首都经济贸易大学

CAPITAL UNIVERSITY OF ECONOMICS AND BUSINESS

人工神经网络期末大作业

深度学习在动物图像识别中的应用：以卷积神经网络为例

学校：首都经济贸易大学

学院：管理工程学院

姓名：马菁含

学号：32021010069

日期：2023 年 12 月 29 日

一.引言

在过去的几年里，深度学习技术的迅猛发展极大地推动了人工智能领域的进步，特别是在图像识别这一领域，其影响更是显著。作为计算机视觉的一个核心分支，图像识别的应用范围广泛，从日常生活中的面部识别到医疗影像的诊断，无处不在。在动物图像识别领域，随着数据量的增加和计算能力的提升，卷积神经网络（CNN）已经成为了主流方法，其卓越的图像处理和特征提取能力为识别任务提供了强大的技术支持。本文便围绕这一主题，展示了如何使用 CNN 构建一个基本的猫狗图像分类系统，并对其性能和效果进行了详细的分析和讨论。通过这项研究，本文旨在探索深度学习在特定图像分类任务中的应用潜力，并提供实用的方法和见解。

二．程序设计思路

1．数据预处理

将所有图像统一调整为 64x64 像素，处理统一尺寸的图像，可以减少每个图像的计算负担。其次使用 transforms.ToTensor()将图像数据转换为 PyTorch 张量，转化为 PyTorch 模型处理的标准格式。最后使用预定义的均值和标准差对图像进行归一化，使得模型更快地收敛，提高其在不同数据分布上的泛化能力。

2．网络架构

在本研究中，通过设计了卷积神经网络的架构，以优化图像特征提取和学习过程，从而有效提升猫狗图像分类任务的性能。

实现细节：

（1）卷积层：模型包含四个卷积层，逐渐增加输出通道数（32, 64, 128, 256），涉及多个卷积层有助于模型从图像中提取从简单到复杂的特征，确保更深层次的特征学习。

（2）池化层：每个卷积层后跟着一个池化层，用于降低特征维度和减少计算量，同时保持特征的有效性。

（3）全连接层：根据不断调整优化，最终决定使用三个层次的全连接层。通过这些层次，模型能够在更复杂的特征空间中进行学习和分类。最终输出两个类别，分别为猫和狗。

（4）Dropout 层：在全连接层中加入 Dropout，以减少模型在训练过程中的过拟合，增强模型的泛化能力。

3．训练策略

（1）损失函数：使用交叉熵损失函数（nn.CrossEntropyLoss），这是多类别分类问题的标准选择。交叉熵损失能够衡量模型预测概率分布和实际标签之间的差异。

（2）优化器：选择 Adam 优化器（optim.Adam），它结合了动量优化和 RMSprop 的优点，能够在训练过程中自动调整学习率，加快收敛速度，同时减少对超参数的敏感性。

二．环境及工具

1．环境

图表 1 运行环境

本研究通过云服务器来执行代码，环境及配置如图 1 所示。

2．工具

（1）PyTorch：一个开源的机器学习库，广泛用于计算机视觉和自然语言处理等领域。

（2）torch.nn： PyTorch 中的一个子库，提供了构建神经网络所需的模块和类。

（3）torch.optim：包含各种优化算法的子库。

（4）torchvision.transforms：提供了一系列用于图像预处理的方法。

（5）torchvision.datasets.ImageFolder：用于加载目录结构中的数据集，其中每个类别的图片被存储在不同的文件夹中。

（6）torch.utils.data.DataLoader：用于包装数据集，提供批量加载以及数据打乱等功能，使数据可以在训练过程中被批量地送入模型。

（7）time：Python 的标准库，用于测量代码段的执行时间，以评估模型训练的效率。

（8）matplotlib.pyplot：是一个 Python 的绘图库，用于绘制图表，如在这段代码中用于显示训练过程中损失的变化。

（9）os：另一个 Python 标准库，用于与操作系统交互，比如设置环境变量等。

三．程序核心代码解析

1．环境配置与硬件选择

程序通过检查是否有 GPU 可用来配置运行环境。使用 GPU 可以显著加快深度学习模型的训练速度。

```
if torch.cuda.is_available():
    device = torch.device("cuda")
    print("GPU可用，使用GPU")
else:
    device = torch.device("cpu")
    print("GPU不可用，使用CPU")
```

此代码段设置了 PyTorch 操作的设备。如果 GPU 可用，将使用 GPU；否则，回退到 CPU。本次项目实现

2．数据加载与预处理

数据预处理是机器学习流程中至关重要的步骤。本研究中数据预处理的目的是为了将图像统一调整大小，转换为张量，并进行归一化。在此之前，应在文件夹中新建两个文件夹，分别为 dog 与 cat，即将每个类别的图片被存储在不同的文件夹中。

```python
def load_dataset(directory, batch_size=32, is_train=True):
    transform = transforms.Compose([
        transforms.Resize((64, 64)),  # 将图片统一调整为64x64
        transforms.ToTensor(),  # 将图片转换为Tensor
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # 归一化处理
    ])

    dataset = ImageFolder(directory, transform=transform)
    loader = DataLoader(dataset, batch_size=batch_size, shuffle=is_train)
    return loader
```

通过定义 load_dataset 函数实现。此函数接受数据集目录、批处理大小和一个标识符指示是否为训练集。首先，它使用 transforms.Compose 创建一个转换流程，包括将图像大小统一调整为 64x64 像素，将图像转换为张量格式，以及使用标准化来调整图像的像素值。标准化过程使用预定义的均值和标准差，这有助于模型更快地收敛。接着，函数利用 ImageFolder 加载指定目录的图像数据，并通过 DataLoader 创建数据加载器，其中包括是否打乱数据集的选项，以提高模型训练时的泛化能力。

3. CNN 模型定义

CNN 模型是此项目的核心。定义了一个包含卷积层、池化层、全连接层和 Dropout 的网络。

```python
class CatDogCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv4 = nn.Conv2d(128, 256, 3, padding=1)  # 新增卷积层
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256 * 4 * 4, 512)  # 调整全连接层参数
        self.fc2 = nn.Linear(512, 256)  # 新增全连接层
        self.fc3 = nn.Linear(256, 2)  # 输出层
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = self.pool(torch.relu(self.conv4(x)))  # 通过新增的卷积层
        x = x.view(-1, 256 * 4 * 4)
        x = torch.relu(self.fc1(self.dropout(x)))
        x = torch.relu(self.fc2(self.dropout(x)))  # 通过新增的全连接层
        x = self.fc3(x)
        return x
```

该网络由四个卷积层、三个全连接层组成，旨在有效区分猫和狗的图像。网络的前四个层是卷积层，分别具有 32、64、128 和 256 个输出通道。每个卷积层后面都跟着一个 ReLU 激活函数和最大池化层，以提取和压缩特征。这种层级结构有助于逐渐捕捉从低级到高级的图像特征。卷积层之后是三个全连接层，负责将提取的特征映射到最终的分类输出。第一个全连接层的输入维度经过调整以适应卷积层的输出，而最后一个全连接层输出两个类别，对应于猫和狗。在全连接层中加入了 Dropout 层，比率设置为 0.25，以减少过拟合并增强模型的泛化能力。forward 方法定义了数据通过网络的路径。数据依次通过卷积层、池化层和全连接层，每层都通过 ReLU 激活函数进行非线性转换，最终输出分类结果。

这一网络结构设计旨在提高猫狗图像分类任务的准确性和效率，通过逐层增加复杂性，使网络能够捕捉和利用更为复杂的图像特征。

4. 训练模型

模型的训练过程涉及前向传播、计算损失、反向传播和优化步骤。

```python
def train_model(model, train_loader, optimizer, epochs=20):
    model.train()
    total_batches = len(train_loader)
    loss_values = []  # 初始化一个列表来存储损失值

    for epoch in range(epochs):
        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = nn.CrossEntropyLoss()(outputs, labels)
            loss.backward()
            optimizer.step()

            # 存储损失值
            loss_values.append(loss.item())

            # 打印损失信息
            if i % 30 == 0:
                print(f'训练的代数: {epoch+1}/{epochs}, 当前完成: {i}/{total_batches}, 损失函数值: {loss.item()}')

    return loss_values
```

模型的训练是通过 train_model 函数实现的。此函数接收模型对象、训练数据加载器、优化器和训练轮数作为参数。训练过程中，模型在每个 epoch 对训练数据集中的所有批次进行迭代。在每个批次中，函数首先将数据传输到适当的设备（CPU 或 GPU），然后执行模型的前向传播，计算交叉熵损失，并进行反向传播和优化器步骤来更新模型权重。此外，损失值被记录下来，以便于后续分析模型的性能。定期输出的损失信息有助于监控训练过程并调整超参数。

5. 模型评估

评估模型性能是检查其在未见数据上的表现的关键步骤。

```
# 评估模型
1 个用法
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # 打印每张图片的测试结果
            for i in range(len(predicted)):
                print(f"图片 {i + 1}: 预测为 {predicted[i]}, 实际为 {labels[i]}")

    print(f'在测试集上的预测准确率: {100 * correct / total}%')
```

在进入评估模式后，函数遍历测试数据加载器中的所有数据。对于每批数据，首先将其转移到适当的设备（CPU 或 GPU），然后模型对这些数据进行前向传播以生成预测输出。使用 torch.max 函数，从输出中提取预测类别，并与实际标签进行比较，以计算正确预测的数量。通过累加所有批次的正确预测数和总数，计算出整个测试集的准确率。最终，打印出的准确率提供了对模型性能的直接量化评估。

6. 性能监控和模型保存

在训练过程中，监控损失和计算耗时是理解模型学习进度的重要方面。最后保存训练好的模型以便将来使用。

```
# 训练和评估
start_time = time.time()
loss_values = train_model(model, train_loader, optimizer, epochs=20)
end_time = time.time()
print(f"训练完成，耗时: {end_time - start_time}秒")

# 绘制损失变化图
plt.plot(loss_values)
plt.title('Training Loss')
plt.xlabel('Batch Number')
plt.ylabel('Loss')
plt.show()


# 评估模型在测试集上的表现
evaluate_model(model, test_loader)

# 保存模型（可选）
torch.save(model.state_dict(), "catdog_model.pth")
```

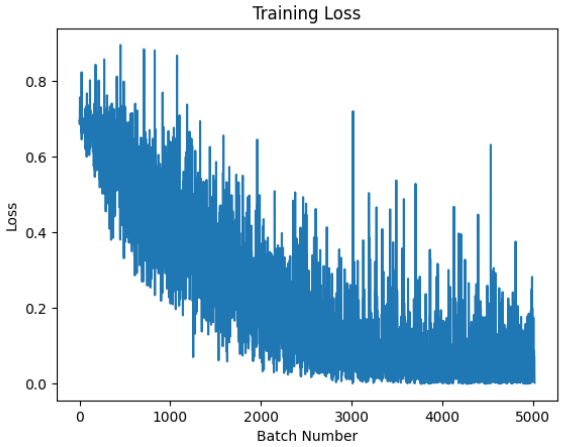
首先记录训练开始的时间，调用 train_model 函数对模型进行训练，保存该函数返回每批次的损失值。训练完成后，记录结束时间并打印总耗时，以评估训练过程的效率。使用 Matplotlib 库绘制损失值随批次变化的图表，便于最后的结论分析。然后执行 evaluate_model 函数来评估模型在测试集上的表现，为达到检验模型泛化能力的目的。最后保存训练好的模型，以便未来的使用和进一步分析。

四．结果分析

```
训练的代数: 20/20, 当前完成: 90/251, 损失函数值: 0.15701702237129211
训练的代数: 20/20, 当前完成: 120/251, 损失函数值: 0.09026765078306198
训练的代数: 20/20, 当前完成: 150/251, 损失函数值: 0.020815227180719376
训练的代数: 20/20, 当前完成: 180/251, 损失函数值: 0.06244288757443428
训练的代数: 20/20, 当前完成: 210/251, 损失函数值: 0.19452311098575592
训练的代数: 20/20, 当前完成: 240/251, 损失函数值: 0.015523114241659641
训练完成，耗时: 299.72854924201965秒
在测试集上的预测准确率: 82.00692041522491%

进程已结束,退出代码0
```

图表 2 实验结果



图表 3 损失函数变化趋势

1．准确率分析

实验结果图一显示，测试集上模型的准确率达到了 82.01%。此结果表明，在识别猫和狗的图像分类任务中，模型具有较高的识别能力。尽管在某些情况下模型可能会遇到误分类的情况，比如有些猫长得与狗极为相似。但总体而言，该模型已经展现出对动物图像有较强的区分能力。未来的研究可以通过引入更复杂的网络结构，更大的数据集，以及更先进的训练技术来进一步提高准确率。

2. 训练时间分析

如图一所示，模型的总训练时间为 299.73 秒。由于本次实验在云服务器上使用 GPU 运算完成，在高配置硬件和网络结构设置下，模型能够在相对较短的时间内完成训练，这对于实时或需要快速迭代的应用场景来说是有利的。然而通过调整模型还可以进一步优化与改进，以达到更快的训练时间。训练时间短长还受到其他多种因素的影响，如网络的深度、宽度和复杂性，以及训练算法的效率等。

3. 损失变化分析

如图二所示，通过对训练过程中损失函数的变化趋势进行观察，可以看到损失值随着批次的增加而稳步下降，表明模型正逐渐拟合训练数据。损失值的下降也显示了模型收敛的情况。然而，由于训练集中的某些图像难以识别，或者是批量数据中存在异常值，导致在某些训练阶段模型遇到了波动，损失曲线中存在着尖峰。为了实现更加平稳的收敛，未来的工作可以探索引入更细致的学习率调整策略，或者改进批次数据的选择机制。

五．结论

本研究通过构建和训练卷积神经网络模型，探讨了深度学习在动物图像识别任务中的应用潜力。实验结果显示该模型在测试集上达到了 82.01%的准确率，证明了它在区分猫和狗图像方面的有效性。模型的设计采用了四个卷积层和三个全连接层，以及 Dropout 技术以减少过拟合，从而使得网络在复杂性和泛化能力之间取得了平衡。此外，使用 Adam 优化器和交叉熵损失函数，模型在 299.73 秒内快速收敛到一个满意的解决方案。

损失函数的变化趋势表明，训练过程是稳定的，尽管某些训练批次中损失出现波动，表明可能存在难以识别的样本或数据中的噪声。未来的研究可以在数据预处理、网络结构优化和训练策略上进行改进，以进一步提高模型性能。

未来的研究可以包括扩展数据集以提高模型泛化能力、探索更复杂的网络架构来提升识别精度，以及实施更精细的超参数调整策略。此外，模型的高效训练表明了其在实时应用中的潜力，为深度学习技术在更广泛的实际应用场景中的部署奠定了基础。本次研究的探索虽然初步，但这为将来更深入的研究奠定了一个起点，同时期待后续研究能在此基础上进行扩展和完善。