中山大學 SUN YAT-SEN UNIVERSITY

计算机视觉期中报告

院(系):智能工程学院　　学号：20354270　　姓名：张子骏

日期：2022/11/05　　实验名称：Plant Pathology 图像分类

1.任务介绍

细粒度多目标分类问题一直是图像分类中较为困难的任务。植物病理学 Plant Pathology 数据集，包含 4000 张苹果叶面病害的高质量 RGB 图像，其中包含了大量人工注释的标签。由于一个植物可能患有多种疾病，所以该任务为多标签分类问题。但由于标签中有重复特征且重复特征体现在图像上没有区别，所以可以通过一热编码（One-Hot Encoding）转化为六个二分类问题。并且由于数据集本身的细粒度特征，类间差距小而类内差别大，使得图像分类更加困难。

针对以上问题，我使用 pytorch 和 Mindspore 框架，分别搭建了 ViT 模型和 ResNet 模型（由于 ViT 模型太大，使用 ascend910 难以单卡训练，多卡过于昂贵），并对损失函数和网络架构进行了优化。

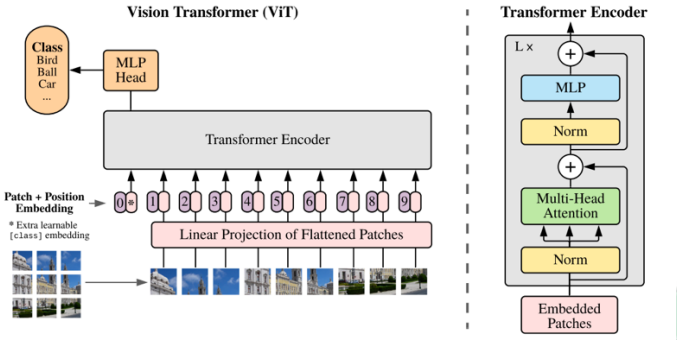
具体的训练和测试的环境和接口说明在两个文件夹（pytorch/mindspore）的 readme.md 文件（一定要看哦）中，最终得到的最优模型位于两个文件夹的 results 目录下。在 run.sh 文件中定义了四卡的分布式训练方式，如果显卡充足，可换成八卡进行训练。

2.模型介绍

2.1 ViT 模型

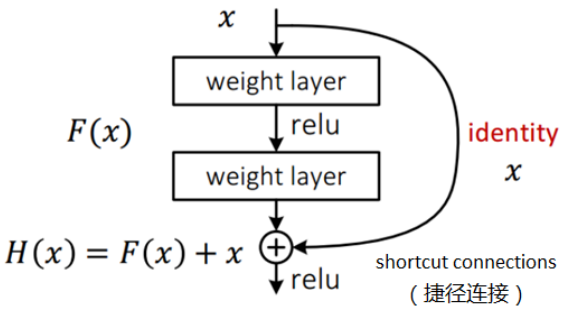
随着 Transformer 模型在 NLP 领域的大火，如何将 Transformer 模型应用于 CV 领域成为了研究的热门。其中，ViT 模型成功得将 Transformer 模型应用于 CV 领域。为解决图片的输入问题，ViT 将图片打成不同的 Patch，从而输入进 Transformer 模型的 Encoder 中，经过若干由归一化、多头注意力层、多层感知机组成的 Encoder Block 和 Decoder Block 后，最后由检测头得到最终的分类结果。

我的模型使用了 ViT 模型搭建而成，修改了其中的多头注意力层以及检测头的输出层后，得到了较为优异的结果，在不使用预训练模型的情况下，测试集上的准确率能达到近 70%。



2.2 ResNet 模型

我使用了 Mindspore 的 modelzoo 中的 ResNet18 模型，根据预期的输出，对模型的结构进行调整，改变了输出维度和部分架构。



虽然在训练效果上远不及 ViT 模型，但 ResNet 经典的架构和可理解性让其成为广受欢迎的模型。

3. 思考以及主要创新点

我的创新点主要有三个：

- 对 ViT 模型中自注意力网络进行改进，将注意力机制的计算方式改为多头注意力模型，增加了注意力头的数量，使得自注意力层可以更多的得到关于图像内的全局信息。

```
class Attention(nn.Module):
    def __init__(self,
                 dim,   # 输入token的dim
                 num_heads=8,
                 qkv_bias=False,
                 qk_scale=None,
                 attn_drop_ratio=0.,
                 proj_drop_ratio=0.):
        super(Attention, self).__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim ** -0.5
        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop_ratio)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop_ratio)
```

- 针对多目标分类任务，使用一热编码（One-Hot Encoding）完成对目标特征的编码，让复杂的十二分类问题转化为简单的六个二分类问题。
- 受到论文 DETR：End-to-End Object Detection with Transformers 的启发，针对原有计算损失函数的方式（交叉熵损失），根据 Transformer 的 Encoder/Decoder Block 不改变输入输出维度特性，对其中两个 Decoder Block 使用 detection head 得到预测结果，综合两个结果的损失，进行反向传播，提升了模型的鲁棒性。（在下文仔细描述）

4. 实验主要模块说明

4.1 One-Hot Encoding

One-Hot Encoding 是广泛应用于自然语言处理 NLP 领域的方法。对每个主要是采用 N 位状态寄存器来对 N 个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候只有一位有效。Onehot 编码是分类变量作为二进制向量的表示。

虽然数据集中数据标签总共有十二类，但由于多标签是由单标签组合而来，所以十二类标签可以由六个二分类标签组合而成。所以可以采用 One-Hot Encoding 的方法对标签进行编码。其代码如下：

```
1. label_dict = {'complex': 0, 'frog_eye_leaf_spot': 1, 'healthy': 2,'
   powdery_mildew': 3, 'rust': 4, 'scab': 5}
2. def one_hot(label):
3.     # label: <type=numpy> [1,4] 表示特征为第 2 个和第 4 个
4.     labels = np.zeros(6, dtype=np.float32)
5.     num = len(label)
6.     for i in range(num):
7.         labels[i] = 1
8.     return labels
```

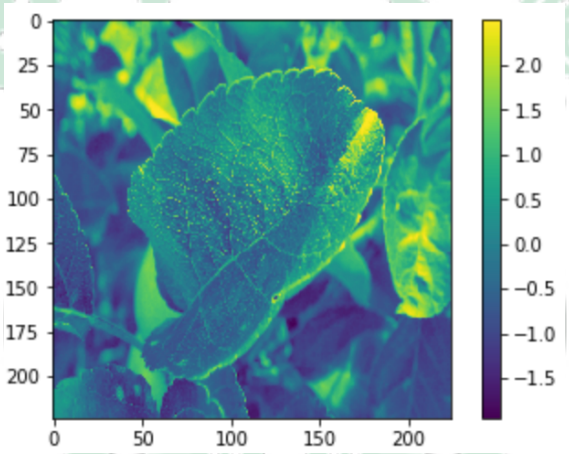
4.2 数据预处理 data_transform

由于图像具有细粒度的特征，图像的绝大部分地方均无法体现叶子生病的特征，例如硕大的叶片中仅有一小块锈块，便会被判断为 rust，如果采用 randomcrop 等会对图片进行裁剪和选择的变换方式，则会产生大量看不出病态特征的图片被标记为有病而非 healthy，这会极大影响判断的结果。

受此因素的影响，我对其仅进行大小缩放，归一化，数据类型变换、随机反转等。这样的数据增强更符合数据本身的特性。

```
1. data_transform = transforms.Compose([
2.     transforms.Resize((224,224)),
3.     transforms.RandomHorizontalFlip(p=0.5),
4.     transforms.ToTensor(),
5.     transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225]
   )
6. ])
```

经过 data_transform 之后的图片如下图所示：



4.3 数据集的构建

无论在 pytorch 中还是在 mindspore 中，数据集的构建都是类似的。都需要一个 dataset 类来完成数据的加载。

在 pytorch 中，我的 dataset 类包含三个在构造类时便会自动调用的函数，init 初始化完成对象的传入和建立。__getitem__完成对数据对象的获取和简单变换，__len__函数完成对数据集长度的获取。

```
1. class MyDataset(Dataset):
2.     def __init__(self, data_type, data_dict, data_transform, read
   _data, train_small_batch=None):
3.         super().__init__()
4.         self.train_small = train_small_batch
5.         self.data_dict = data_dict
6.         self.read_data = read_data
7.         self.data_transform = data_transform
8.         self.images, self.labels = read_data(data_type, self.data
   _dict, self.data_transform, self.train_small)
```

```
        print('read ' + str(len(self.images)) + ' ' + (data_type + ' examples'))

    def __getitem__(self, index):
        return (self.images[index], self.labels[index])

    def __len__(self):
        return len(self.images)
```

在 mindspore 中，dataset 类与在 pytorch 中类似。__next__完成对数据对象的获取和简单变换，__len__函数完成对数据集长度的获取。

4.4 损失函数 Loss

由于分类问题为多标签分类转化为的六个二分类问题，网络最终输出的预测结果维度是[batch_size, 6]维度的 tensor。所以可以使用交叉熵损失函数进行误差计算和反向传播。其类如下：

```
class SoftTargetCrossEntropy(nn.Module):

    def __init__(self):
        super(SoftTargetCrossEntropy, self).__init__()

    def forward(self, x: torch.Tensor, target: torch.Tensor) -> torch.Tensor:
        loss = torch.sum(-target * F.log_softmax(x, dim=-1), dim=-1)
        return loss.mean()
```

据此可以计算出输出和 label 的交叉熵损失，从而进行反向传播。在 Mindspore 的 ResNet 中，我是用 reduction=’mean’的 BCEloss 实现。

由于 ViT 中 Decoder 模型不改变输入输出维度的特征，我对损失函数进行了优化，如下图为 ViT 的一个 Block 的结构图：

```
(9): Block(
  (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (attn): Attention(
    (qkv): Linear(in_features=768, out_features=2304, bias=True)
    (attn_drop): Dropout(p=0.0, inplace=False)
    (proj): Linear(in_features=768, out_features=768, bias=True)
    (proj_drop): Dropout(p=0.0, inplace=False)
  )
  (drop_path): Identity()
  (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (mlp): Mlp(
    (fc1): Linear(in_features=768, out_features=3072, bias=True)
    (act): GELU()
    (fc2): Linear(in_features=3072, out_features=768, bias=True)
    (drop): Dropout(p=0.0, inplace=False)
  )
)
```

受到论文 DETR：End-to-End Object Detection with Transformers 的启发，我在中间一个 Block 后，使用 detection head 得到预测结果。和经历完整的网络后的输出相结合，对二者的 loss 进行加权，之后进行反向传播。

```
self.blocks = nn.Sequential(*[
    Block(dim=embed_dim, num_heads=num_heads, mlp_ratio=mlp_ratio, qkv_bias=qkv_bias, qk_scale=qk_scale,
          drop_ratio=drop_ratio, attn_drop_ratio=attn_drop_ratio, drop_path_ratio=dpr[i],
          norm_layer=norm_layer, act_layer=act_layer)
    for i in range(depth)
])
self.blocks1 = nn.Sequential(*[
    Block(dim=embed_dim, num_heads=num_heads, mlp_ratio=mlp_ratio, qkv_bias=qkv_bias, qk_scale=qk_scale,
          drop_ratio=drop_ratio, attn_drop_ratio=attn_drop_ratio, drop_path_ratio=dpr[i],
          norm_layer=norm_layer, act_layer=act_layer)
    for i in range(depth-1)
])
```

对 ViT 类的 forward 函数修改如下，可返回两个计算结果。从而输出，计算误差。

```
def forward(self, x):
    x, x1 = self.forward_features(x)
    if self.head_dist is not None:
        x, x_dist,x1, x1_dist = self.head(x[0]), self.head_dist(x[1]),self.head(x1[0]), self.head_dist(x1[1])
        if self.training and not torch.jit.is_scripting():
            # during inference, return the average of both classifier predictions
            return x, x_dist,x1, x1_dist
        else:
            return (x + x_dist) / 2, (x1 + x1_dist) / 2
    else:
        x1 = self.head(x1)
    return x,x1
```

4.5 训练步骤

使用 mindspore 和 pytorch 都有的 Adam 优化器，将 learning_rate 学习率设置较低。根据实验得到的反馈得知，在本实验中，learning_rate 设置的很小利于 loss 的降低。所以将其设置为 1e-5。

在 Mindspore 中设置 CheckpointConfig，callback 函数。随着训练的进行，每隔几次保存训练的权重在 results 文件夹中。

```
model.train(2000, train_set,callbacks=[ckpoint_cb, LossMonitor(per_print_times=1),eval_cb],dataset_sink_mode=dataset_sink_mode)
```

使用 train 进行训练。

在 pytorch 中，与 mindspore 不同之处在于需要自行定义模型训练的步骤。典型环节是 pytorch 反向传播“三大件”：清除梯度，反向传播，梯度更新，代码如下所示：

```
1. loss = criterion(output, labels)
2. optimizer.zero_grad() # 清除梯度
3. loss.backward() # 反向传播
4. optimizer.step() # 梯度更新
```

在保存模型时，我选择保存在验证集上取得效果最好的模型。对 eval_loss 进行判断，保留使 eval_loss 最小的模型。

5. 实验结果

5.1 ResNet 结果

```
epoch: 534 step: 93, loss is 0.008114872500300407
{'Accuracy': 0.6041666666666666}
epoch: 535 step: 93, loss is 0.003699594410136342
{'Accuracy': 0.5954861111111112}
epoch: 536 step: 93, loss is 0.0006669489666819572
{'Accuracy': 0.6319444444444444}
epoch: 537 step: 93, loss is 0.0001250770001206547
{'Accuracy': 0.5833333333333334}
epoch: 538 step: 93, loss is 0.0003483676700852811 3
{'Accuracy': 0.6076388888888888}
epoch: 539 step: 93, loss is 0.001983528956770897
{'Accuracy': 0.59375}
epoch: 540 step: 93, loss is 0.0005714065046049654
{'Accuracy': 0.5607638888888888}
epoch: 541 step: 93, loss is 0.0001068188576027751
{'Accuracy': 0.6336805555555556}
```

上图为 ResNet18 在 Mindspore 中的训练过程。最终模型位于 result 文件夹中。

```
[WARNING] MD(1029,7f10189c8740,python):2022-11-05-14:19:40.203.235 [mindspore/ccsrc/minddata/dataset/kernels/data/compose_o
p.cc:58] ComposeOp] Compose: input 'transforms'(op_list) has only 1 op. Compose is probably not needed.
[WARNING] MD(1029,7f10189c8740,python):2022-11-05-14:19:40.204.580 [mindspore/ccsrc/minddata/dataset/kernels/data/compose_o
p.cc:58] ComposeOp] Compose: input 'transforms'(op_list) has only 1 op. Compose is probably not needed.
[WARNING] MD(1029,7f10189c8740,python):2022-11-05-14:19:40.285.256 [mindspore/ccsrc/minddata/dataset/kernels/data/compose_o
p.cc:58] ComposeOp] Compose: input 'transforms'(op_list) has only 1 op. Compose is probably not needed.
{'Accuracy': 0.6597222222222222}
```

5.2 ViT 结果

```
———第 24 轮训练开始———
train: 100%|████████████████████████████████████████| 47/47 [00:22<00:00,  2.05it/s]
epoch: 23, Loss: 1.497764229774475
整体验证集上的Loss: 1.4135369062423706
整体验证集上的Acc: 0.6597222222222222
———第 25 轮训练开始———
train: 100%|████████████████████████████████████████| 47/47 [00:22<00:00,  2.07it/s]
epoch: 24, Loss: 1.2861411571502686
整体验证集上的Loss: 1.4056404829025269
整体验证集上的Acc: 0.6180555555555555
———第 26 轮训练开始———
train: 100%|████████████████████████████████████████| 47/47 [00:22<00:00,  2.07it/s]
epoch: 25, Loss: 1.476710557937622
整体验证集上的Loss: 1.3861044645309448
整体验证集上的Acc: 0.6736111111111112
```

上图为训练过程中的案例，大致能体现出训练集的训练过程。

6.思考和心得

对于细粒度的分类问题，主要难点在于类间差距小而类内差别大，使得图像分类更加困难。

首先是编码问题，由于数据集中数据标签总共有十二类，但由于多标签是由单标签组合而来，所以十二类标签可以由六个二分类标签组合而成。所以我才用了 One-Hot Encoding 进行编码。

在使用 pytorch 进行代码编写时，遇到了许许多多的问题。尤其是在多卡分布式训练时，给我造成了很大的麻烦。由于单卡 2080Ti 仅 11G 显存，我不得不使用多卡进行编写和训练。在开始时总遇到仅一个卡再跑，其他的空闲，后来在我不断的调整下，终于让四卡同时跑了起来。

在使用 mindspore 进行 resnet18 的输出时，发现输出的预测结果全是 0，loss 一直无法下降.这是由于经过了 sigmoid 函数，但阈值设置的不合理，计算损失时传播无法进行导致的。后来经过修改后终于可以输出正常的结果。

在本次作业中，经过实践，我学到了许许多多的知识。首先是熟悉了 mindspore 的框架，能够使用 mindspore 完成深度学习任务。其次，我对 pytorch 框架达到了更深的了解。可以使用多卡进行大型项目的编写。我还对经典架构进行了复现和改进，让我对 resnet 以及“一统天下”的 transformer 有了更深的了解。