人工智能实践作业一

傅申 PB20000051

目录

1. 实验内容

本次实验需要完成图像二分类任务.

1.1. 数据集

本次实验提供的数据集包含猫和狗两类, 其中每一类中有 1000 张 JPEG 格式的图片. 数据集目录结构如下:

```
data
├── cat
│   ├── cat.1.jpg
│   ├── ...
│   └── cat.1000.jpg
└── dog
    ├── dog.1.jpg
    ├── ...
    └── dog.1000.jpg
```

1.2. 实验要求

设计模型, 尽可能正确分类猫和狗.

2. 实验过程

实验的代码位于 `code` 目录下, 目录结构如下:

```
code
├── arg_parser.py
├── data_loader.py
├── early_stopping.py
├── logger.py
├── main.py
├── models.py
└── procedures.py
```

实验中使用 PyTorch 框架来搭建模型,

2.1. 实验环境

本次实验使用 PyTorch 框架来搭建模型, 并使用 BitaHub 平台的 GPU 和镜像环境对模型进行训练和测试. 具体的软硬件环境如下:

- GPU: NVIDIA Tesla V100
  - 驱动版本 470.63.01
  - CUDA 版本为 11.8
- 软件环境
  - Ubuntu 20.04.6 LTS (Docker 镜像)
  - Python 3.10
  - PyTorch 2.1.0

2.2. 模型原理

本次实验选择了 AlexNet [1, 2] 和 ResNet [3] 两种模型来进行图像二分类.

2.2.1. AlexNet

AlexNet 的架构如图 1 所示. AlexNet 是一个卷积神经网络, 它采用了深度卷积神经网络的思想, 通过卷积层, 池化层和全连接层来提取和学习图像特征.

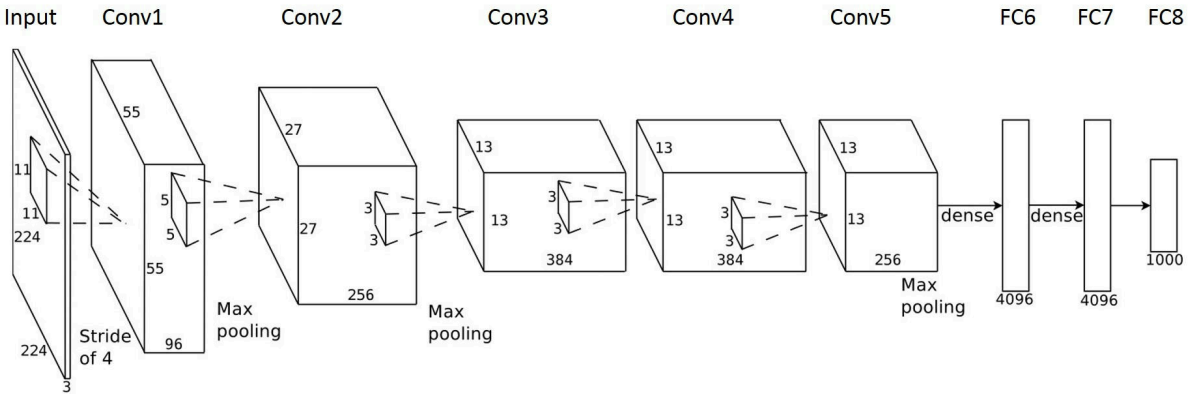

图 1 AlexNet 网络架构

AlexNet 中的各个结构的原理如下:

卷积层 AlexNet 包含多个卷积层, 这些层通过卷积操作, 学习图像的局部特征.

ReLU 激活函数 AlexNet 引入了线性的 ReLU 激活函数, 相比于传统的非线性激活函数 (如 sigmoid 和 tanh), ReLU 具有更好的梯度传播特征, 使得网络在训练时能够更好的收敛.

池化层 AlexNet 通过池化层对卷积层的输出进行下采样, 降低了特征图的尺寸, 减少了计算复杂度, 同时保留了重要的特征.

局部响应归一化 在激活函数之后, AlexNet 引入了局部响应归一化层, 对神经元的活动进行侧抑制, 增强了模型的返还能力.

全连接层 在卷积层和池化层之后, AlexNet 使用全连接层对高层次的语义信息进行整合和分类. 最后的全连接层讲输出图像对应到各类别的概率.

Dropout 为了防止过拟合, AlexNet 在全连接层中引入了 Dropout, 以随机丢弃一些神经元, 减少神经元之间的依赖关系, 提高泛化能力.

2.2.2. ResNet

ResNet 是一种深度卷积神经网络, 以其对深度网络训练的改进方法而著称. ResNet 引入了残差学习的概念, 通过使用残差块, 解决了梯度消失和梯度爆炸的问题. ResNet 在 ImageNet 图像分类比赛中取得了卓越的性能, 证明了其在实际任务中的有效性. 图 2 展示了 ResNet-18 的网络架构.

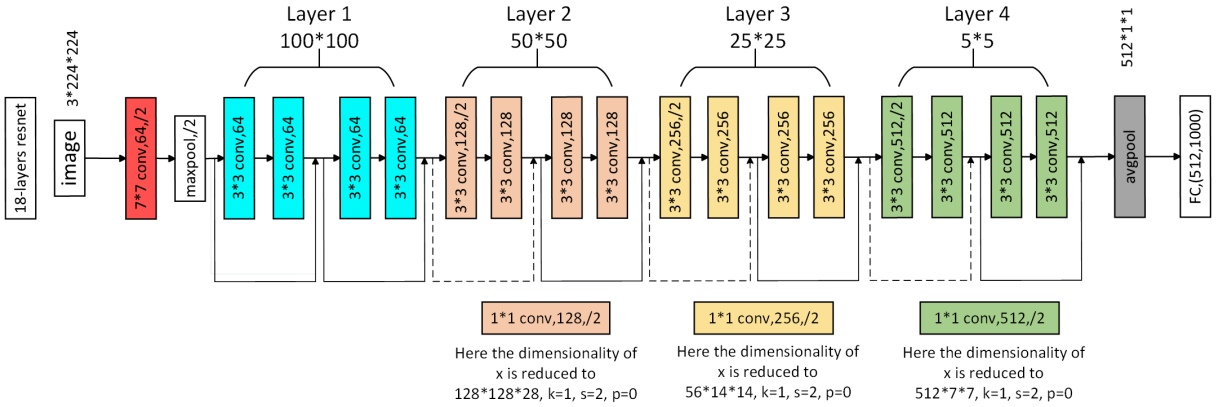

图 2 ResNet-18 网络架构

ResNet 中的一些关键模块的原理如下:

残差块 ResNet 的基本组成单元是残差块. 每个残差块包含两个分支, 一个是恒等映射, 另一个是学习到的残差. 这两个分支的输出会被相加, 使得网络可以学习如何拟合残差.

跳跃连接 残差块中的跳跃连接允许梯度直接通过网络层级进行传播, 使得信息可以从网络的前部直接传播到后部, 从而减缓梯度消失的问题.

全局平均池化 在网络的最后, ResNet 使用全局平均池化来将最后一层的特征图转化为固定大小的向量, 用于分类任务. 这种操作减少了参数数量, 同时提供了更好的泛化性能.

2.3. 实验步骤

该部分展示的代码中不包含 logging/输出的部分.

2.3.1. 数据预处理

对数据进行预处理的代码位于 `code/data_loaders.py` 中.

首先, 对于整个数据集目录, 将其加载为 `torchvision.datasets.ImageFolder`, 以便后续处理. 在加载的过程中, 我对图像进行了一系列变换, 以增强模型的泛化能力, 并加速模型的收敛. 具体流程如下所示:

```
def __get_image_folder(size: int) -> datasets.ImageFolder:
    image_folder = datasets.ImageFolder(
        DATA_DIR,
        transform=transforms.Compose(
```

```
            [
                transforms.RandomResizedCrop(size),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize(
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225],
                ),
            ]
        ),
    )
    return image_folder
```

然后, 将数据集划分为训练集, 验证集和测试集, 其中划分比例为 8:1:1. 这些数据集会被转换为 `torch.utils.data.DataLoader` 对象, 以便后续训练和测试. 具体代码如下:

```
def get_data_loaders(
    image_size,
    train_val_test_ratio: list[float],
    batch_size: int,
) -> tuple[DataLoaders, list[str]]:
    """
    Creates data loaders for train, test and validation data sets.
    Returns data loaders and a list of target classes.
    """
    image_folder = __get_image_folder(image_size)
    target_classes = image_folder.classes

    train_set, val_set, test_set = random_split(
        image_folder, train_val_test_ratio
    )

    def subset_to_loader(subset: Subset):
        return DataLoader(
            subset, batch_size=batch_size, shuffle=True, num_workers=4
        )

    data_loaders = DataLoaders(
        train=subset_to_loader(train_set),
        val=subset_to_loader(val_set),
        test=subset_to_loader(test_set),
    )
    return data_loaders, target_classes
```

2.3.2. 模型初始化, 训练与测试

2.3.2.1. 初始化

模型的初始化部分的代码位于 `code/models.py` 中. 该部分主要的函数为 `new_model()` 和 `load_model()`, 如下:

```
def new_model(model_name: str, num_classes: int, pretrained: bool) -> nn.Module:
    if pretrained:
        return __get_pretrained_model(model_name, num_classes)
    return __get_model(model_name, num_classes)


def load_model(model_path: str) -> nn.Module:
```

```
    model = torch.load(model_path)
    if not isinstance(model, nn.Module):
        raise ValueError(f"Model {model_path} is not an instance of nn.Module")
    return model
```

其中, 如果使用预训练参数, 则模型的输出不是二分类结果, 需要修改网络结构. 以 AlexNet 为例, 使用预训练参数和不使用预训练参数的逻辑如下:

```
# 使用预训练参数
model = alexnet(weights=AlexNet_Weights.DEFAULT) # 输出结果由预训练参数决定
model.classifier[-1] = nn.Linear(model.classifier[-1].in_features, num_classes)
# 不使用预训练参数
model = alexnet(num_classes=num_classes)
```

2.3.2.2. 训练

模型的训练和测试部分的代码位于 `code/procedures.py` 中.

在对模型的训练中, 我使用了交叉熵 `torch.nn.CrossEntropy` 作为损失函数, 并利用梯度下降优化器 `torch.optim.SGD` 来对模型参数更新. 同时, 我引入了 Early Stopping, 即在验证集下的损失不再下降时停止对模型的训练. 具体代码如下:

```
def train(
    model: nn.Module,
    n_epochs: int,
    data_loaders: DataLoaders,
    learning_rate: float,
    momentum: float,
    early_stopping: Optional[EarlyStopping],
) -> tuple[list[float], list[float]]:
    optimizer = torch.optim.SGD(
        model.parameters(), lr=learning_rate, momentum=momentum
    )
    loss_fn = nn.CrossEntropyLoss()

    device = next(model.parameters()).device

    train_losses = []
    val_losses = []

    for epoch in range(n_epochs):
        train_loss, train_acc = __train_one_epoch(
            model, data_loaders.train, loss_fn, optimizer, device
        )
        train_losses.append(train_loss)

        val_loss, val_acc = __val_one_epoch(
            model, data_loaders.val, loss_fn, device
        )
        val_losses.append(val_loss)
        # logging
        # ...
        # early stopping
        if early_stopping is None:
            continue
        early_stopping(val_loss, model)
        if early_stopping.early_stop:
```

```
            break
    return train_losses, val_losses
```

2.3.2.3. 测试

对模型的测试部分比较简单, 如下所示:

```python
def test(model: nn.Module, data_loader: DataLoader) -> tuple[float, float]:
    device = next(model.parameters()).device
    loss_fn = nn.CrossEntropyLoss()
    test_loss, test_acc = __val_one_epoch(model, data_loader, loss_fn, device)
    return test_loss, test_acc
```

2.3.3. 主程序

本实验代码的主程序为 `code/main.py`, 它会解析命令行参数, 并根据命令行参数决定执行的步骤. 命令行参数的部分可以查看 `code/arg_parser.py` 或运行 `python main.py --help`, 这里不再展开.

3. 实验结果

在 BitaHub 上使用合适的命令行参数运行实验代码, 并收集相关结果. 其中, 训练过程中的相关参数如下:

- 学习率 learning rate: 0.001
- 动量 momentum: 0.9
- 批大小 batch size: 4
- 早停止 early stopping:
  - Patience: 7, 即在连续 7 次验证集 loss 不再下降时停止训练
  - Delta: 0
- 随机种子 random seed: 42
- 训练集, 验证集, 测试集大小比例为 8:1:1
- 所有模型均使用了预训练参数

对于 AlexNet 和 ResNet 模型, 其在训练过程中的 loss 如图 3 所示.

图 3 训练过程中的 loss 曲线, 左图为训练集上 loss, 右图为验证集上 loss

可以看出, 四种模型都达到了较好的收敛.

各个模型在测试集上的结果如表 1 所示.

模型	准确率	交叉熵损失
AlexNet	88.50%	0.22436
ResNet-18	93.00%	0.13883
ResNet-34	92.50%	0.14340
ResNet-50	95.00%	0.10850

表 1 各模型在测试集上的表现

可以看到, 相比于 AlexNet, ResNet 有着更好的表现, 并且规模最大的 ResNet-50 在测试集上的表现最好.

参考文献

1. Krizhevsky, A., Sutskever, I., Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM. 60, 84 (2017). https://doi.org/10.1145/3065386
2. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks, (2014)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition, (2015)