日期：2022 年 11 月 5 日

成绩：

学院：智能工程学院　　课程：计算机视觉　　教师：梁小丹

专业：智能科学与技术　　姓名：王习羽　　学号：20354251

Plant Pathology-2021 分类任务

1 实验目的

1.1 学会使用 MindSpore1 进行简单卷积神经网络的开发。

1.2 学会使用 MindSpore1 进行 **Plant Pathology-2021** 数据集分类任务的训练和测试。

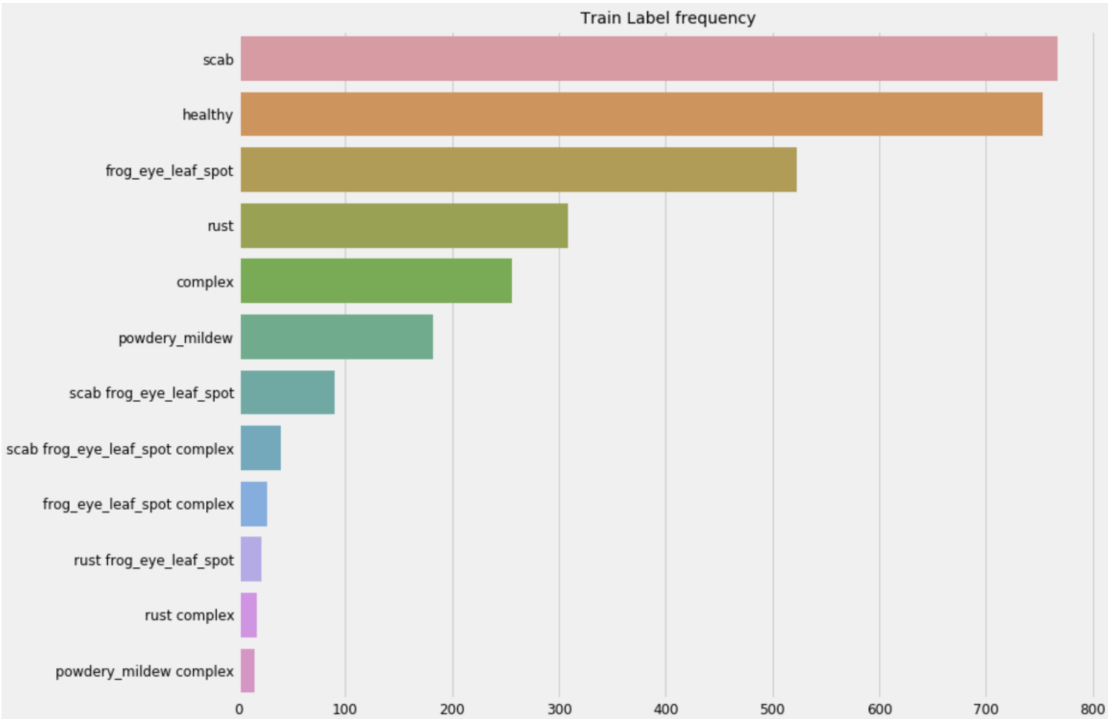
1.3 可视化学习到的特征表达器，和手工定义的特征进行分析和比较。

2 数据集预处理——one hot 编码

One hot 编码又叫独热编码，其为一位有效编码，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。One hot 编码是分类变量作为二进制向量的表示。这首先要求将分类值映射到整数值。然后，每个整数值被表示为二进制向量，除了整数的索引之外，它都是零值，它被标记为 1。

以我们需要处理的数据集 plant_dataset 为例：

训练集中原始标签的分类：



根据 one hot 编码原理将原始的 12 类标签分为 6 类，即：['complex', 'frog_eye_leaf_spot', 'healthy', 'powdery_mildew', 'rust', 'scab'].

那么原始标签为 ['scab frog_eye_leaf_spot complex'] 的新标签为 ['scab', 'frog_eye_leaf_spot' , 'complex']——>[1,1,0,0,0,1]

使用 one hot 有什么好处？

one hot 编码是将类别变量转换为机器学习算法易于利用的一种形式的过程。

这样做的好处主要有：

1.解决了分类器不好处理属性数据的问题

2.在一定程度上也起到了扩充特征的作用

直接原因：

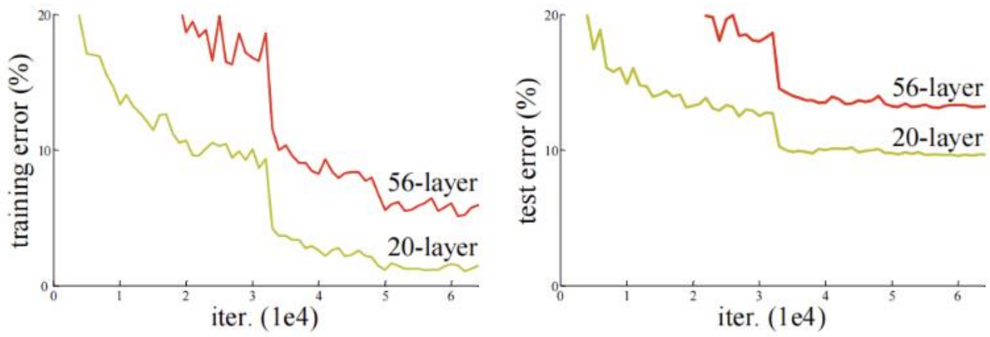
使用 One-hot 的直接原因是现在多分类 cnn 网络的输出通常是 softmax 层，而它的输出是一个概率分布，从而要求输入的标签也以概率分布的形式出现。

3 模型简述

3.1 RESNET

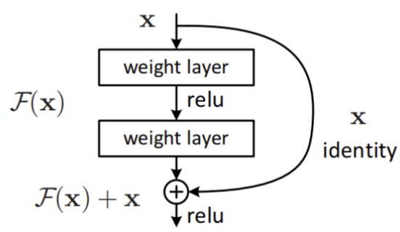
在描述这次使用的模型 SE-RESNET 之前，先来对 RESNET 做一个初步的了解。

在 ResNet 网络提出之前，传统的卷积神经网络都是将一系列的卷积层和池化层堆叠得到的，但当网络堆叠到一定深度时，就会出现退化问题，如下图所示：



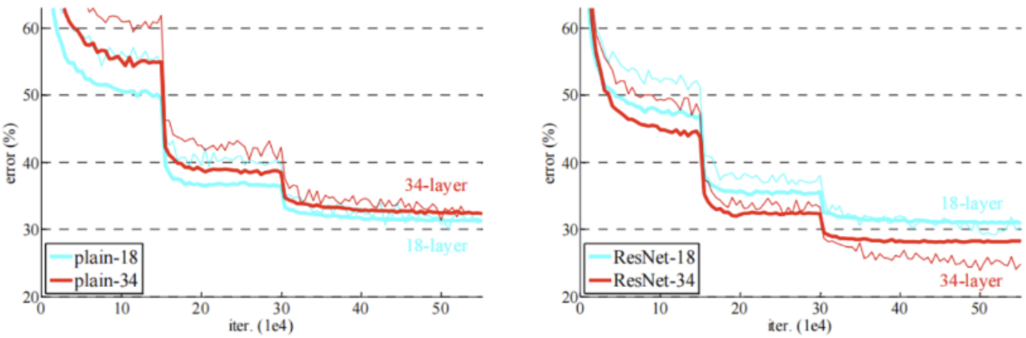
左图和右图分别是 20 层和 56 层网络在 CIFAR-10 数据集上的训练误差曲线图及测试误差曲线图，可以清晰地看出，56 层网络的训练误差和测试误差更大，而不是如预想中的“误差理应减小”。

为了解决上诉的退化问题，ResNet 网络提出了残差结构，其如下图所示：



图中输入 x，输出为 H(x)=F(x)+x，此公式可以直观地理解为输出来自两部分，一部分源于输入 x 本身，一部分源于将输入进行一系列非线性变换后的结果 F(x)。需要网络学习的部分就是 F(x)，即只需要学习输入输出差别的那一部分，简化了学习的目标。

通过将 plain network(不加残差结构)和 residual network 对比分析，发现残差网络在没有添加额外参数量的情况下性能更好，训练误差和测试误差均比 plain network 更低。

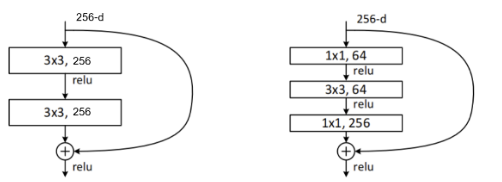


下表对比了 18 层 ResNet 和 34 层 plain network 的测试误差：

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

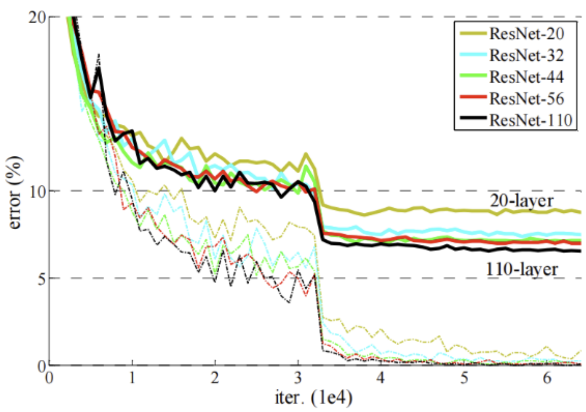
- 18 层的 ResNet 比 18 层的 plain network 性能更好，说明了残差结构的有效性。
- 34 层的 ResNet 比 18 层的 ResNet 的测试误差低，说明了通过残差结构能够很好地解决“当网络堆叠到一定深度时性能出现退化”的问题。

对于 18/34 层以及 50/101/152 层网络分别设计了两种残差块，结构图如下：



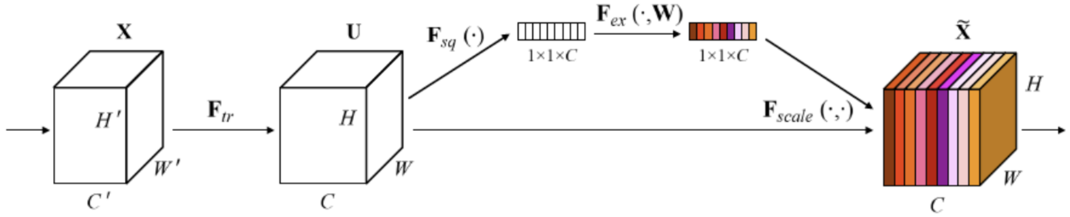
左图通过堆叠 2 个 3×3 卷积层实现残差函数，考虑到训练更深的网络所需付出的训练时间，又设计了一种瓶颈(Bottleneck)结构用于训练 ResNet50/101/512，在瓶颈结构中残差函数通过堆叠 1×1、3×3、1×1 卷积层得以实现，其中 1×1 主要起到调整维数的作用。即使右图结构层数比左图多，但是这两种不同残差块有着相近的时间复杂度，且在输入维度相同的情况下，右图的参数量比左图小。

使用不同深度的 ResNet 网络在 CIFAR-10 数据集上的训练误差与测试误差图如下图所示，图中虚线表示训练误差，实线表示测试误差。由图中数据可以看出，ResNet 网络层数越深，其训练误差和测试误差越小。



3.2 SE-RESNET

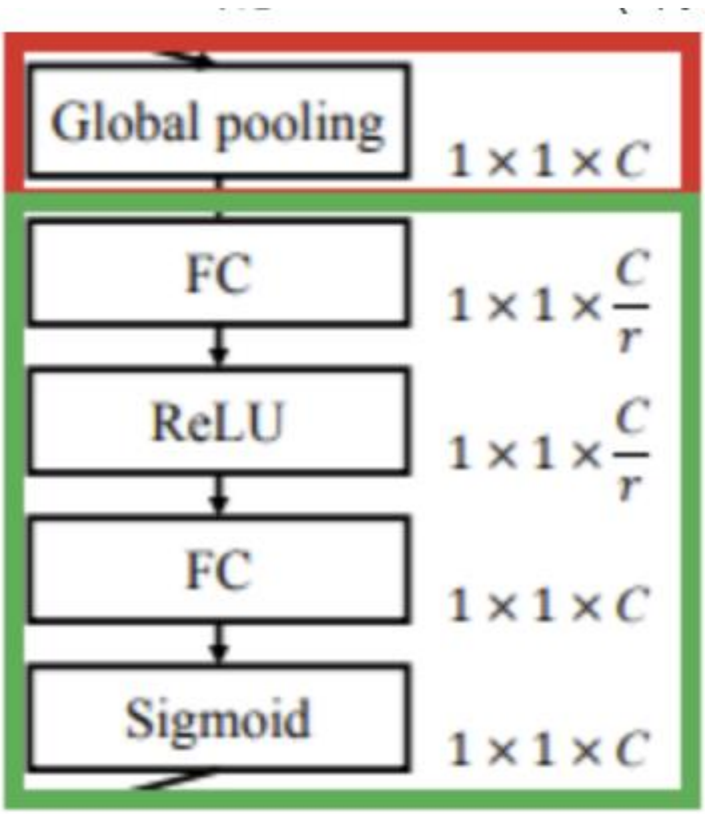
SE：Squeeze-and-Excitation 的缩写，特征压缩与激发的意思。可以把 SENet 看成是 channel-wise 的 attention，可以嵌入到含有 skip-connections 的模块中，ResNet ,VGG, Inception 等等。



Squeeze: 如下图的红框。把每个 input feature map 的 spatial dimension 从 H * W squeeze 到 1。一般是通过 global average pooling 完成的，Squeeze 操作，我们顺着空间维度来进行特征压缩，将每个二维的特征通道变成一个实数，这个实数某种程度上具有全局的感受野，并且输出的维度和输入的特征通道数相匹配。它表征着在特征通道上响应的全局分布，而且使得靠近输入的层也可以获得全局的感受野，这一点在很多任务中都是非常有用的。

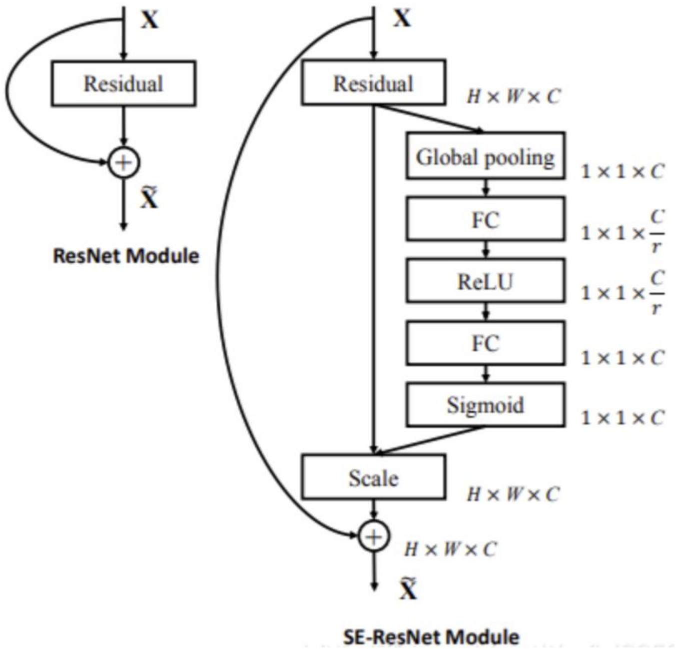
$$z_c = F_{sq}(u_c) = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W} u_c(i,j)$$

Excitation: 如下图的绿框。通过一个 bottleneck 结构来捕捉 channel 的 inter-dependency，从而学到 channel 的 scale factor(或者说是 attention factor) 。



Reweight 的操作:将 Excitation 的输出的权重看做是特征选择后的每个特征通道的重要性，然后通过乘法逐通道加权到先前的特征上，完成在通道维度上的对原始特征的重标定。即实现 attention 机制。

在 resnet 中加入 SE:下图是 SE-ResNet，可以看到 SE module 被 apply 到了 residual branch 上。我们首先将特征维度降低到输入的 1/r，然后经过 ReLu 激活后再通过一个 Fully Connected 层升回到原来的维度。这样做比直接用一个 Fully Connected 层的好处在于：1）具有更多的非线性，可以更好地拟合通道间复杂的相关性；2）极大地减少了参数量和计算量。然后通过一个 Sigmoid 的门获得 01 之间归一化的权重，最后通过一个 Scale 的操作来将归一化后的权重加权到每个通道的特征上。在 Addition 前对分支上 Residual 的特征进行了特征重标定。如果对 Addition 后主支上的特征进行重标定，由于在主干上存在 01 的 scale 操作，在网络较深 BP 优化时就会在靠近输入层容易出现梯度消散的情况，导致模型难以优化。



4 模型代码

```
""" MODEL:SE-RESNET """

import mindspore.nn as nn
import mindspore.ops.operations as P
from mindspore.common import dtype as mstype



conv_weight_init = 'HeUniform'


class GroupConv(nn.Cell):
    """
    group convolution operation.

    Args:
        in_channels (int): Input channels of feature map.
```

```
        out_channels (int): Output channels of feature map.
        kernel_size (int): Size of convolution kernel.
        stride (int): Stride size for the group convolution layer.

    Returns:
        tensor, output tensor.
    """
    def __init__(self, in_channels, out_channels, kernel_size,
                 stride, pad_mode="pad", padding=0, group=1,
has_bias=False):
        super(GroupConv, self).__init__()
        assert in_channels % group == 0 and out_channels % group ==
0
        self.group = group
        self.convs = nn.CellList()
        self.op_split = P.Split(axis=1, output_num=self.group)
        self.op_concat = P.Concat(axis=1)
        self.cast = P.Cast()
        for _ in range(group):
            self.convs.append(nn.Conv2d(in_channels//group,
out_channels//group,
                                        kernel_size=kernel_size,
stride=stride, has_bias=has_bias,
                                        padding=padding,
pad_mode=pad_mode, group=1, weight_init=conv_weight_init))

    def construct(self, x):
        features = self.op_split(x)
        outputs = ()
        for i in range(self.group):
            outputs = outputs +
(self.convs[i](self.cast(features[i], mstype.float32)),)
        out = self.op_concat(outputs)
        return out


class SEModule(nn.Cell):
    """
    SEModule
    """
    def __init__(self, channels, reduction):
        super(SEModule, self).__init__()
        self.avg_pool = P.ReduceMean(keep_dims=True)
```

```
        self.fc1 = nn.Conv2d(in_channels=channels,
out_channels=channels // reduction, kernel_size=1,
                              pad_mode='pad', padding=0, has_bias=True,
weight_init=conv_weight_init)
        self.relu = nn.ReLU()
        self.fc2 = nn.Conv2d(in_channels=channels // reduction,
out_channels=channels, kernel_size=1,
                              pad_mode='pad', padding=0,
has_bias=False, weight_init=conv_weight_init)
        self.sigmoid = nn.Sigmoid()

    def construct(self, x):
        """
        construct
        """
        module_input = x
        x = self.avg_pool(x, (2, 3))
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return module_input * x


class Bottleneck(nn.Cell):
    """
    Base class for bottlenecks that implements `forward()` method.
    """
    def construct(self, x):
        """
        construct
        """
        residual = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)
```

```python
        if self.downsample is not None:
            residual = self.downsample(x)

        out = self.se_module(out) + residual
        out = self.relu(out)

        return out


class SEBottleneck(Bottleneck):
    """
    Bottleneck for SENet154.
    """
    expansion = 4

    def __init__(self, inplanes, planes, group, reduction,
stride=1,
                 downsample=None):
        super(SEBottleneck, self).__init__()
        self.conv1 = nn.Conv2d(inplanes, planes * 2, kernel_size=1,
has_bias=False, weight_init=conv_weight_init)
        self.bn1 = nn.BatchNorm2d(planes * 2)
        self.conv2 = GroupConv(planes * 2, planes * 4,
kernel_size=3, pad_mode='pad',
                               stride=stride, padding=1, group=group,
                               has_bias=False)
        self.bn2 = nn.BatchNorm2d(planes * 4)
        self.conv3 = nn.Conv2d(planes * 4, planes * 4,
kernel_size=1,
                               has_bias=False,
weight_init=conv_weight_init)
        self.bn3 = nn.BatchNorm2d(planes * 4)
        self.relu = nn.ReLU()
        self.se_module = SEModule(planes * 4, reduction=reduction)
        self.downsample = downsample
        self.stride = stride


class SEResNetBottleneck(Bottleneck):
    """
    ResNet bottleneck with a Squeeze-and-Excitation module. It
follows Caffe
```

```
    implementation and uses `stride=stride` in `conv1` and not in
`conv2`
    (the latter is used in the torchvision implementation of
ResNet).
    """
    expansion = 4

    def __init__(self, inplanes, planes, group, reduction,
stride=1,
                 downsample=None):
        super(SEResNetBottleneck, self).__init__()
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1,
has_bias=False,
                               stride=stride,
weight_init=conv_weight_init)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
pad_mode='pad', padding=1,
                               group=group, has_bias=False,
weight_init=conv_weight_init)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, planes * 4, kernel_size=1,
has_bias=False, weight_init=conv_weight_init)
        self.bn3 = nn.BatchNorm2d(planes * 4)
        self.relu = nn.ReLU()
        self.se_module = SEModule(planes * 4, reduction=reduction)
        self.downsample = downsample
        self.stride = stride


class SEResNeXtBottleneck(Bottleneck):
    """
    ResNeXt bottleneck type C with a Squeeze-and-Excitation
module.
    """
    expansion = 4

    def __init__(self, inplanes, planes, group, reduction,
stride=1,
                 downsample=None, base_width=4):
        super(SEResNeXtBottleneck, self).__init__()
        width = int(planes * (base_width / 64.0)) * group
        self.conv1 = nn.Conv2d(inplanes, width, kernel_size=1,
has_bias=False,
```

```
                                stride=1, weight_init=conv_weight_init)
        self.bn1 = nn.BatchNorm2d(width)
        self.conv2 = GroupConv(width, width, kernel_size=3,
stride=stride, pad_mode='pad',
                                padding=1, group=group, has_bias=False)
        self.bn2 = nn.BatchNorm2d(width)
        self.conv3 = nn.Conv2d(width, planes * 4, kernel_size=1,
has_bias=False, weight_init=conv_weight_init)
        self.bn3 = nn.BatchNorm2d(planes * 4)
        self.relu = nn.ReLU()
        self.se_module = SEModule(planes * 4, reduction=reduction)
        self.downsample = downsample
        self.stride = stride


class SENet(nn.Cell):
    """
    SENet.
    """
    def __init__(self, block, layers, group, reduction,
dropout_p=0.2,
              inplanes=128, input_3x3=True,
downsample_kernel_size=3,
              downsample_padding=1, num_classes=1000):

        super(SENet, self).__init__()
        self.inplanes = inplanes
        if input_3x3:
            layer0_modules = [
                nn.Conv2d(in_channels=3, out_channels=64,
kernel_size=3, stride=2, pad_mode='pad',
                          padding=1, has_bias=False,
weight_init=conv_weight_init),
                nn.BatchNorm2d(num_features=64, momentum=0.9),
                nn.ReLU(),
                nn.Conv2d(in_channels=64, out_channels=64,
kernel_size=3, stride=1, pad_mode='pad',
                          padding=1, has_bias=False,
weight_init=conv_weight_init),
                nn.BatchNorm2d(num_features=64, momentum=0.9),
                nn.ReLU(),
                nn.Conv2d(in_channels=64, out_channels=inplanes,
kernel_size=3, stride=1,
```

```
                         pad_mode='pad', padding=1, has_bias=False,
weight_init=conv_weight_init),
                nn.BatchNorm2d(num_features=inplanes, momentum=0.9),
                nn.ReLU(),
            ]
        else:
            layer0_modules = [
                nn.Conv2d(in_channels=3, out_channels=inplanes,
kernel_size=7, stride=2, pad_mode='pad',
                         padding=3, has_bias=False,
weight_init=conv_weight_init),
                nn.BatchNorm2d(num_features=inplanes, momentum=0.9),
                nn.ReLU(),
            ]
        layer0_modules.append(nn.MaxPool2d(kernel_size=3, stride=2,
pad_mode='same'))
        self.layer0 = nn.SequentialCell(layer0_modules)
        self.layer1 = self._make_layer(
            block,
            planes=64,
            blocks=layers[0],
            group=group,
            reduction=reduction,
            downsample_kernel_size=1,
            downsample_padding=0
        )
        self.layer2 = self._make_layer(
            block,
            planes=128,
            blocks=layers[1],
            stride=2,
            group=group,
            reduction=reduction,
            downsample_kernel_size=downsample_kernel_size,
            downsample_padding=downsample_padding
        )
        self.layer3 = self._make_layer(
            block,
            planes=256,
            blocks=layers[2],
            stride=2,
            group=group,
            reduction=reduction,
            downsample_kernel_size=downsample_kernel_size,
```

```
            downsample_padding=downsample_padding
        )
        self.layer4 = self._make_layer(
            block,
            planes=512,
            blocks=layers[3],
            stride=2,
            group=group,
            reduction=reduction,
            downsample_kernel_size=downsample_kernel_size,
            downsample_padding=downsample_padding
        )
        self.avg_pool = nn.AvgPool2d(kernel_size=7, stride=1,
pad_mode='valid')
        self.dropout = nn.Dropout(keep_prob=1.0 - dropout_p) if
dropout_p is not None else None
        self.last_linear = nn.Dense(in_channels=512 *
block.expansion, out_channels=num_classes, has_bias=False)

    def _make_layer(self, block, planes, blocks, group, reduction,
stride=1,
                    downsample_kernel_size=1, downsample_padding=0):
        """
        _make_layer
        """
        downsample = None
        if stride != 1 or self.inplanes != planes *
block.expansion:
            downsample = nn.SequentialCell([
                nn.Conv2d(in_channels=self.inplanes,
out_channels=planes * block.expansion,
                          kernel_size=downsample_kernel_size,
stride=stride, pad_mode='pad',
                          padding=downsample_padding, has_bias=False,
weight_init=conv_weight_init),
                nn.BatchNorm2d(num_features=planes *
block.expansion, momentum=0.9),
            ])

        layers = []
        layers.append(block(self.inplanes, planes, group,
reduction, stride,
                            downsample))
        self.inplanes = planes * block.expansion
```

```
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes, group,
reduction))

        return nn.SequentialCell([*layers])

    def features(self, x):
        """
        features
        """
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x

    def logits(self, x):
        """
        logits
        """
        x = self.avg_pool(x)
        if self.dropout is not None:
            x = self.dropout(x)
        x = P.Reshape()(x, (P.Shape()(x)[0], -1,))
        x = self.last_linear(x)
        return x

    def construct(self, x):
        """
        construct
        """
        x = self.features(x)
        x = self.logits(x)
        return x


def senet154(num_classes=1000):
    model = SENet(SEBottleneck, [3, 8, 36, 3], group=64,
reduction=16,
                  dropout_p=0.2, num_classes=num_classes)
    return model
```

```python
def se_resnet50(num_classes=1000):
    model = SENet(SEResNetBottleneck, [3, 4, 6, 3], group=1,
reduction=16,
                  dropout_p=None, inplanes=64, input_3x3=False,
                  downsample_kernel_size=1, downsample_padding=0,
                  num_classes=num_classes)
    return model


def se_resnet101(num_classes=1000):
    model = SENet(SEResNetBottleneck, [3, 4, 23, 3], group=1,
reduction=16,
                  dropout_p=None, inplanes=64, input_3x3=False,
                  downsample_kernel_size=1, downsample_padding=0,
                  num_classes=num_classes)
    return model


def se_resnet152(num_classes=1000):
    model = SENet(SEResNetBottleneck, [3, 8, 36, 3], group=1,
reduction=16,
                  dropout_p=None, inplanes=64, input_3x3=False,
                  downsample_kernel_size=1, downsample_padding=0,
                  num_classes=num_classes)
    return model


def se_resnext50_32x4d(num_classes=1000):
    model = SENet(SEResNeXtBottleneck, [3, 4, 6, 3], group=32,
reduction=16,
                  dropout_p=None, inplanes=64, input_3x3=False,
                  downsample_kernel_size=1, downsample_padding=0,
                  num_classes=num_classes)
    return model


def se_resnext101_32x4d(num_classes=1000):
    model = SENet(SEResNeXtBottleneck, [3, 4, 23, 3], group=32,
reduction=16,
                  dropout_p=None, inplanes=64, input_3x3=False,
                  downsample_kernel_size=1, downsample_padding=0,
                  num_classes=num_classes)
    return model
```

```
if __name__ == "__main__":
    import mindspore
    model = se_resnext50_32x4d(num_classes=5)
    input = mindspore.numpy.rand(1, 3, 512, 512)
    #output, low_level_feat = model(input)
    output = model(input)
    print('SE-RESNET output.shape() :  ',output.shape) # (1, 2048,
64, 64)
    print("SE-RESNET OK!")
```

5 训练代码

```
"""" TRAINING """
#导入相关库
from mindspore.train import Model
from mindspore import context
context.set_context(mode=context.GRAPH_MODE, device_target="GPU")
from mindvision.engine.callback import ValAccMonitor
import mindspore as ms
from mindspore import ops
import mindspore.nn as nn
from model import se_resnext50_32x4d
from dataset_transforms import create_dataset
import pandas as pd
from sklearn.model_selection import KFold
from PIL import Image

net_loss = nn.MultiClassDiceLoss(weights=None,
ignore_indiex=None, activation="softmax")

#交叉验证
class KF_Dataset():

    def __init__(self, csv,spilt='train'):
        super(KF_Dataset, self).__init__()
        self.train = csv
        self.spilt = spilt
        self.imgs = self.train['images'].values
        self.labels = self.train.drop(['images'], axis=1).values


    def __getitem__(self, index):
        if self.spilt == 'train':
```

```
            img =
Image.open('./plant_dataset/train/images/'+self.imgs[index]).conv
ert('RGB')
        else:
            img =
Image.open('./plant_dataset/test/images/'+self.imgs[index]).conve
rt('RGB')
        return img, self.labels[index]

    def __len__(self):
        return len(self.imgs)

kf = KFold(n_splits=5, shuffle=True, random_state=2022)
train_path = './plant_dataset/train'
train = pd.read_csv(train_path + '/train_label.csv')
train_index = []
val_index = []
for train_index_, val_index_ in kf.split(train):
    train_index.append(train_index_)
    val_index.append(val_index_)
train_set = []
val_set = []
for i in range(5):

train_set.append(KF_Dataset(train.iloc[train_index[i]],spilt='tra
in'))

val_set.append(KF_Dataset(train.iloc[val_index[i]],spilt='train')
)

def main():

 def train_main(model, dataset, loss_fn, optimizer):
# Define forward function
    def forward_fn(data, label):
        logits = model(data)
        loss = loss_fn(logits, label)
        return loss, logits

    # Get gradient function
    grad_fn = ops.value_and_grad(forward_fn, None,
optimizer.parameters, has_aux=True)

    # Define function of one-step training
```

```
    def train_step(data, label):
        (loss, _), grads = grad_fn(data, label)
        loss = ops.depend(loss, optimizer(grads))
        return loss

    size = dataset.get_dataset_size()
    model.set_train()
    for batch, (data, label) in
enumerate(dataset.create_tuple_iterator()):
        loss = train_step(data, label)

        if batch % 100 == 0:
            loss, current = loss.asnumpy(), batch
            print(f"loss: {loss:>7f}\n")


    se_resnext = se_resnext50_32x4d(num_classes=5)
    dataset_train = create_dataset(train_set[0], batch_size=128,
target='train', image_size=224)
    dataset_val = create_dataset(val_set[0], batch_size=128,
target='val', image_size=224)
    net_opt = nn.Adam(se_resnext.trainable_params(),
learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08,
loss_scale=1.0)
    epochs = 50
    for t in range(epochs):
        print(f"Epoch {t+1}\n")
        train_main(se_resnext, dataset_train, net_loss, net_opt)
    print("--------Successfully Trained!--------")

    se_resnext = se_resnext50_32x4d(num_classes=5)
    dataset_train = create_dataset(train_set[1], batch_size=128,
target='train', image_size=224)
    dataset_val = create_dataset(val_set[1], batch_size=128,
target='val', image_size=224)
    net_opt = nn.Adam(se_resnext.trainable_params(),
learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08,
loss_scale=1.0)
    epochs = 50
    for t in range(epochs):
        print(f"Epoch {t+1}\n")
        train_main(se_resnext, dataset_train, net_loss, net_opt)
    print("--------Successfully Trained!--------")
```

```
    dataset_train = create_dataset(train_set[2], batch_size=128,
target='train', image_size=224)
    dataset_val = create_dataset(val_set[2], batch_size=128,
target='val', image_size=224)
    net_opt = nn.Adam(se_resnext.trainable_params(),
learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08,
loss_scale=1.0)
    epochs = 50
    for t in range(epochs):
        print(f"Epoch {t+1}\n")
        train_main(se_resnext, dataset_train, net_loss, net_opt)
    print("--------Successfully Trained!--------!")

    se_resnext = se_resnext50_32x4d(num_classes=5)
    dataset_train = create_dataset(train_set[3], batch_size=128,
target='train', image_size=224)
    dataset_val = create_dataset(val_set[3], batch_size=128,
target='val', image_size=224)
    net_opt = nn.Adam(se_resnext.trainable_params(),
learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08,
loss_scale=1.0)
    epochs = 50
    for t in range(epochs):
        print(f"Epoch {t+1}\n")
        train_main(se_resnext, dataset_train, net_loss, net_opt)
        print("--------Successfully Trained!--------!")

    se_resnext = se_resnext50_32x4d(num_classes=5)
    dataset_train = create_dataset(train_set[4], batch_size=128,
target='train', image_size=224)
    dataset_val = create_dataset(val_set[4], batch_size=128,
target='val', image_size=224)
    net_opt = nn.Adam(se_resnext.trainable_params(),
learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08,
loss_scale=1.0)
    epochs = 50
    for t in range(epochs):
        print(f"Epoch {t+1}\n")
        train_main(se_resnext, dataset_train, net_loss, net_opt)
    print("--------Successfully Trained!--------!")

    se_resnext = se_resnext50_32x4d(num_classes=5)
    param_dict = ms.load_checkpoint("trained_model_param.ckpt")
```

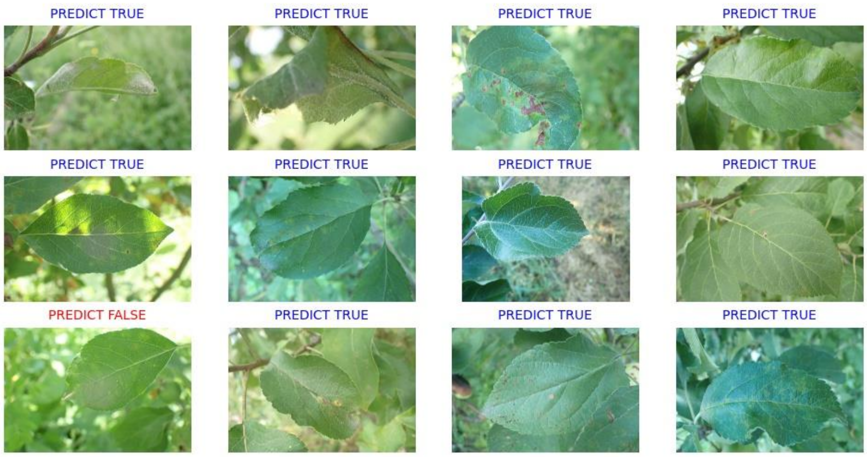
```
    param_not_load = ms.load_param_into_net(se_resnext,
param_dict)
    if param_not_load==[]:
        print("------Successfully Saved Checkpoints!------")


if __name__ == "__main__":
    main()
```

6 可视化结果

在测试集上测试训练好的模型，得到一下结果：

```
TESTING ACCURACY: 89.3%
TESTING LOSS: 0.082288
```



由结果可知训练得到的模型在测试集上有一个不错的结果。

7 实验心得与体会

由于 mindspore 1.7-1.8 的 API 发生了大部分的变化，在完成作业时确实遇到了一点困难，但是在这次作业中也对 RESNET 和 SE-RESNET 网络更加了解，希望在接下来的学习中能加深对计算机视觉的理解。