实验 ResNet50

实验目的

1. 实现ResNet残差块Bottleneck。
2. 基于mindspore构造ResNet50网络。
3. 使用ResNet50完成CIFAR10图片分类。

实验数据

CIFAR-10是一个常用的计算机视觉数据集，由加拿大的研究机构CIFAR（Canadian Institute for Advanced Research）维护。它被广泛用于机器学习和计算机视觉领域的研究和实验中，特别是在图像识别和分类任务中。

实验步骤

1. 数据集加载与增强

CIFAR-10数据集共有60000张32*32的彩色图像，分为10个类别，每类有6000张图，数据集一共有50000张训练图片和10000张评估图片。首先，如下示例使用 `download` 接口下载并解压，目前仅支持解析二进制版本的CIFAR-10文件（CIFAR-10 binary version）。

```
!pip install download nltk
from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/datasets/cifar-10-binary.tar.gz"

download(url, "./datasets-cifar10-bin", kind="tar.gz", replace=True)
```

首次运行下载代码后可以将该部分注释掉。下载后的数据集目录结构如下：

```
datasets-cifar10-bin/cifar-10-batches-bin
├── batches.meta.text
├── data_batch_1.bin
├── data_batch_2.bin
├── data_batch_3.bin
```

```
├── data_batch_4.bin
├── data_batch_5.bin
├── readme.html
└── test_batch.bin
```

然后，使用 `mindspore.dataset.Cifar10Dataset` 接口来加载数据集，并进行相关图像增强操作。该部分代码已给出，训练数据储存在 `train_dataset` 中，测试数据储存在 `test_dataset` 中。

数据可视化

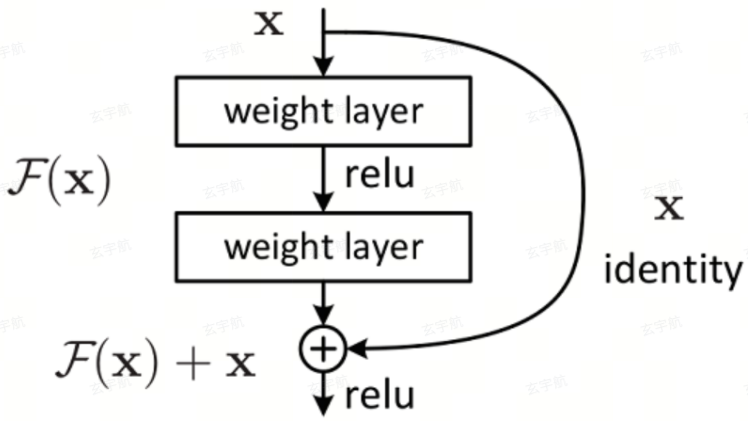
定义 `visualize_dataset()` 函数对数据进行可视化。

```python
def visualize_dataset(dataset, num_samples=5):
    data_iter = dataset.create_dict_iterator(output_numpy=True)
    for i, data in enumerate(data_iter):
        if i * data['image'].shape[0] >= num_samples:
            break
        images = data['image']
        labels = data['label']
        batch_size = images.shape[0]
        for j in range(batch_size):
            if i * batch_size + j >= num_samples:
                break
            img = images[j]
            label = labels[j]
            img = img.transpose((1, 2, 0))  # CHW to HWC
            img = (img * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]) * 255
            img = np.clip(img, 0, 255).astype(np.uint8)
            plt.subplot(1, num_samples, i * batch_size + j + 1)
            plt.title(f'Label: {label}')
            plt.imshow(img)
            plt.axis('off')
    plt.show()

visualize_dataset(train_dataset)
```

2. 构造残差快ResidualBlock

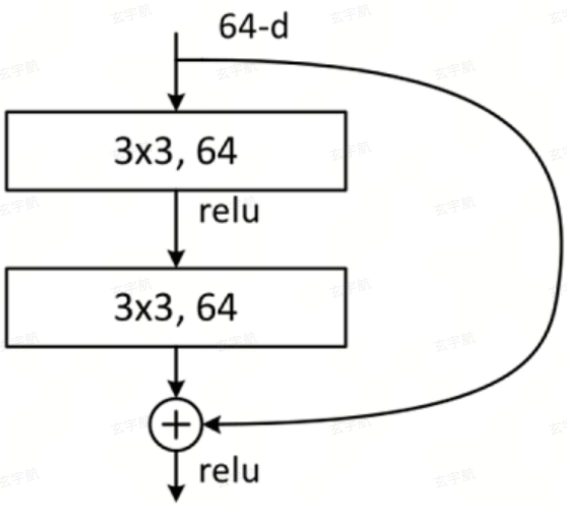
残差网络结构(Residual Network)是ResNet网络的主要亮点，ResNet使用残差网络结构后可有效地减轻退化问题，实现更深的网络结构设计，提高网络的训练精度。本节首先讲述如何构建残差网络结构，然后下节通过堆叠残差网络来构建ResNet50网络。

残差网络结构图如下图所示，残差网络由两个分支构成：一个主分支，一个shortcuts（图中弧线表示）。主分支通过堆叠一系列的卷积操作得到，shotcuts从输入直接到输出，主分支输出的特征矩阵F(x)加上shortcuts输出的特征矩阵x得到F(x)+x，通过Relu激活函数后即为残差网络最后的输出。



残差网络结构主要由两种，一种是Building Block，适用于较浅的ResNet网络，如ResNet18和ResNet34；另一种是Bottleneck，适用于层数较深的ResNet网络，如ResNet50、ResNet101和ResNet152。

1. Building Block



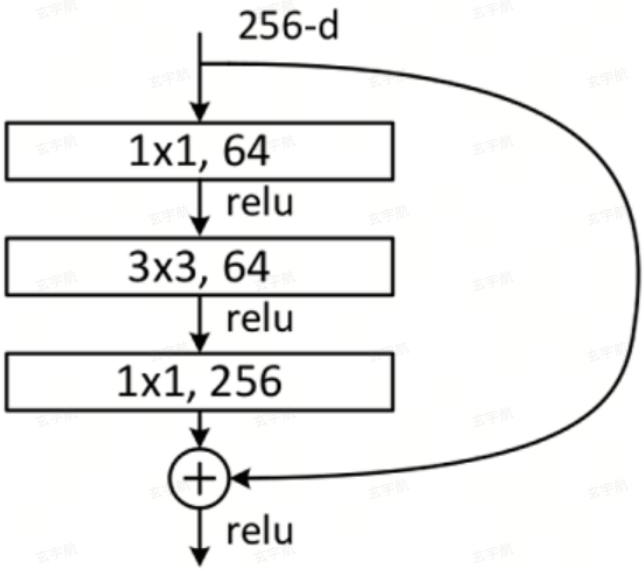
该部分代码已经给出。Building Block结构图如上图所示，主分支有两层卷积网络结构：

- 主分支第一层网络以输入channel为64为例，首先通过一个3×3的卷积层，然后通过Batch Normalization层，最后通过Relu激活函数层，输出channel为64；
- 主分支第二层网络的输入channel为64，首先通过一个3×3的卷积层，然后通过Batch Normalization层，输出channel为64。

最后将主分支输出的特征矩阵与shortcuts输出的特征矩阵相加，通过Relu激活函数即为Building Block最后的输出。

```
class ResidualBlock_buildingblock(nn.Cell):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock_buildingblock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, pad_mode='same', has_bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, pad_mode='same', has_bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def construct(self, x):
        identity = 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:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out
```

2. Bottleneck



Bottleneck结构图如上图所示，在输入相同的情况下Bottleneck结构相对Building Block结构的参数数量更少，更适合层数较深的网络，ResNet50使用的残差结构就是Bottleneck。该结构的主分支有三层卷积结构，分别为1×1的卷积层、3×3卷积层和1×1的卷积层，其中1×1的卷积层分别起降维和升维的作用。

- 主分支第一层网络以输入channel为256为例，首先通过数量为64，大小为1×1的卷积核进行降维，然后通过Batch Normalization层，最后通过Relu激活函数层，其输出channel为64；
- 主分支第二层网络通过数量为64，大小为3×3的卷积核提取特征，然后通过Batch Normalization层，最后通过Relu激活函数层，其输出channel为64；
- 主分支第三层通过数量为256，大小1×1的卷积核进行升维，然后通过Batch Normalization层，其输出channel为256。

最后将主分支输出的特征矩阵与shortcuts输出的特征矩阵相加，通过Relu激活函数即为Bottleneck最后的输出。

请参照building block代码补全ResidualBlock类实现bottleneck：

```
class ResidualBlock(nn.Cell):
    def __init__(self, in_channels, out_channels, stride=1, downsample=False):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=0, pad_mode='pad')
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, pad_mode='pad')
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels * 4, kernel_size=1, stride=1, padding=0, pad_mode='pad')
        self.bn3 = nn.BatchNorm2d(out_channels * 4)
        self.relu = nn.ReLU()
        self.downsample = downsample
```

```
        if self.downsample:
            self.shortcut = nn.SequentialCell([
                nn.Conv2d(in_channels, out_channels * 4, kernel_size=1,
stride=stride, padding=0, pad_mode='pad'),
                nn.BatchNorm2d(out_channels * 4)
            ])

    def construct(self, x):
        identity = x
        '''

        code here

        '''
        if self.downsample:
            identity = self.shortcut(x)
        '''

        code here

        '''
        return out
```

3. 构造ResNet50网络模型

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet50网络共有5个卷积结构，一个平均池化层，一个全连接层，以CIFAR-10数据集为例：

conv1：输入图片大小为32×32，输入channel为3。首先经过一个卷积核数量为64，卷积核大小为7×7，stride为2的卷积层；然后通过一个Batch Normalization层；最后通过Reul激活函数。该层输出feature map大小为16×16，输出channel为64。

conv2_x：输入feature map大小为16×16，输入channel为64。首先经过一个卷积核大小为3×3，stride为2的最大下采样池化操作；然后堆叠3个$[1×1，64；3×3，64；1×1，256]$结构的Bottleneck。该层输出feature map大小为8×8，输出channel为256。

conv3_x：输入feature map大小为8×8，输入channel为256。该层堆叠4个[1×1，128；3×3，128；1×1，512]结构的Bottleneck。该层输出feature map大小为4×4，输出channel为512。

conv4_x：输入feature map大小为4×4，输入channel为512。该层堆叠6个[1×1，256；3×3，256；1×1，1024]结构的Bottleneck。该层输出feature map大小为2×2，输出channel为1024。

conv5_x：输入feature map大小为2×2，输入channel为1024。该层堆叠3个[1×1，512；3×3，512；1×1，2048]结构的Bottleneck。该层输出feature map大小为1×1，输出channel为2048。

average pool & fc：输入channel为2048，输出channel为分类的类别数。

```
# 定义ResNet
class ResNet(nn.Cell):
    def __init__(self, block, layers, num_classes=10):
        super(ResNet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2)
        self.layer1 = self._make_layer(block, 64, layers[0], stride=1)
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.avgpool = nn.AvgPool2d(kernel_size=4)
        self.fc = nn.Dense(512 * 4, num_classes)

    def _make_layer(self, block, out_channels, blocks, stride):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride,
downsample=True))
        self.in_channels = out_channels * 4
        for _ in range(1, blocks):
            layers.append(block(self.in_channels, out_channels))
        return nn.SequentialCell(layers)

    def construct(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
```

```
        x = self.relu(x)
        ''' code here '''

        x = self.layer1(x)
        '''

        code here

        '''
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x

def ResNet50(num_classes=10):
    return ResNet(ResidualBlock, [3, 4, 6, 3], num_classes)
```

4. 模型训练与评估

```
# 定义模型、损失函数和优化器
resnet50 = ResNet50()
loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
optimizer = nn.Momentum(resnet50.trainable_params(), learning_rate=0.01,
momentum=0.9)
```

```
# 训练模型
model = Model(resnet50, loss_fn, optimizer, metrics={"accuracy": Accuracy()})
model.train(5, train_dataset, callbacks=[LossMonitor()],
dataset_sink_mode=False)

# 测试模型
acc = model.eval(test_dataset, dataset_sink_mode=False)
print(f"Accuracy: {acc['accuracy']}")
```

这部分你可以改变参数观察训练效果，ResNet50的训练时间很长，单个epoch可能需要数个小时的时间训练。所以本次作业不需要有模型准确率，完成上面的网络构造即可。