

|  |  |
| --- | --- |
| **题 目：** | 基于深度学习的图像分类识别系统 |
| **学 院：** | 人工智能学院 |
| **专 业：** | 数据科学与大数据技术 |
| **年级班别：** | 21数据科学与大数据技术1班 |
| **姓 名**： | 利韵诗 |
| **学 号：** | 421470128 |

**提交日期：** 2024 **年** 5 **月**

□□

目 录

[1 项目背景与目的 1](#_Toc183162442)

[2 数据预处理 2](#_Toc183162443)

[3 模型构建 4](#_Toc183162444)

[4 模型评估 6](#_Toc183162445)

[4.1 卷积云神经网络CNN 6](#_Toc183162446)

[4.2 循环神经网络RNN 10](#_Toc183162447)

[5 结果分析与优化 15](#_Toc183162448)

**1 项目背景与目的**

**背景：**

在信息化和数字化高速发展的时代，图像作为信息传递的重要载体，其分类识别技术正逐渐成为众多领域的研究热点。从安全监控到医疗影像分析，从自动驾驶到智能零售，图像分类识别系统均扮演着关键角色。然而，传统的图像分类识别方法受限于手工特征提取的不完备性等问题。

**目的：**

本项目旨在开发一种基于深度学习的图像分类识别系统，利用深度神经网络自动学习图像中的层次化特征，提升分类识别的准确率和鲁棒性。这不仅能推动图像分类识别技术的进步，还为多个实际应用场景提供有力的技术支撑，具有重要的现实意义和应用价值。

**意义：**

（1）提升分类准确率：通过深度学习模型，自动提取图像特征，减少人为干预，提高分类的准确性。

（2）增强系统鲁棒性：利用大量数据进行训练，使系统能够适应不同环境和条件下的图像分类任务。

（3）广泛应用前景：图像分类技术在医疗、安防、自动驾驶等领域有着广泛的应用，能够显著提升这些领域的智能化水平。

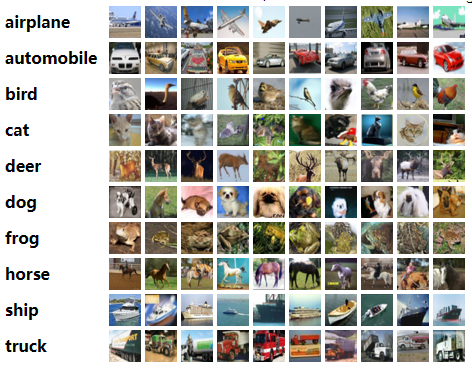
**2 数据预处理**

**数据处理：**

CIFAR-10数据集由10个类（包括）的60000张32x32彩色图像组成，每个类有6000张图像。有50000张训练图像和10000张测试图像。

数据集分为5个训练批次和1个测试批次，每个批次有10000张图像。测试批次包含每个类中随机选择的1000张图像。训练批次包含按随机顺序排列的剩余图像，但某些训练批次可能包含来自一个类的图像多于另一个类的图像。在它们之间，训练批次包含来自每个类的5000张图像。

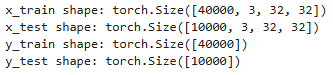
以下是数据集中的类，以及每个类的10张随机图像：



**数据处理：**

第一步：

将下载好的数据集进行划分，划分为(x\_train,y\_train),(x\_test,y\_test),并打印出其类型如下图：



第二步：

对已经划分好的训练集和测试集进行图片归一化前如图(图1)、归一化后如图(图2)：

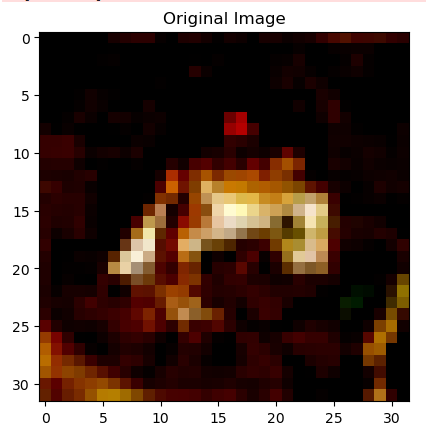
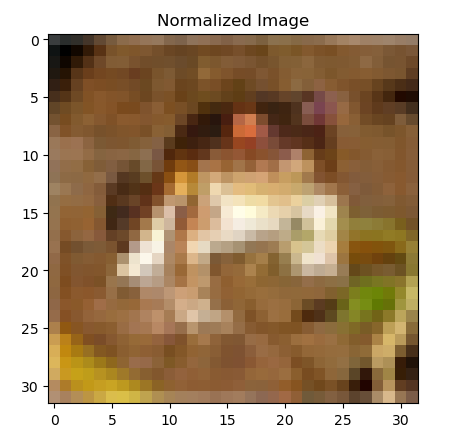
 

图1 归一化前 图2 归一化后

经过归一化的对比，归一化前后只改变图片的范围，并不改变图片的形状及其调度。

**数据的分析：**

基于 cifar10 图片数据集可以发现， cifar10 数据集 RGB 图片大小为 32× 32，颜色较为鲜明，图片锐度及其对比度较低。为了让图片导入模型进行分析，将图片进行最大值与最小值归一化处理，将取值范围从 0~255，变为 0~1，这样大大提高了分类器的准确性。

**3 模型构建**

（1）数据准备：

①数据收集：获取足够的训练数据，这是模型训练的基础。

②数据预处理：包括数据清洗、归一化、数据增强等步骤，以提高模型的泛化能力。

（2）数据集划分：

将数据集划分为训练集、验证集和测试集。通常的划分比例为80%训练集，10%验证集，10%测试集。

（3）定义模型：

①选择合适的模型架构（如卷积神经网络、循环神经网络等）。

②使用深度学习框架（如PyTorch、TensorFlow）定义模型的层次结构。

（4）选择损失函数和优化器：

①根据任务选择合适的损失函数（如交叉熵损失、均方误差等）。

②选择优化器（如SGD、Adam）来更新模型参数。

（5）模型训练：

①使用训练集进行模型训练，通过反向传播算法更新模型参数。

②在训练过程中使用验证集监控模型性能，防止过拟合。

（6）模型评估：

使用测试集评估模型的最终性能，计算准确率、精确率、召回率等指标。

（7）模型优化：

根据评估结果调整模型参数、架构或进行超参数调优，以提高模型性能。

（8）模型部署：

将训练好的模型部署到生产环境中，进行实际应用。

**4 模型评估**

**4.1 卷积云神经网络CNN**

卷积神经网络（CNN）是一种深度学习模型，特别适用于处理图像和视频数据。

①卷积层：CNN的基础层，通过卷积操作提取图像中的局部特征。卷积层使用滤波器（或卷积核）在图像上滑动，生成特征图。

②池化层：也称为下采样层，用于减少特征图的尺寸，从而降低计算复杂度和防止过拟合。常见的操作包括最大池化和平均池化。

③损失函数和优化器：CNN通过损失函数衡量预测结果与实际结果之间的差异，并使用优化器（如梯度下降）来调整网络参数，以最小化损失。

**实验代码示例：**

# 定义CNN模型

class SimpleCNN(nn.Module):

def \_\_init\_\_(self):

super(SimpleCNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 32, 3, padding=1)

self.conv2 = nn.Conv2d(32, 64, 3, padding=1)

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(64 \* 8 \* 8, 512)

self.fc2 = nn.Linear(512, 10)

def forward(self, x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = x.view(-1, 64 \* 8 \* 8)

x = F.relu(self.fc1(x))

x = self.fc2(x)

return x

net = SimpleCNN().to(device)

# 定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# 训练模型

for epoch in range(10): # 迭代次数

running\_loss = 0.0

for i in range(len(x\_train)):

inputs = x\_train[i].unsqueeze(0)

labels = y\_train[i].unsqueeze(0)

optimizer.zero\_grad()

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

if i % 200 == 199: # 每200个小批量打印一次

print(f'[Epoch {epoch + 1}, Batch {i + 1}] loss: {running\_loss / 200:.3f}')

running\_loss = 0.0

print('Finished Training')

# 保存模型

torch.save(net.state\_dict(), 'simple\_cnn.pth')

net.load\_state\_dict(torch.load('simple\_rnn.pth'))

net.eval() # 设置模型为评估模式

# 测试模型

correct = 0

total = 0

with torch.no\_grad():

for i in range(len(x\_test)):

images = x\_test[i].unsqueeze(0)

labels = y\_test[i].unsqueeze(0)

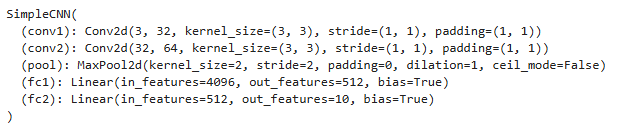
outputs = net(images)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 \* correct / total} %')



模型评估结果



测试模型结果

**4.2 循环神经网络RNN**

循环神经网络（RNN）是一种用于处理序列数据的神经网络架构，特别适合处理时间序列、自然语言处理等任务。

①循环连接：RNN的每个节点不仅接收当前输入，还接收前一个时刻的隐藏状态。这种循环连接使RNN能够记住之前的信息，并将其应用于当前的计算。

②隐藏状态：隐藏状态是RNN的记忆单元，存储了序列中前面部分的信息。随着时间步的推进，隐藏状态不断更新。

③时间步：RNN在每个时间步上处理输入数据，并更新隐藏状态。

④长短期记忆（LSTM）和门控循环单元（GRU）：为了克服传统RNN在处理长序列时的梯度消失问题，LSTM和GRU引入了门控机制，能够更好地捕捉长距离依赖关系。

⑤应用领域：RNN广泛应用于自然语言处理（如机器翻译、文本生成）、语音识别、时间序列预测等领域。

**实验代码示例：**

# 定义RNN模型

class SimpleRNN(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, num\_layers=1):

super(SimpleRNN, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.num\_layers = num\_layers

self.rnn = nn.RNN(input\_size, hidden\_size, num\_layers, batch\_first=True)

self.fc = nn.Linear(hidden\_size, output\_size)

def forward(self, x):

h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)

out, \_ = self.rnn(x, h0)

out = self.fc(out[:, -1, :])

return out

input\_size = 32 \* 32 \* 3 # CIFAR-10 图片大小为32x32，3个通道

hidden\_size = 128

output\_size = 10 # CIFAR-10 有10个类别

num\_layers = 2

net=SimpleRNN(input\_size,hidden\_size,output\_size,num\_layers).to(device)

# 定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# 训练模型

for epoch in range(10): # 迭代次数

running\_loss = 0.0

for i in range(len(x\_train)):

inputs = x\_train[i].view(1,-1, input\_size).to(device) # 调整输入形状

labels = y\_train[i].unsqueeze(0).to(device)

optimizer.zero\_grad()

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

if i % 200 == 199: # 每200个小批量打印一次

print(f'[Epoch {epoch + 1}, Batch {i + 1}] loss: {running\_loss / 200:.3f}')

running\_loss = 0.0

print('Finished Training')

# 保存模型

torch.save(net.state\_dict(), 'simple\_rnn.pth')

net.load\_state\_dict(torch.load('simple\_rnn.pth'))

net.eval() # 设置模型为评估模式

# 测试模型

correct = 0

total = 0

with torch.no\_grad():

for i in range(len(x\_test)):

images=x\_test[i].view(1,-1,input\_size)#调整输入形状

labels = y\_test[i].unsqueeze(0)

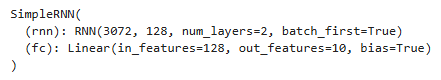
outputs = net(images)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 \* correct / total} %')



模型评估结果



测试模型结果

**5 结果分析与优化**

**结果分析：**

在训练和测试模型后，从结果中可以看出，模型在训练过程中损失逐渐减少，说明模型在学习数据的特征。然而，测试集上的准确率只有 55%，这表明模型的性能还有很大的提升空间。

**优化建议：**

（1）增加数据增强：

数据增强可以通过增加训练数据的多样性来提高模型的泛化能力。可以使用 torchvision.transforms 提供的各种数据增强方法，如随机裁剪、旋转、翻转等。

（2）调整模型架构：

当前的 RNN 模型可能不适合处理图像数据。可以尝试使用更深的卷积神经网络（CNN）架构，如 ResNet 或 VGG，这些模型在图像分类任务上表现更好。

（3）使用更好的优化器：

可以尝试使用 Adam 优化器，它在许多情况下比 SGD 表现更好。

（4）调整学习率：

学习率对模型的训练效果有很大影响。可以尝试使用学习率调度器，在训练过程中动态调整学习率。

（5）增加训练轮数：

增加训练轮数可以让模型有更多的时间学习数据的特征，但要注意防止过拟合。

（6）正则化：

使用 Dropout 或 L2 正则化可以防止模型过拟合，提高模型的泛化能力。

（7）批量归一化：

批量归一化可以加速训练过程并提高模型性能。