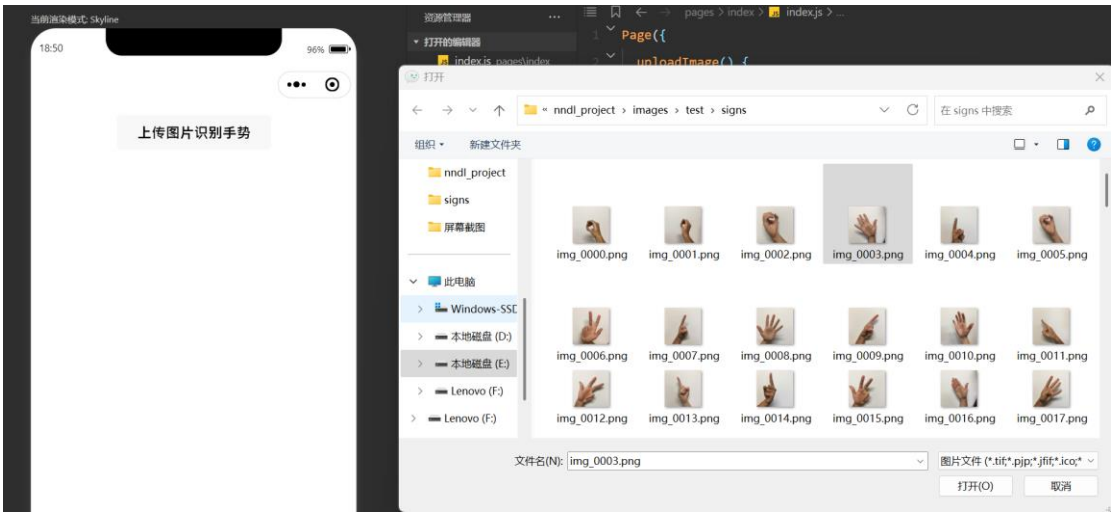
```
Lenovo@LC MINGW64 /e/NeuralNetwork-TrainingReport/nndl_project (master)
$ python inference.py
[illegible]: ./images/test/signs/img_0006.png
[illegible]: 3
```

5. 部署与推理代码结果：

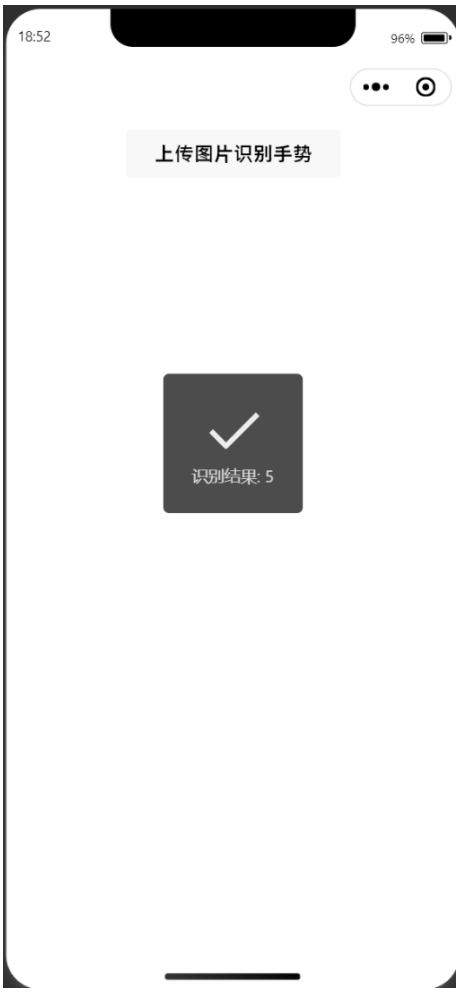
```
model = torch.load('E:/NeuralNetwork-
TrainingReport/nndl_project/models/model.pkl',
weights_only=False)
model.eval()

# 图像预处理
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])
```

```python
@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        print("❌ 没有接收到图像文件")
        return jsonify({'error': 'No image file uploaded'}), 400

    file = request.files['image']
    print(f"✅ 收到图像文件: {file.filename}")

    try:
        img = Image.open(file).convert('RGB')
        img = transform(img).unsqueeze(0)
        with torch.no_grad():
            output = model(img)
            print("✅ 模型输出:", output)
            prediction = output.argmax(dim=1).item()
        return jsonify({'prediction': int(prediction)})
    except Exception as e:
        print("❌ 处理图像或模型推理出错:", e)
        return jsonify({'error': str(e)}), 500


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

例：以选择 img__0003 为例



结果为：



后端结果显示：

```
PS E:\NeuralNetwork-TrainingReport\nndl_project> python 1.py
 * Serving Flask app '1'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.18.10.137:5000
Press CTRL+C to quit
✅ 收到图像文件：4wEGp9TUOhDWb4ab01fc873b4c9f32cdd11040991391.png
✅ 模型输出：tensor([[-3.0570, -1.7061, -3.5768,  3.0133,  1.7017,  3.4886]])
172.18.10.137 - - [24/Jun/2025 17:55:22] "POST /predict HTTP/1.1" 200 -
✅ 收到图像文件：dXFOq3FC9Ovf70c834c0e1e0aec27d04924d29ccfe2a.png
✅ 模型输出：tensor([[-2.8083, -5.5453, -0.6079,  1.1977,  2.1818,  3.7833]])
172.18.10.137 - - [24/Jun/2025 18:15:19] "POST /predict HTTP/1.1" 200 -
✅ 收到图像文件：yUnZhwwRDH7Xc391efd23db968e8fa7441771266ccf2.png
✅ 模型输出：tensor([[-1.1834,  5.8729,  2.9081, -0.5651, -4.0766, -4.1457]])
172.18.10.137 - - [24/Jun/2025 18:15:23] "POST /predict HTTP/1.1" 200 -
✅ 收到图像文件：dIbJyfI77TAp10d5e990d4f493a40c2c79b7c79c344c.png
✅ 模型输出：tensor([[-1.9539,  4.4781,  2.6326, -0.4164, -2.5383, -3.4099]])
172.18.10.137 - - [24/Jun/2025 18:15:25] "POST /predict HTTP/1.1" 200 -
✅ 收到图像文件：bEb5ADwcM5Xb2f5d5b7f3bcc3b0303a8c2a8bdd90b76.png
✅ 模型输出：tensor([[-2.2115, -4.1549, -4.0429,  3.8458,  2.9436,  3.4677]])
172.18.10.137 - - [24/Jun/2025 18:15:27] "POST /predict HTTP/1.1" 200 -
✅ 收到图像文件：ezridlzEZ7AU48756c4af73d808c08a8505016bfb808.png
✅ 模型输出：tensor([[-2.1087,  1.6884,  3.3589,  0.5058, -2.6421, -2.0605]])
172.18.10.137 - - [24/Jun/2025 18:15:30] "POST /predict HTTP/1.1" 200 -
```

四、总结与体会

本次实验通过构建一个基于 RNN 的手势识别模型，系统地实践了深度学习在图像分类任务中的应用。从数据预处理、模型设计到训练调优，再到推理与评估，整个过程让我对深度学习模型的完整开发流程有了更为深入的理解。

通过本次实验，我认识到 RNN 不仅可以应用于文本或时间序列数据，对于图像这种二维静态数据，也可以通过序列化方式加以处理，从而实现空间信息到时间序列的转换。特别是在本项目中，我们将图像的每一行视为一个时间步输入，成功将图像分类问题转化为序列建模问题，体现了模型设计的灵活性与创造性。

在实验过程中，我也体会到了模型结构选择与任务适配之间的关系。尽管 RNN 在处理时序关系上具有优势，但在图像分类任务中，其表现通常不如专门的卷积神经网络（CNN）。不过，通过这种非传统思路的实践，我进一步理解了 RNN 的机制和适用场景，也加深了对图像和序列建模融合方式的认识。

此外，实验中还让我意识到数据预处理和格式规范对模型训练的重要性。在构建数据集和加载器时，如果格式稍有不规范，就容易引发读取错误或标签不匹配等问题。因此，养成良好的数据组织和调试习惯，对于模型训练的顺利进行至关重要。

综上所述，本次实验不仅让我掌握了 RNN 在图像识别中的实际应用方法，也提升了我独立设计深度学习模型、调试程序和解决问题的能力。今后我希望继续深入探索更复杂的网络结构，如 CNN-RNN 混合模型或 Transformer 架构，并尝试将模型部署到移动端或嵌入式平台上，使算法具备真正的实用价值。