神经网络作业

——利用 VGG16 实现图像分类

南京大学 软件学院　邢骏洋*

2024 年 11 月 29 日

1　引言

VGG16 网络是由牛津大学 Visual Geometry Group 提出的深度卷积神经网络模型，在 2014 年 ILSVRC 竞赛中取得了优异成绩。该网络以其简单且有效的架构著称，采用小型卷积核（3×3）和最大池化层的连续堆叠来构建深层网络结构。本实验基于 VGG16 网络，在 Caltech 101 数据集上展开一系列图像分类实验。

2　网络结构与基线实验

2.1　VGG16 网络架构

VGG16 网络采用简洁的层叠式架构，通过重复使用相同的基础构件模块来构建深层网络。网络结构可以分为特征提取部分和分类部分：

2.1.1　特征提取部分

特征提取网络包含 5 个卷积块，每个块的具体结构如下：

- 块 1：两个 64 通道的 3×3 卷积层，后接最大池化层
- 块 2：两个 128 通道的 3×3 卷积层，后接最大池化层
- 块 3：三个 256 通道的 3×3 卷积层，后接最大池化层
- 块 4：三个 512 通道的 3×3 卷积层，后接最大池化层
- 块 5：三个 512 通道的 3×3 卷积层，后接最大池化层

每个卷积层后都紧跟 ReLU 激活函数，最大池化层的核大小为 2×2，步长为 2。这种设计使得特征图的空间维度在每个池化层后减半。通道数逐渐增加，有助于提取更加抽象的特征表示。

*Software Institute, Nanjing University, Nanjing, China. E-mail: xingjunyang@smail.nju.edu.cn

2.1.2 分类部分

特征提取后接三层全连接层：

- 第一层：输入维度 4096，ReLU 作为激活函数
- 第二层：输入维度 4096，激活函数同上
- 输出层：输入维度为类别数（在 Caltech 101 数据集中为 101）

2.2 实现细节

在 PyTorch 框架下，我们通过以下代码实现 VGG16 网络：

Listing 1: 网络配置

```
class VGG16(nn.Module):
    def __init__(self, num_classes=101):
        super(VGG16, self).__init__()
        self.features = nn.Sequential(
            # Block 1
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # Block 2
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # [Blocks 3-5 structure remains similar]
        )

        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, num_classes)
        )
```

2.3 训练配置

基线实验与实验手册所述相近，主要包括数据预处理、优化器与训练过程配置。数据预处理包含缩放、张量化和标准化处理。使用 SGD 作为优化器，设置初始学习率为 0.01，训练 10 轮，每次数据批量大小为 32。采用 CosineAnnealingLR 调度器实现学习率动态调整。

具体的训练代码实现如下：

```
# 优化器设置
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight_decay=2e-4)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)
```

2.4 基线实验结果

在未经任何优化的基础模型上，我们观察到以下实验结果：

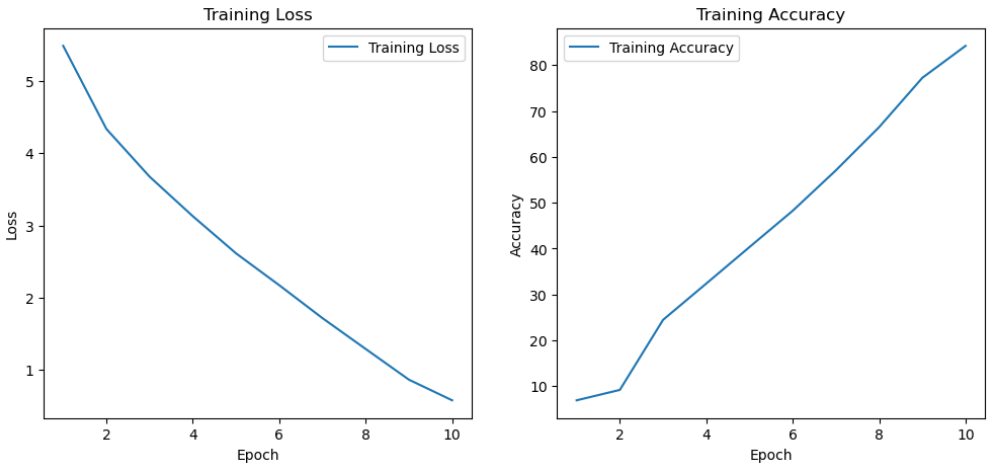

图 1: 基线实验结果

2.4.1 训练结果分析

在训练过程中，模型的损失从初始的约 4.5 逐渐下降至 0.5 左右，显示出良好的收敛趋势。训练精度在前三轮快速上升，并在随后的训练中持续提高，最终达到了约 80%。然而，测试精度仅为 57.57%，与训练精度存在明显差距，表明模型存在严重的过拟合问题。尽管似乎在 10 轮训练下，最后训练集的精度与损失尚未收敛。但由于已经发生了过拟合，进行更多轮次的训练对实验意义不大。

通过实验结果可以看出，基础模型存在模型泛化能力较差，在测试集上表现不佳的问题。主要原因是数据集规模相对较小，而模型参数量大，并且未使用数据增广等提升泛化性的技术。此外，在训练时没有使用验证集进行模型选择，可能舍弃了更好的模型。

因此，后续将基于此分析，通过一系列改进策略来提升模型性能。

3 模型改进策略

为了提升模型性能，在新的模型中实施了多项改进策略，主要包括数据增强、Dropout 正则化和早停机制。

3.1 数据增强策略

在训练过程中，我们采用了以下数据增强技术：

Listing 2: 训练数据增强转换

```
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.3),
    transforms.RandomRotation(30),
    transforms.ColorJitter(
        brightness=0.3,
        contrast=0.3,
        saturation=0.3,
        hue=0.1
    ),
    transforms.RandomPerspective(distortion_scale=0.5, p=0.5),
    transforms.RandomErasing(p=0.3),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

主要新增的数据增强策略包括：

- 随机水平翻转（RandomHorizontalFlip）：50% 概率水平翻转图像
- 颜色抖动（ColorJitter）：随机调整亮度、对比度和饱和度
- 随机旋转（RandomRotation）：在 ±15 度范围内随机旋转
- 随机透视变换（RandomPerspective）：随机应用透视变换
- 随机擦除（RandomErasing）：随机擦除图像的一部分

这些数据增强策略可以增加训练数据的多样性，提升模型的泛化能力。

3.2 Dropout 正则化

此外，在每个卷积块后的全连接层中加入了 Dropout 层，以减少模型的复杂度并防止过拟合。Dropout 层的概率设置为 0.5，共新增了 5 个 Dropout 层。

3.3 早停机制

通过将数据集继续分划为训练集和验证集，可以实现基于验证集损失的早停机制：

Listing 3: 早停实现代码

```python
# Early stopping check
if val_loss < best_val_loss:
    best_val_loss = val_loss
    best_model_wts = copy.deepcopy(model.state_dict())
    patience_counter = 0
else:
    patience_counter += 1

if patience_counter >= patience:
    print(f'Early stopping triggered after epoch {epoch+1}')
    break
```

其中，patience 参数表示在验证集损失连续 patience 次未下降时停止训练。这样，可以及时识别过拟合的情况。

此外，在早停的过程中，我们采用了模型参数的深拷贝，以保存最优模型参数。

3.4 改进效果对比

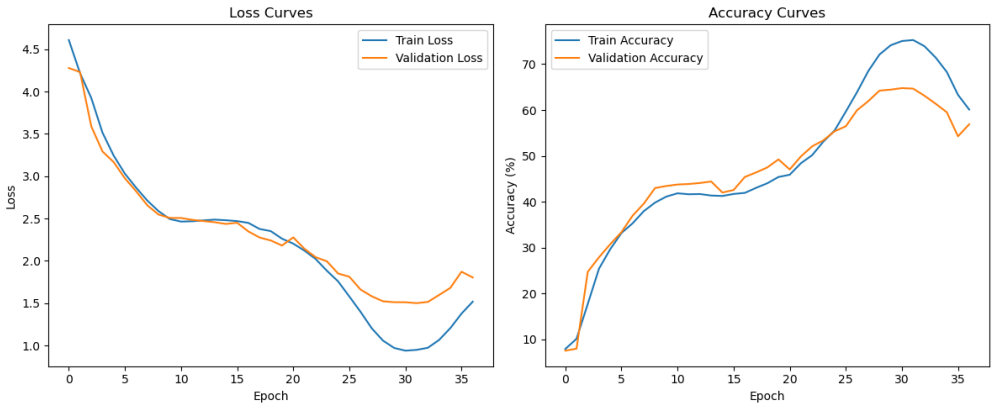

图 2: 模型改进结果

结合以上改进策略后，模型性能略有提升。最终测试集准确率达到 61.3%，比基础模型仅提升了约 3-4 个百分点。我们可以通过图像看出，早停机制起到了相应的作用。但由于数据集规模较小，训练后期训练机和验证集的差距无法缩小，模型性能提升存在瓶颈。

4 预训练模型与迁移学习

4.1 预训练模型的使用

下一步，我们采用了在 ImageNet 数据集上预训练的 VGG16 模型，并进行了针对性的微调：

Listing 4: 加载预训练模型

```
# 加载预训练模型
model = models.vgg16(weights=models.VGG16_Weights.DEFAULT)

# 冻结预训练模型的参数
if fine_tune:
        for param in model.parameters():
            param.requires_grad = False

        for param in model.features[-4:].parameters():
            param.requires_grad = True

# 修改最后的全连接层以适应新的类别数
model.classifier = nn.Sequential(
        nn.Linear(512 * 7 * 7, 4096),
        nn.ReLU(True),
        nn.Dropout(p=0.5),
        nn.Linear(4096, 2048),
        nn.ReLU(True),
        nn.Dropout(p=0.5),
        nn.Linear(2048, num_classes)
    )
```

为了更好地利用预训练权重，我们采用了分层微调策略。对于卷积层，我们冻结了前几层的参数，只对后几层进行微调。对于全连接层，我们重新定义了分类器，以适应我们实验中的类别数。

4.2 预训练模型与从零训练的对比

通过使用预训练模型，以及之前的早停、Dropout 等策略，我们观察到了显著的性能提升。预训练模型可以在 1-2 个 epoch 内达到较高的训练精度，且在验证集上能一开始就取得很好的结果。最终，我们的模型在测试集上达到了 92.79% 的准确率。这表明预训练模型在小数据集上的迁移学习效果显著。

4.3 预训练模型的优势分析

预训练模型能够取得更好效果的原因主要是：第一，预训练模型在 ImageNet 上学习到的特征具有良好的泛化性，这些特征对于 Caltech 101 数据集同样适用。第二，相比随机初始化，预训练权重提供了更好的参数起点，使模型更容易收敛到更优的解。此外，预训练模型已经形成了从低层到高层的特征提取能力，低层特征（如边缘、纹理）具有普适性，高层特征通过微调可以适应新任务。因此，预训练模型降低了对训练数据量的需求，特别适合像 Caltech 101 这样的中小型数据集。

4.4 使用 TSNE 可视化特征

使用 TSNE 算法对预训练模型提取的特征进行可视化，可以看到不同类别的特征在特征空间中的分布情况。下图展示了预训练模型提取的 10 个特征在 2 维空间的分布情况：

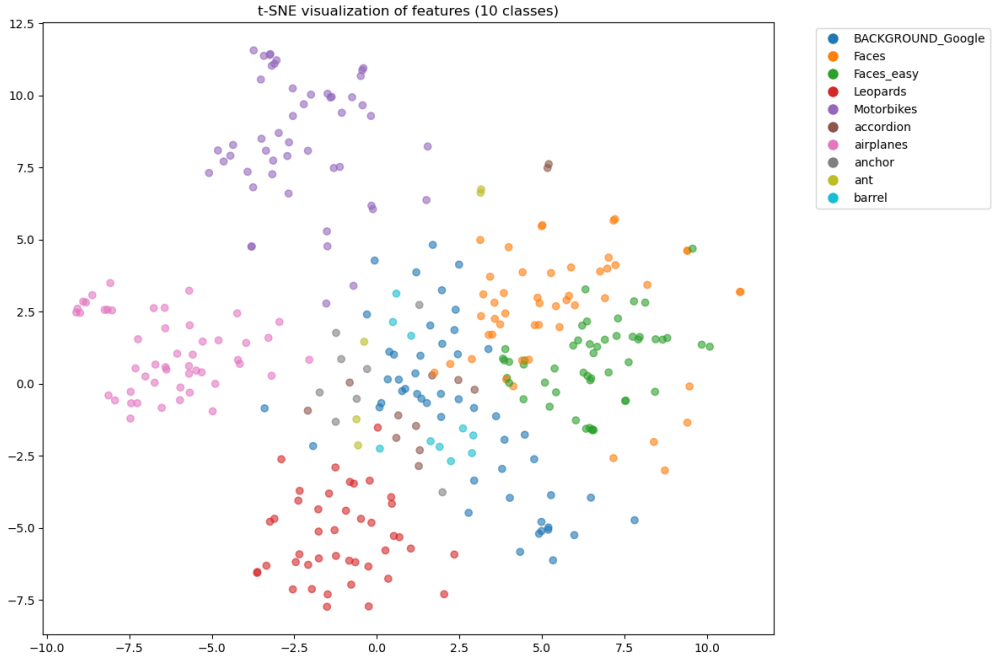

图 3: TSNE 可视化特征

可以看出，不同类别的特征在特征空间中有较好的分离性，这表明预训练模型提取的特征具有较好的可分性。

5 附录

完整代码于 `code` 文件夹中，其中包含了

- `nn_homework_baseline.ipynb`：基线实验代码
- `nn_homework_with_transform_dropout_and_earlystop.ipynb`：改进模型代码
- `nn_homework_with_pre_trained_model_and_tsne.ipynb`：预训练模型代码

所有代码使用 Jupyter Notebook 编写，需要安装 PyTorch、TorchVision 等库，并将数据集放置在 `dataset` 文件夹中。