实验题目	Kaggle 分类比赛			实验日期	
班级	1903103	学号	1190201225	姓名	韩庸平
班级	1903602	学号	1190201215	姓名	冯开来

模式识别与深度学习实验 3 报告

实验结果的正确性(60%)		表达能力(10%)	
实验过程的规范性(10%)		实验报告(20%)	
加分(5%)		总成绩(100%)	

实验报告

一、实验目的（介绍实验目的）

1、基于 PyTorch 实现 VGG/ResNet/SENet 等结构：
   1）自己实现 VGG(11)：要求 Conv 部分参照论文，可动态调整结构；
   2）自己实现 ResNet(18)：要求基于残差块，参照论文，可动态调整；
   3）在 ResNet 基础上，添加 SE block；
   4）对比其性能表现
2、要求基于 CUDA 实现
   1）可设定是否使用 GPU，通过 argparse 包实现，默认参数设定为 GPU
3、性能调优：
   1）进行优化器（SGD 与 Adam）对比
   2）进行 data augmentation(翻转、旋转、移位等操作对比)

二、实验环境（介绍实验使用的硬件设备、软件系统、开发工具等）

windows 10
Python 3.8　torch 1.11.0　cuda 1.10

三、实验过程（介绍实验过程、设计方案、实现方法、实验结果等）

一、参考论文

1、Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. ICLR, 2015. (VGG)
2、Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. CVPR, 2016. (ResNet)
3、Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu. Squeeze-and-Excitation Networks. CVPR, 2018. (SENet)

二、 简单描述 VGG\ResNet\SE-Net 等结构

1、VGG

VGG 组成：
VGG 由 5 层卷积层、3 层全连接层、softmax 输出层构成，层与层之间使用 max-pooling 分开，所有隐

实验题目	Kaggle 分类比赛			实验日期	
班级	1903103	学号	1190201225	姓名	韩庸平
班级	1903602	学号	1190201215	姓名	冯开来

层的激活单元都采用 ReLU 函数。

使用多个小卷积核代替卷积核较大的卷积层，在减少参数的同时，进行了更多的非线性映射，增加了网络的拟合、表达能力。VGG 第一层通道数位 64，后面通道数增加到 512，可以提取更多的信息。

```
class VGG(torch.nn.Module):  # 继承 torch 的 Module
    def __init__(self):
        super(VGG, self).__init__()  #
        self.module = torch.nn.Sequential(
            # 1
            torch.nn.Conv2d(3, 64, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # 2
            torch.nn.Conv2d(64, 128, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # 3
            torch.nn.Conv2d(128, 256, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.Conv2d(256, 256, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # 4
            torch.nn.Conv2d(256, 512, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.Conv2d(512, 512, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # 5
            torch.nn.Conv2d(512, 512, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.Conv2d(512, 512, kernel_size=3, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classify = torch.nn.Sequential(
            torch.nn.Flatten(),
            torch.nn.Linear(25088, 4096),
            torch.nn.ReLU(inplace=True),
            torch.nn.Dropout(0.5),#有效降低过拟合
            torch.nn.Linear(4096, 4096),
            torch.nn.ReLU(inplace=True),
```

实验题目	Kaggle 分类比赛			实验日期	
班级	1903103	学号	1190201225	姓名	韩庸平
班级	1903602	学号	1190201215	姓名	冯开来

```
            torch.nn.Dropout(0.5),
            torch.nn.Linear(4096, 12)
        )

    def forward(self, x):
        x = self.module(x)
        x = self.classify(x)
        return x
```

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Vgg11 网络结构

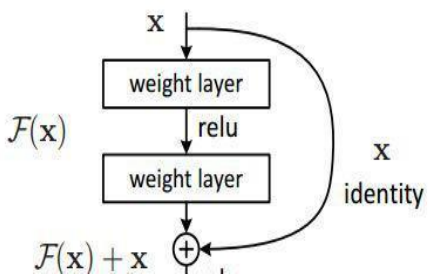
注：有所不同的是输入图片的大小改为 25088，最后一个全连接层的 FC 是 12 即 12species

11 层共有 8 个卷积层以及 3 个全链接层

2. ResNet 残差网络

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。再 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

1）残差块



实验题目	Kaggle 分类比赛			实验日期	
班级	1903103	学号	1190201225	姓名	韩庸平
班级	1903602	学号	1190201215	姓名	冯开来

2）ResNet(18) 17 个卷积层 1 个全链接层

其中网络参数如下图所示

layer name	output size	18-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$
	1×1	
FLOPs		1.8×10^9

3）具体实现

```
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 7, 2, 3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(3, 2, 1)
        )
        self.conv2 = nn.Sequential(
            Block.CommonBlock(64, 64),
            Block.CommonBlock(64, 64)
        )
        self.conv3 = nn.Sequential(
            Block.SpecialBlock(64, 128, 2),         # stride != 1 and in_channel != out_channel，需要下采样
            Block.CommonBlock(128, 128)
        )
        self.conv4 = nn.Sequential(
            Block.SpecialBlock(128, 256, 2),
            Block.CommonBlock(256, 256)
        )
        self.conv5 = nn.Sequential(
            Block.SpecialBlock(256, 512, 2),
            Block.CommonBlock(512, 512)
        )
        self.dense = nn.Sequential(                 # 最后用于分类的全连接层，根据需要灵活变化
            nn.AdaptiveAvgPool2d(output_size=(1, 1)),       # 自适应平均池化
```

实验题目	Kaggle 分类比赛			实验日期	
班级	1903103	学号	1190201225	姓名	韩庸平
班级	1903602	学号	1190201215	姓名	冯开来

```
            nn.Flatten(),
            nn.Linear(512, 12)
        )

    def forward(self, image):
        img = self.conv1(image)
        img = self.conv2(img)          # 四个卷积单元
        img = self.conv3(img)
        img = self.conv4(img)
        img = self.conv5(img)
        img = self.dense(img)            # 全连接
        return img
```

注释：CommonBlock 表示简单完成两次卷积操作，SpecialBlock 完成两次卷积操作以及一次下采样。

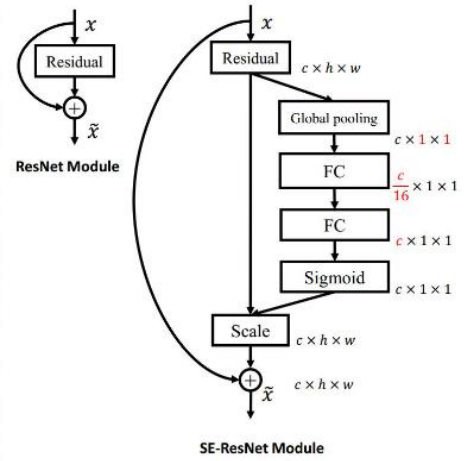
所以卷积层：1*Conv2d+5*CommonBlock + 3*SpecialBlock = 1 + 5*2 +3*2 = 17

3、SEResNet

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：



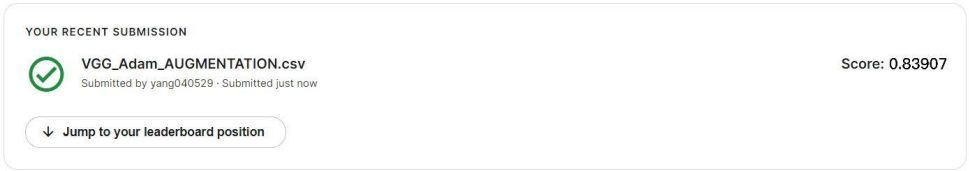
SEblock 的实现：

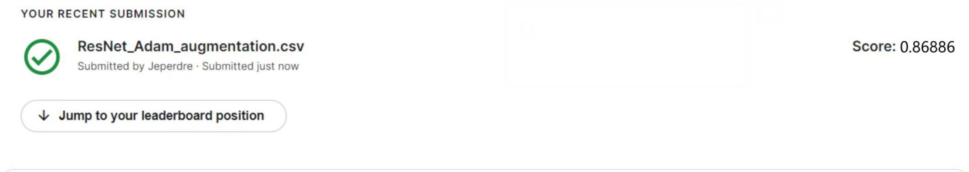
```
class SEBlock(nn.Module):
    def __init__(self, in_channel, reduction=16):
        super(SEBlock, self).__init__()
        self.global_pooling = nn.AdaptiveAvgPool2d(output_size=(1, 1))       # 全局池化
        self.fc = nn.Sequential(
            nn.Linear(in_channel, in_channel // reduction),
```

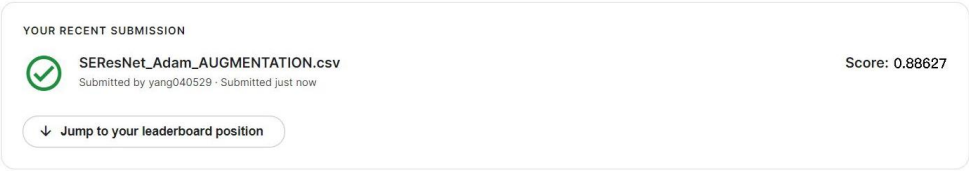
```
            nn.ReLU(),
            nn.Linear(in_channel // reduction, in_channel),
            nn.Sigmoid()
        )

    def forward(self, image):
        b, c, _, _ = image.size()
        y = self.global_pooling(image).view(b, c)   # 得到 B*C*1*1,然后转成 B*C，才能送入到 FC 层中
        y = self.fc(y).view(b, c, 1, 1)     # 得到 B*C 的向量，C 个值就表示 C 个通道的权重。把 B*C 变为 B*C*1*1 是为了与四维的 x 运算。
        return image * y.expand_as(image)   # 先把 B*C*1*1 变成 B*C*H*W 大小，其中每个通道上的 H*W 个值都相等。*表示对应位置相乘。
```

列表对比不同结构的测试时间（millisecond/image）和 Score 分数（Multi Class Log Loss，测试方式见下页）；

模型	Score	Millisecond/image
VGG(11)	0.83907	11.7
ResNet(18)	0.86886	9.04
SENet	0.88627	10.3







三、对比分析不同的优化器和数据增广方式对性能的影响

1、在 SENet 上对于 Adam,SGD 两个优化器的效果如下表所示：

优化器	Score
Adam	0.88627
SGD	0.37801

由上表可以看出，Adam 优化器的效果比较好。而 SGD 的效果较差，可能是因为随机梯度下降是选择 mini-batch 而不是全部样本。而 Adam 是 SGDM 和 RMSProp 优化后的产物，对于随即小样本以及自适应学

习率等问题处理的更好。

2、不同增广方式对比

增强方式	Score
不做变换	0.88627
随即平移 0~0.7 倍图像	0.84553
随即旋转（0，90）度	0.93178
随即水平垂直方向旋转	0.91561

可以看出后面两种数据增强方式对于 SENet 的性能又较为显著的提升。因为通过数据增强相当于增强了训练集的数据类型，增加了更多种的情况，增加了模型的泛化能力。

四、参数、是否使用 GPU、数据增广、以及选用的网络结构等

```
# 运算设备,默认选择 GPU
if opt.device == 'GPU':
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
else:
    device = torch.device("cpu")
print("设备: " + torch.cuda.get_device_name(device))
```

五、自选模型与结果

选用的网络结构是 ResNet50。Resnet50 网络中包含了 49 个卷积层、一个全连接层，可以分成七个部分。第一部分不包含残差块，主要对输入进行卷积、正则化、激活函数、最大池化的计算。第二、三、四、五部分结构都包含了残差块， 残差块都有三层卷积，所以网 络 总 共 有 1+3×（3+4+6+3）=49 个卷积层，加上最后的全连接层总共是 50 层。

ResNet50 相比 ResNet18 网络更深，参数更多，模型更加复杂，在加入 SE block 会有更好的性能。我们对于 ResNet50，最终得到的结果如下。

六、成员分工

冯开来：resnet senet 以及测试分析

韩庸平：vgg 以及实验报告整理