

摘要：三叶青块根识别模型的最终训练

1. 实验准备

经过不同神经网络模型的迁移学习，再对每个模型的参数进行调整，得到最佳模型。

2. 实验步骤

2.1. 使用 mobilenet_v3_large 模型

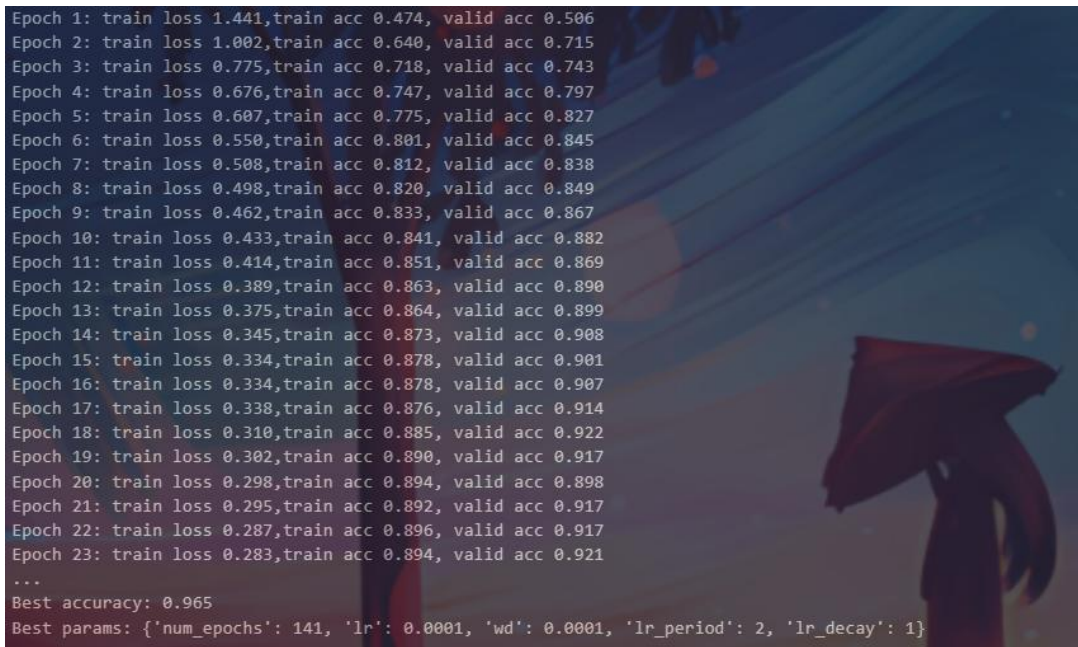
Mobilenet 系列模型是一种针对移动和嵌入式设备优化的轻量级深度学习网络结构。它主要通过使用深度可分离的卷积来降低计算复杂性，而不牺牲太多的准确性。

考虑到模型最后要部署到小程序端和 Web 端，模型文件不能太大，同时经过尝试，mobilenet_v3_large 模型在三叶青块根图像分类的训练过程中准确率较高。

2.2. 在 kaggle 上使用 GPU 训练得到模型参数文件

通过比较在每一 epoch 上的验证集的准确率，得出最大的 valid acc，同时保存对应的最佳模型。

例如，下图中最佳模型在验证集上的准确率为 0.965。



```
Epoch 1: train loss 1.441,train acc 0.474, valid acc 0.506
Epoch 2: train loss 1.002,train acc 0.640, valid acc 0.715
Epoch 3: train loss 0.775,train acc 0.718, valid acc 0.743
Epoch 4: train loss 0.676,train acc 0.747, valid acc 0.797
Epoch 5: train loss 0.607,train acc 0.775, valid acc 0.827
Epoch 6: train loss 0.550,train acc 0.801, valid acc 0.845
Epoch 7: train loss 0.508,train acc 0.812, valid acc 0.838
Epoch 8: train loss 0.498,train acc 0.820, valid acc 0.849
Epoch 9: train loss 0.462,train acc 0.833, valid acc 0.867
Epoch 10: train loss 0.433,train acc 0.841, valid acc 0.882
Epoch 11: train loss 0.414,train acc 0.851, valid acc 0.869
Epoch 12: train loss 0.389,train acc 0.863, valid acc 0.890
Epoch 13: train loss 0.375,train acc 0.864, valid acc 0.899
Epoch 14: train loss 0.345,train acc 0.873, valid acc 0.908
Epoch 15: train loss 0.334,train acc 0.878, valid acc 0.901
Epoch 16: train loss 0.334,train acc 0.878, valid acc 0.907
Epoch 17: train loss 0.338,train acc 0.876, valid acc 0.914
Epoch 18: train loss 0.310,train acc 0.885, valid acc 0.922
Epoch 19: train loss 0.302,train acc 0.890, valid acc 0.917
Epoch 20: train loss 0.298,train acc 0.894, valid acc 0.898
Epoch 21: train loss 0.295,train acc 0.892, valid acc 0.917
Epoch 22: train loss 0.287,train acc 0.896, valid acc 0.917
Epoch 23: train loss 0.283,train acc 0.894, valid acc 0.921
...
Best accuracy: 0.965
Best params: {'num_epochs': 141, 'lr': 0.0001, 'wd': 0.0001, 'lr_period': 2, 'lr_decay': 1}
```

2.3. 将模型参数文件转化为 CPU 上运行的模型文件

将得到的最佳模型参数文件从 kaggle 上下载下来，但是由于使用 GPU 训练得到的，其无法在 CPU 上运行，故要先将其从 GPU 训练得到的模型参数文件转化为 CPU 上运行的模型文件。

使用 python 代码将其转化，代码如下所示：

GPU转为CPU

```
import torch
import torchvision
from torch import nn

def get_mobilenet(device, model_name, num_classes=6):
    base_model_func = getattr(torchvision.models, model_name) # 加载预训练的MobileNetV2模型
    base_model = base_model_func(pretrained=True)
    if model_name == "mobilenet_v2":
        num_features = base_model.classifier[-1].in_features
        classifier = nn.Sequential( # 定义一个新的分类头
            nn.Linear(num_features, 256),
            nn.ReLU(),
            nn.Linear(256, num_classes)
        )
        base_model.classifier = classifier # 替换原有的分类头
    else:
        num_features = base_model.features[-1].out_channels # 获取最后一个卷积层的输出通道数
        classifier = nn.Sequential( # 定义一个新的分类头
            nn.Linear(num_features, 256),
            nn.ReLU(),
            nn.Linear(256, num_classes)
        )
        base_model.classifier = classifier # 替换原有的分类头
    base_model = base_model.to(device) # 将模型参数分配到指定设备
    for name, param in base_model.named_parameters(): # 冻结特征提取器的参数
        if 'classifier' not in name: # 确保只冻结特征提取器部分的参数
            param.requires_grad = False
    return base_model

def load_model_on_cpu(model_path, model_name):
    device = torch.device('cpu') # 创建模型
    model = get_mobilenet(device, model_name)
    state_dict = torch.load(model_path, map_location='cpu') # 加载模型参数
    state_dict = {k.replace('module.', ''): v for k, v in state_dict.items()} # 如果模型是在多GPU环境下训练的，模型参数的键会包含 'module'
    model.load_state_dict(state_dict) # 加载参数到模型
    print(model)
    return model

# 使用方法
model_path = r"D:\SanYeQing_Project\wht_sanyeqing_image-Classification\model_zheng_path\model_best_canshu_97.07.pth"
model = load_model_on_cpu(model_path, model_name="mobilenet_v3_large")
torch.save(model, 'model_cpu_97.07.pth')
```

2.4. 最终模型在预测集上的效果

加载转化得到的模型文件，在测试集上评估模型精度：在测试集上的准确率 test_acc 等于 97.07%；在前三个类别预测正确的概率 top_n 等于 99.89%。

```
{'0': '云南省', '1': '广西省', '2': '未知', '3': '浙江省', '4': '贵州省', '5': '陕西省'}
总耗时: 264.40秒
测试集图片数: 1842张
每张图片耗时photo_time: 0.143541秒
=====
预测准确率test_acc: 97.07%
top_n 预测正确的概率是 99.89%
=====
分类报告与各类别准确率
```

	precision	recall	f1-score	support
云南省	0.97	0.97	0.97	296
广西省	0.97	0.96	0.96	339
未知	1.00	1.00	1.00	310
浙江省	0.98	0.97	0.98	293
贵州省	0.96	0.97	0.96	308
陕西省	0.94	0.96	0.95	296
accuracy			0.97	1842
macro avg	0.97	0.97	0.97	1842
weighted avg	0.97	0.97	0.97	1842

```
100%|██████████| 6/6 [00:00<00:00, 652.22it/s]
```

	precision	recall	f1-score	support	accuracy
云南省	0.972789	0.966216	0.969492	296.0	0.966216
广西省	0.970149	0.958702	0.964392	339.0	0.958702
未知	1.000000	1.000000	1.000000	310.0	1.000000
浙江省	0.979381	0.972696	0.976027	293.0	0.972696
贵州省	0.958199	0.967532	0.962843	308.0	0.967532
陕西省	0.943522	0.959459	0.951424	296.0	0.959459
macro avg	0.970673	0.970768	0.970696	1842.0	0.970768
weighted avg	0.970789	0.970684	0.970712	1842.0	0.970684

3. 实验结果

得到了训练好的模型文件，得到了一系列评估指标。

