实验三：Plant Seedlings Classification

120L021716 蔡泽栋 120L021822 黄一洋

一、实验环境配置

本次实验使用的是本机的GPU,英伟达3060显卡,安装了合适的$pytorch$和$cuda$版本。

二、前置知识

1. VGG_{11} Architecture [1]

1.1 VGG block

经典卷积神经网络的基本组成部分是下面的这个**序列**：

1. 带填充以保持分辨率的卷积层
2. 非线性激活函数，如$ReLU$
3. 汇聚层，如最大汇聚层

而一个VGG块与之类似，由一系列卷积层组成，后面再加上用于空间下采样的最大汇聚层。在最初的VGG论文中$(Simonyan and Zisserman, 2014)$，作者使用了带有$3 \times 3$卷积核、填充为$1$（保持高度和宽度）的卷积层

和带有2×2汇聚窗口、步幅为2（每个块后的分辨率减半）的最大汇聚层。

1.2 Network details

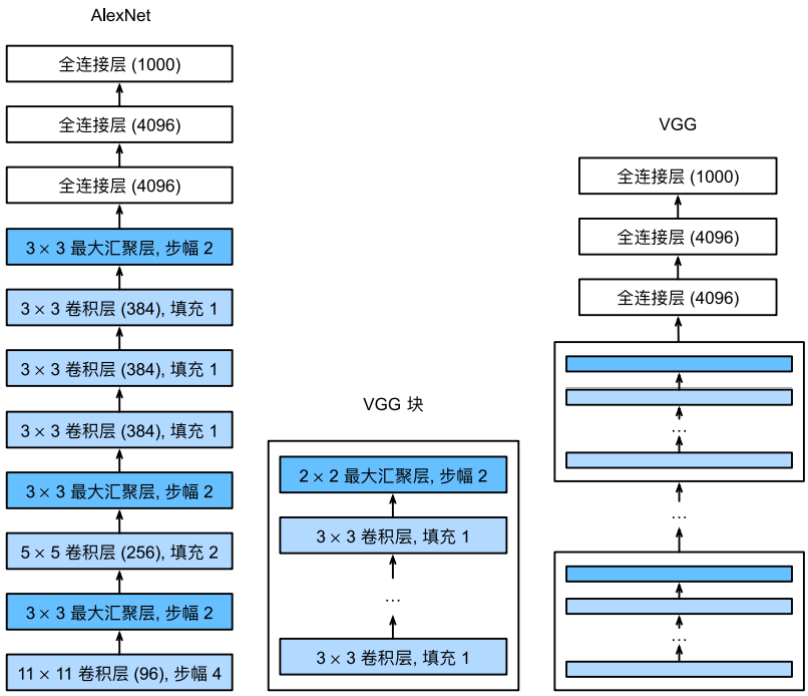


图1 VGG网络的架构

与$AlexNet$、$LeNet$一样，VGG网络可以分为两部分：第一部分主要由卷积层和汇聚层组成，第二部分由全连接层组成，如上图1 [1] 所示。

1.3 Add BatchNormalization in it

在$ResNet$的学习中，我们可以得到一些比较好用的结构，比如说$Batch\ Normalization$层。BN层的公式详见公式(1),至于BN层为什么会帮助我们训练更深的网络我们可以从下图2[2]（*Source from Stanford cs231n*)中得到解释,即将我们的数据分布发生变化，假如我们面临的是一个二分类问题，则左侧的直线斜率如果有一个较小的改动则会对很多分类结果造成影响，即不易收敛，但是我们归一化后数据分布就转化成了右侧的这种形式，在某种程度上便更加容易收敛。

$$BN(x) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta. \tag{1}$$

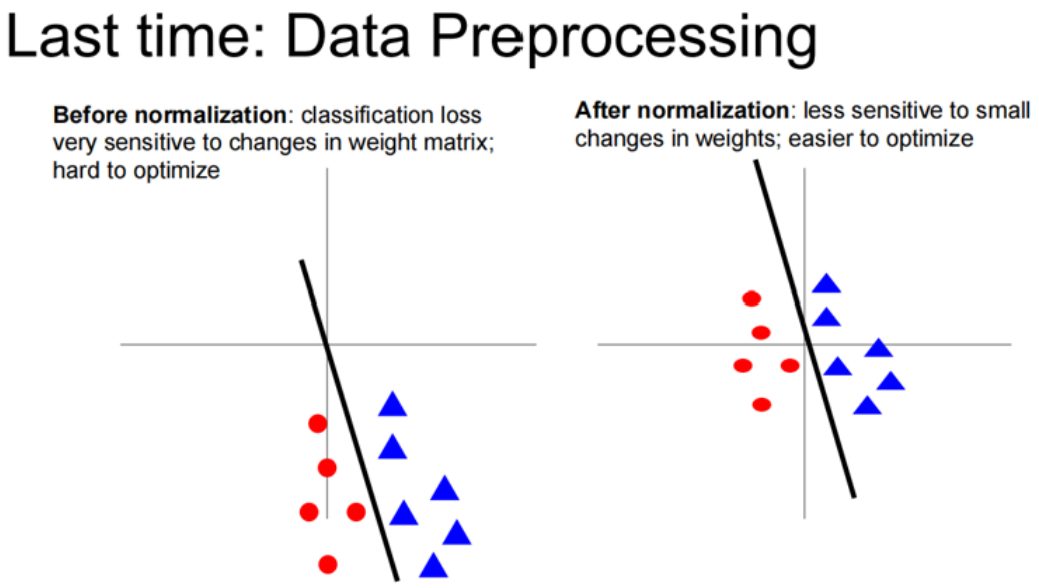


图2 BN层的作用

这部分是在最后实现网络时所做的修改，一定程度上减少了网络的训练难度。

1.4 $Xavier$初始化

假设输入层有n个权重，那么我们可以使用高斯分布来对权重就行赋值（均值为0方差为X），我们考虑下面的方程：

$$\mathrm{var}(y) = \mathrm{var}(w_1x_1 + w_2x_2 + \ldots w_nx_n + b) \tag{2}$$

对于每一个网络中的隐层，我们**希望每一层的输出方差能够固定**，因为这样能够防止我们的信号变得很大/直接消失。换句话来说，**我们需要一种权重初始化的手段，使得输入、输出的方差保持不变，**这就是$Xavier$初始化做的事。

$Xavier$初始化的作用还是很大的，它能很好的保证网络的收敛性，使得训练更加容易，我起初的实现里就是没有加这个初始化，导致网络收敛的比较差。

2. $ResNet$ Architecture

ResNet 用来构建近似恒等映射来解决之前出现的梯度消失现象。在ResNet18当中，每两个`Residual block`组成layer。对于每一个BasicBlock，输入数据分成两条路，一条路经过3*3 卷积，另一条路直接短接，二者相加并通过ReLU输出。

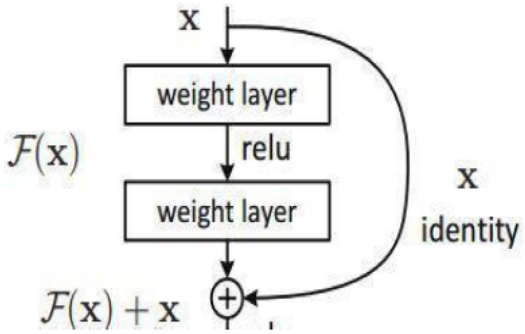


图3 ResNet中BasicBlock的结构

3. $ResNet+SE$ Architecture

Squeeze部分：压缩部分。原始feature map 的维度为$H*W*C$,经过Squeeze 将齐压缩为$1*1*C$。

Excitation部分：得到Squeeze 的$1*1*C$的表示后，加入一个FC 全连接层(Fully Connected)，对每个通道的重要性进行预测，得到不同channel 的重要性大小后再作用（激励）到之前的feature map的对应channel 上，再进行后续操作。

通过SENet block 插入到现有的多种分类网络中，希望显示地建模特征通道之间的相互依赖关系。通过学习的方式来自动获取到每个特征通道的重要程度，然后依照这个重要程度去提升有用的特征并抑制对当前任务用处不大的特征。将SE 模块嵌入到ResNet 的模型当中，示意图如下：

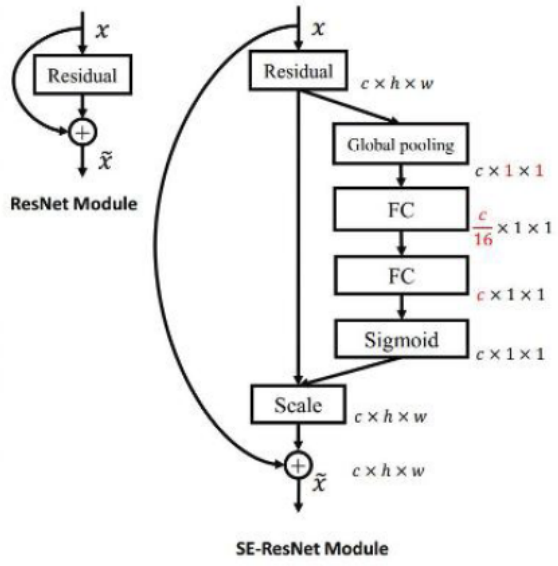
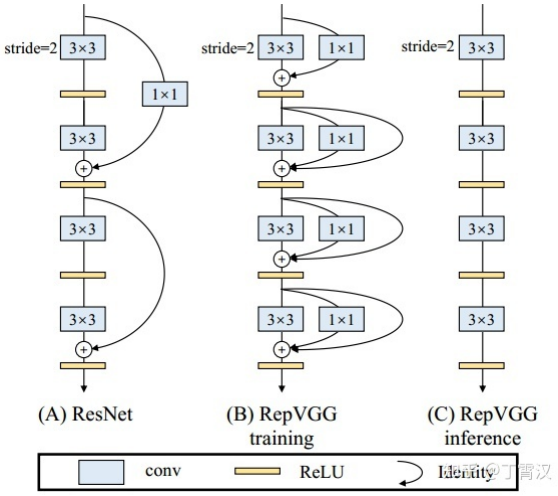


图4 ResNet中SEBlock的结构

4. $RepVGG$（性能调优自选部分）



RepVGG: Making VGG-style ConvNets Great Again

Xiaohan Ding [1*] Xiangyu Zhang [2] Ningning Ma [3]
Jungong Han [4] Guiguang Ding [1†] Jian Sun [2]
[1] Beijing National Research Center for Information Science and Technology (BNRist);
School of Software, Tsinghua University, Beijing, China
[2] MEGVII Technology
[3] Hong Kong University of Science and Technology
[4] Computer Science Department, Aberystwyth University, SY23 3FL, UK
dxh17@mails.tsinghua.edu.cn zhangxiangyu@megvii.com nmaac@cse.ust.hk
jungonghan77@gmail.com dinggg@tsinghua.edu.cn sunjian@megvii.com

图5、6 RepVGG相关文献

这部分内容已经在参考文献[3]中进行了详细的解释,且这部分不是必做内容，我在此不做详细的解释，上附网络图，详情请见文献内容。

5. DataSet：Plant photos

数据集中共有5544张(测试+训练)图像，12类各种不同类别的植物。

图7 数据集中图像展示

图像展示代码详见$data2.py$文件

三、实验过程

1. 构建数据集与性能调优

基本数据准备

为保证模型能有一个更好的训练效果本次数据处理部分首先合并整体数据集进行$resize$后再进行数据集均值和方差的计算。计算代码如下图所示：

```
def getStat(train_data):
    print('计算均值和方差：')
    print(len(train_data))
    train_loader = torch.utils.data.DataLoader(
        train_data, batch_size=1, shuffle=False, num_workers=4,
        pin_memory=True) #使用锁页内存，加快速度，数据集比较小
    mean = torch.zeros(3)
    std = torch.zeros(3)
    #不同的维数做相加
    for X, _ in train_loader:
```

```
        for d in range(3):
            X = np.array(X)
            mean[d] += X[:, d, :, :].mean()
            std[d] += X[:, d, :, :].std()
    #除个数
    mean.div_(len(train_data))
    std.div_(len(train_data))
    return list(mean.numpy()), list(std.numpy())
```

可得计算结果如下所示，后续$transform$中使用的也是这个值：

```
([0.3288099, 0.28887436, 0.20684063], [0.09116989, 0.094423905, 0.1035209])
```

但是比较可惜的一点是，当时训练$RepVGG$的时候这个参数设置的不是这个值，从而导致性能上不是那么的理想，最后的准确率没有很高。

2.性能调优

1）数据增广

VGG网络

为了提升性能我在VGG网络中使用了如下的一些简单的数据增广方式，但实际上效果不一定会好，原因在于实际上的植物图像分布的方差还是很小的，像下面的引入亮度的变化不一定会使其效果变得更好。

```
transform = transforms.Compose([
    transforms.Resize([size, size]),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1),
    transforms.Resize([size,size]),
    transforms.ToTensor(), #转化成张量，从 224 * 224 * 3 转化为 3 * 224 * 224
    transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
])
```

为了增加数据多样性，进行数据增广，我参考了跟李沐学AI[1],对数据进行了增广，在一定程度上提高了神经网络的性能。

结果比较如下：

共训练30个epoch

未做数据增广：

使用数据增广：

RepVGG网络

在$RepVGG$网络中我使用了$Cutout$和$Mixup$这两种增强方式：

```
  mixup_fn = Mixup(
    mixup_alpha=0.8, cutmix_alpha=1.0, cutmix_minmax=None,
    prob=0.1, switch_prob=0.5, mode='batch',
    label_smoothing=0.1, num_classes=12)
criterion_train = SoftTargetCrossEntropy()
```

参数详解：

- mixup_alpha (float): mixup alpha 值，如果 > 0，则 mixup 处于活动状态。
- cutmix_alpha (float)：cutmix alpha 值，如果 > 0，cutmix 处于活动状态。
- cutmix_minmax (List[float])：cutmix 最小/最大图像比率，cutmix 处于活动状态，如果不是 None，则使用这个 vs alpha。
- 如果设置了 cutmix_minmax 则cutmix_alpha 默认为1.0
- prob (float): 每批次或元素应用 mixup 或 cutmix 的概率。
- switch_prob (float): 当两者都处于活动状态时切换cutmix 和mixup 的概率 。
- mode (str): 如何应用 mixup/cutmix 参数（每个'batch'，'pair'（元素对），'elem'（元素）。
- correct_lam (bool): 当 cutmix bbox 被图像边框剪裁时应用。 lambda 校正
- label_smoothing (float)：将标签平滑应用于混合目标张量。
- num_classes (int): 目标的类数

  RepVGG网络没有进行结果的比较，而是直接用来测试了，因为当时初始化的参数弄错了，但是性能也还好，所以就没再训。

2）不同梯度下降算法的比较

在ResNet-18的网络下，对比使用SGD和Adam作为优化器，学习率0.0001，迭代30轮的结果：

使用SGD：

使用Adam:

可以看出在同样的轮数下，Adam优化器的性能明显高于SGD，因此在之后的训练中考虑使用Adam作为优化器。

3. 构建神经网络

1.VGG网络

VGG网络的核心部件为$VGG-blocks$,其中的构造已在上文中进行了阐述，在VGG11网络中只需要规定好输出和输入的通道数即可，核心代码如下所示：

```
conv_arch = ((1,64),(1,128),(2,256),(2,512),(2,512))

def vgg(conv_arch):
    conv_blks = []
    in_channels = 3
    for (num_convs,out_channels) in conv_arch:
        conv_blks.append(vgg_block(num_convs,in_channels,out_channels))
        in_channels = out_channels

    return nn.Sequential(*conv_blks,nn.Flatten(),
                         nn.Linear(out_channels * 7 *
7,4096),nn.BatchNorm1d(4096),nn.ReLU(),

nn.Dropout(0.5),nn.Linear(4096,4096),nn.BatchNorm1d(4096),nn.ReLU(),
                         nn.Dropout(0.5),nn.Linear(4096,12))
```

```
def vgg_block(num_convs,in_channels,out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(
            nn.Conv2d(in_channels,out_channels,kernel_size=3,padding=1))
        layers.append(nn.BatchNorm2d(out_channels))
        layers.append(nn.ReLU())
        in_channels = out_channels
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    #拆分成一个又一个元素
    return nn.Sequential(*layers)
```

2. ResNet网络

定义 ResNet-18 模型的主体结构：

```
class ResNet18(nn.Module):
    def __init__(self, num_classes, block_type='basic', use_gpu=True):
        super(ResNet18, self).__init__()
        self.use_gpu = use_gpu
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block_type, 64, 2)
        self.layer2 = self._make_layer(block_type, 128, 2, stride=2)
        self.layer3 = self._make_layer(block_type, 256, 2, stride=2)
        self.layer4 = self._make_layer(block_type, 512, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
```

```
        self.fc = nn.Linear(512, num_classes)
```

在这个结构中，我们首先定义了输入的第一层卷积层 conv1，其卷积核大小为 7×7，步长为 2，padding 为 3，以便使输入尺寸缩小。然后是批归一化层 bn1 和池化层 maxpool。接下来，我们通过多个重复的残差块构建 ResNet 的主体结构，其中每个重复的残差块由多个 `BasicBlock` 组成。每个残差块中的第一个 `BasicBlock` 可以使用 stride=2 进行降采样，以便在通道数发生变化时调整特征图的尺寸。最后，我们通过自适应平均池化层和一个全连接层来完成分类任务。如果 use_gpu 为 True，将模型移动到 GPU 上进行计算。

接下来，我们定义 `_make_layer` 函数，用于构建重复的残差块：

```
    def _make_layer(self, block_type, out_channels, num_blocks, stride=1):
        if block_type == 'basic':
            block = BasicBlock
        else:
            raise ValueError('Unsupported block type: %s' % block_type)
        downsample = None
        if stride != 1 or self.in_channels != out_channels * block.expansion:
            downsample = nn.Sequential(
                nn.Conv2d(self.in_channels, out_channels * block.expansion,
kernel_size=1,                            stride=stride, bias=False),
                nn.BatchNorm2d(out_channels * block.expansion)
            )
        layers = [block(self.in_channels, out_channels, stride, downsample,
self.use_gpu)]
        self.in_channels = out_channels * block.expansion
        for i in range(1, num_blocks):
            layers.append(block(self.in_channels, out_channels,
use_gpu=self.use_gpu))
        return nn.Sequential(*layers)
```

该函数接受四个参数： `BasicBlock` 类； `out_channels` 表示当前重复的残差块的输出通道数；blocks 表示当前重复的残差块中 `BasicBlock` 的数量；stride 表示第一个 `BasicBlock` 是否需要降采样。

在该函数中，我们首先根据 stride和当前输入通道数判断是否需要进行降采样，并构建 `downsample`。然后我们循环调用 block 来创建 `BasicBlock`，并将这些 `BasicBlock` 放入 `layers` 列表中。

最后，我们定义了 ResNet-18 模型所需的 `BasicBlock` 类，包含了两个卷积层和一个残差连接，其定义如下：

```
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channels, out_channels, stride=1, downsample=None,
                 use_gpu=True):
        super(BasicBlock, self).__init__()
        self.use_gpu = use_gpu
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride,
                               padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
```

```
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample is not None:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

3. ResNet + SE 网络

`SEBlock`是一个注意力机制，用于调整每个通道的重要性。它的实现如下：

```
class SEBlock(nn.Module):
    def __init__(self, in_channels, reduction_ratio):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(in_channels, in_channels // reduction_ratio)
        self.fc2 = nn.Linear(in_channels // reduction_ratio, in_channels)
        self.relu = nn.ReLU(inplace=True)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc1(y)
        y = self.relu(y)
        y = self.fc2(y)
        y = self.sigmoid(y).view(b, c, 1, 1)
        return x * y
```

`SEBlock` 接受一个 `channel` 参数，表示输入张量的通道数； `reduction` 参数表示缩小比例。`SEBlock` 包括两个部分：全局平均池化和全连接层。

在 `__init__` 方法中，我们定义了全局平均池化和一个两层全连接层，用于计算每个通道的重要性。全局平均池化将输入张量的所有通道的每个位置的值平均为一个标量值，这样我们就得到了每个通道的平均值。然后，我们将这个平均值输入到全连接层中，对其进行压缩，并将其输出到另一个全连接层中，对其进行放大。最后，我们使用 sigmoid 激活函数得到一个权重向量 `y`，其中的每个元素表示该通道的重要性。

在`forward`方法中，我们首先计算全局平均池化，并将其形状变为`(batch_size, channels)`。然后，我们将它输入到全连接层中，得到一个形状为`(batch_size, channels // reduction)`的输出。接下来，我们将输出输入到另一个全连接层中，得到一个形状与输入张量相同的输出。最后，我们将输入张量与计算出的权重向量相乘，得到加强后的特征图。

修改`BasicBlock`类，在ResNet-18中加入`SEBlock`：

```
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channels, out_channels, stride=1, downsample=None,
                 use_gpu=True):
        super(BasicBlock, self).__init__()
        self.use_gpu = use_gpu
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride,
                               padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample
        # 添加 SE block
        self.stride = stride
        self.se = SEBlock(out_channels, reduction_ratio=16)
```

4.RepVGG网络

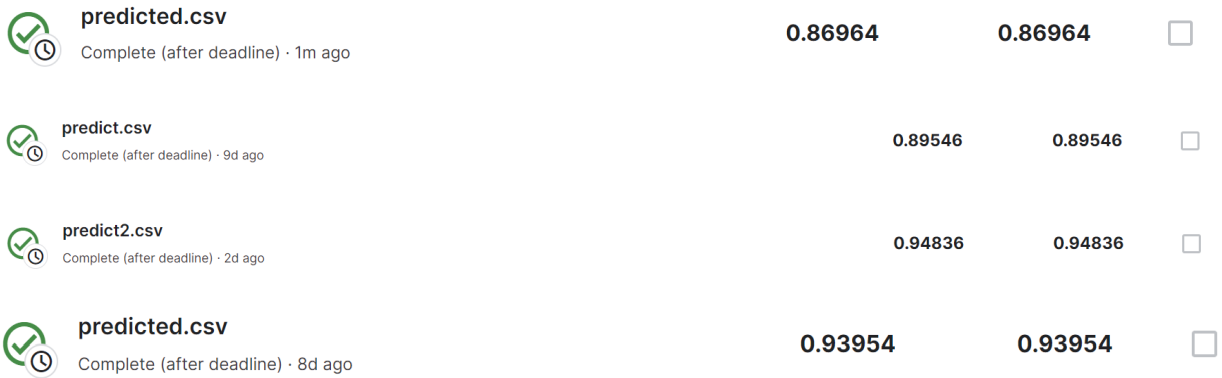
RepVGGBlock的构成如下所示：

```
class RepVGGBlock(nn.Module):

    def __init__(self, in_channels, out_channels, kernel_size,
                 stride=1, padding=0, dilation=1, groups=1, padding_mode='zeros',
deploy=False, use_se=False):
        super(RepVGGBlock, self).__init__()
        self.deploy = deploy
        self.groups = groups
        self.in_channels = in_channels

        assert kernel_size == 3
        assert padding == 1

        padding_11 = padding - kernel_size // 2

        self.nonlinearity = nn.ReLU()

        if use_se:
            self.se = SEBlock(out_channels, internal_neurons=out_channels // 16)
        else:
            self.se = nn.Identity()
```

```
        if deploy:
            self.rbr_reparam = nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=kernel_size, stride=stride,
                                         padding=padding, dilation=dilation,
groups=groups, bias=True, padding_mode=padding_mode)

        else:
            self.rbr_identity = nn.BatchNorm2d(num_features=in_channels) if
out_channels == in_channels and stride == 1 else None
            self.rbr_dense = conv_bn(in_channels=in_channels,
out_channels=out_channels, kernel_size=kernel_size, stride=stride, padding=padding,
groups=groups)
            self.rbr_1x1 = conv_bn(in_channels=in_channels,
out_channels=out_channels, kernel_size=1, stride=stride, padding=padding_11,
groups=groups)
            print('RepVGG Block, identity = ', self.rbr_identity)
```

3. 训练结果分析

神经网络架构	训练轮数	测试集正确率
VGG_{11}	30	0.86964
$ResNet$	50	0.89546
$ResNet+SE$	50	0.94836
$RepVGG$	80	0.93954



$ResNet$过程分析：

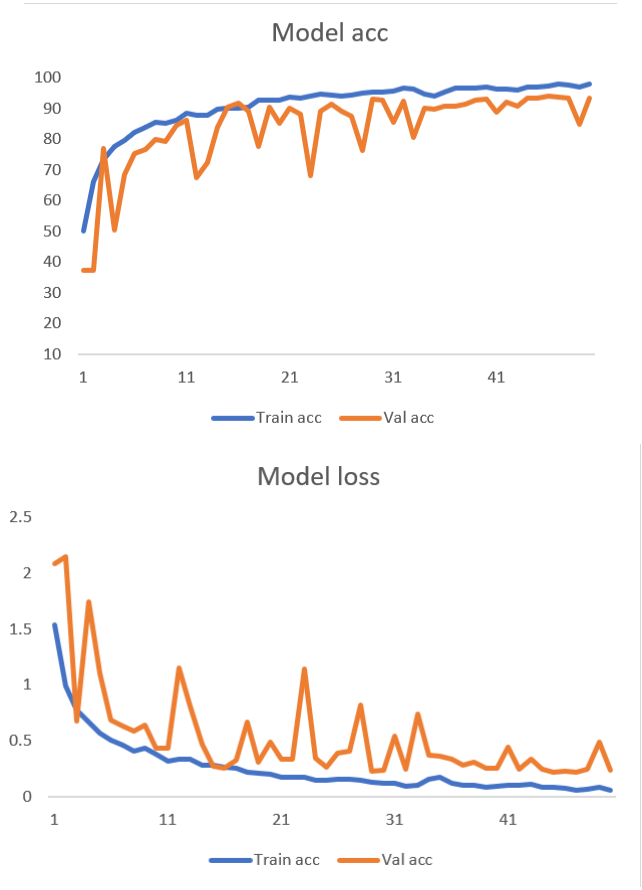
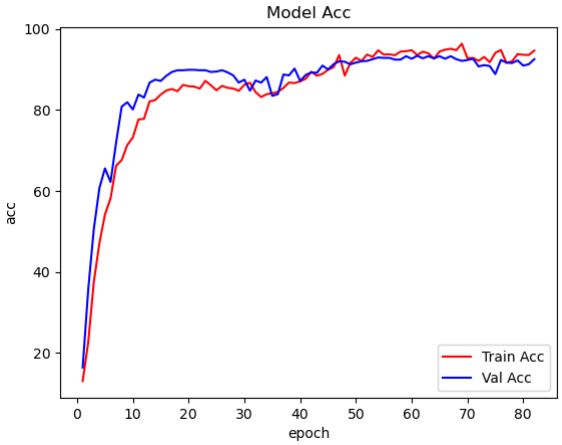


图8 ResNet训练过程图像

$RepVGG$过程分析：



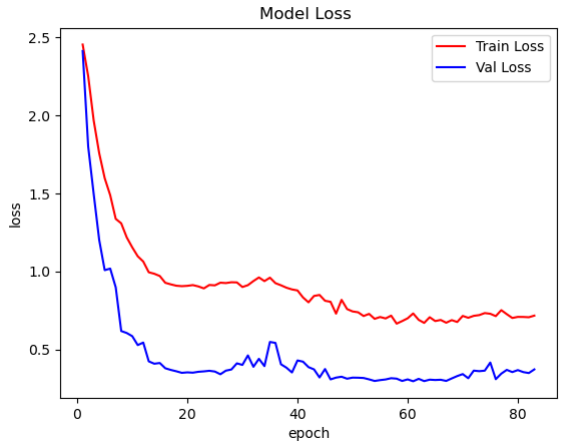


图9 RepVGG训练过程图像

如上图3我们可以看出训练过程前期过程较为欠拟合，后期过程出现了一些可能过拟合的情况，所以我终止了训练，或许将前期$transform$的均值和方差改为正确值会使得结果的准确率更高。

四、分工情况

- VGG: 黄一洋
- ResNet：蔡泽栋
- ResNet + SE：蔡泽栋
- RepVGG(性能调优自选)：黄一洋
- 性能调优公共部分：黄一洋、蔡泽栋
- 报告总结：黄一洋、蔡泽栋

五、参考文献

1. 跟李沐学AI ↩ ↩ ↩

2. stanford cs231n ↩

3. [*RepVGG*：极简架构，SOTA性能，让VGG式模型再次伟大（CVPR-2021）] ↩