深度学习基础

实验三

版本：2.2



华为技术有限公司

|  |
| --- |
| **版权所有 © 华为技术有限公司 2021。 保留一切权利。**  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  **商标声明**  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  **注意**  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验介绍 4](#_Toc123299258)

[1.1 实验目的 4](#_Toc123299259)

[1.2 实验清单 4](#_Toc123299260)

[1.3 实验开发环境 5](#_Toc123299261)

[1.4 开发平台介绍 5](#_Toc123299262)

[1.5 背景知识 5](#_Toc123299263)

[2 正则化实验 9](#_Toc123299264)

[2.1 实验介绍 9](#_Toc123299265)

[2.2 实验环境要求 9](#_Toc123299266)

[2.3 实验总体设计 9](#_Toc123299267)

[2.4 实验过程 9](#_Toc123299268)

[2.4.1 下载调用包 10](#_Toc123299269)

[2.4.2 超参数设定 10](#_Toc123299270)

[2.4.3 建立数据集 11](#_Toc123299271)

[2.4.4 建立网络 11](#_Toc123299272)

[2.4.5 启动训练 12](#_Toc123299273)

[2.5 实验总结 16](#_Toc123299274)

[3 FashionMnist分类任务正则化对比实验 17](#_Toc123299275)

[3.1 实验介绍 17](#_Toc123299276)

[3.2 实验环境要求 17](#_Toc123299277)

[3.3 实验总体设计 17](#_Toc123299278)

[3.4 实验过程 17](#_Toc123299279)

[3.4.1 数据准备 18](#_Toc123299280)

[3.4.2 导入实验环境 18](#_Toc123299281)

[3.4.3 数据读取和预处理 19](#_Toc123299282)

[3.4.4 训练模型 22](#_Toc123299283)

[3.4.5 观察总结 28](#_Toc123299284)

[3.5 实验总结 29](#_Toc123299285)

[4 花卉图像分类实验 30](#_Toc123299286)

[4.1 实验介绍 30](#_Toc123299287)

[4.2 实验环境要求 30](#_Toc123299288)

[4.3 实验总体设计 30](#_Toc123299289)

[4.3.1 功能结构 30](#_Toc123299290)

[4.3.2 体系结构 30](#_Toc123299291)

[4.4 实验过程 31](#_Toc123299292)

[4.4.1 导入实验环境 31](#_Toc123299293)

[4.4.2 数据集获取与预处理 33](#_Toc123299294)

[4.4.3 构建CNN图像识别模型 35](#_Toc123299295)

[4.4.4 图像分类模型验证 37](#_Toc123299296)

[4.5 实验总结 38](#_Toc123299297)

[5 LeNet的手写数字识别实验 39](#_Toc123299298)

[5.1 实验介绍 39](#_Toc123299299)

[5.2 实验环境要求 39](#_Toc123299300)

[5.3 实验准备 39](#_Toc123299301)

[5.4 实验总体设计 40](#_Toc123299302)

[5.5 实验过程 40](#_Toc123299303)

[5.5.1 导入实验环境 40](#_Toc123299304)

[5.5.2 数据处理 41](#_Toc123299305)

[5.5.3 定义模型 42](#_Toc123299306)

[5.5.4 训练 43](#_Toc123299307)

[5.6 实验总结 44](#_Toc123299308)

[6 图像识别全流程代码实践 45](#_Toc123299309)

[6.1 实验介绍 45](#_Toc123299310)

[6.2 实验环境要求 45](#_Toc123299311)

[6.3 实验总体设计 45](#_Toc123299312)

[6.4 实验过程 45](#_Toc123299313)

[6.4.1 数据集获取 46](#_Toc123299314)

[6.4.2 导入实验环境 46](#_Toc123299315)

[6.4.3 读取数据集 48](#_Toc123299316)

[6.4.4 模型构建训练 49](#_Toc123299317)

[6.4.5 模型预测 60](#_Toc123299318)

[6.4.6 模型保存 61](#_Toc123299319)

[6.5 实验总结 63](#_Toc123299320)

[7 前沿网络案例- DeepLabv3 64](#_Toc123299321)

[7.1 实验介绍 64](#_Toc123299322)

[7.2 实验环境要求 64](#_Toc123299323)

[7.3 背景知识 64](#_Toc123299324)

[7.3.1 网络介绍 64](#_Toc123299325)

[7.4 实验步骤 66](#_Toc123299326)

[7.5 实验总结 89](#_Toc123299327)

[8 前沿网络案例-YOLOV3 90](#_Toc123299328)

[8.1 实验介绍 90](#_Toc123299329)

[8.2 实验环境要求 90](#_Toc123299330)

[8.3 实验准备 90](#_Toc123299331)

[8.4 背景知识 91](#_Toc123299332)

[8.4.1 网络介绍 91](#_Toc123299333)

[8.5 实验步骤 95](#_Toc123299334)

[8.6 实验总结 131](#_Toc123299335)

[9 附录：ModelArts开发环境搭建 132](#_Toc123299336)

# 实验介绍

卷积网络，也叫做卷积神经网络，是专门用来处理具有类似网格结构的数据的神经网络。例如时间序列数据（可以认为是在时间轴上有规律的采样形成的一维网格）和图像数据（可以看作二维的像素网格）。卷积在诸多应用领域都表现优异。本章主要围绕深度学习的卷积网络而开设的实验。本章实验难度分为初级，中级和高级。

初级实验：①花卉图像分类卷积网络训练验证实验；

中级实验：②正则化实验；③Fashion Mnist卷积网络正则化前后对比；

高级实验：④基于Lenet的手写数字识别；⑤图像识别全流程代码实践；⑥前沿网络案例Yolov3/ Deeplabv3。

## 实验目的

本章实验的主要目的是掌握卷及网络相关基础知识点。掌握不同神经网络架构的设计原理，熟悉使用MindSpore框架实验的一般流程，以及最后将模型部署上线。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验** | **简述** | **难度** | **软件环境** | **开发环境** |
| 正则化实验 | 定义各正则化方法并进行应用比较。 | 中级 | MindSpore-1.7-python3.7 | ModelArts |
| Fashion Mnist正则化前后对比 | FashionMnist卷积网络分类识别实验外加正则化效果对比。 | 中级 | MindSpore-1.7-python3.7 | ModelArts |
| 花卉图像分类卷积网络训练验证实验 | 进行花卉图像分类卷积网络分类识别实验。 | 初级 | Mindspore-1.7-python3.7 | ModelArts |
| Lenet手写数字识别实验 | Mnist手写图像Lenet卷积网络分类识别实验。 | 高级 | MindSpore-1.7-python3.7 | ModelArts |
| 图像识别全流程代码实践 | 训练并部署上线花卉图像分类卷积网络分类识别实验。 | 高级 | MindSpore-1.7-python3.7 | ModelArts |
| 前沿网络案例 | 熟悉前沿网络Yolov3，Deeplabv3并应用实例。 | 高级 | MindSpore-1.7-python3.7 | ModelArts |

## 实验开发环境

MindSpore

若选择在华为云ModelArts上快速搭建开发环境，可参考文末附录：ModelArts开发环境搭建。

## 开发平台介绍

MindSpore 最佳匹配昇腾芯片的开源AI计算框架，支持Asend、GPU、CPU平台。MindSpore官网：https://www.mindspore.cn

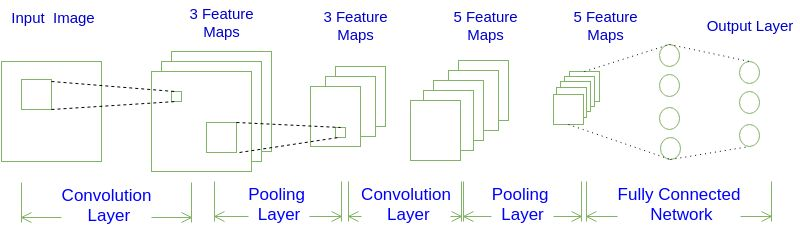
ModelArts是面向开发者的一站式AI开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期AI工作流。

具体内容请参考平台介绍ppt。

## 背景知识

卷积神经网络结构

卷积神经网络是深度学习与神经网络算法中主流算法之一，主要用于图像识别。其结构图如下：

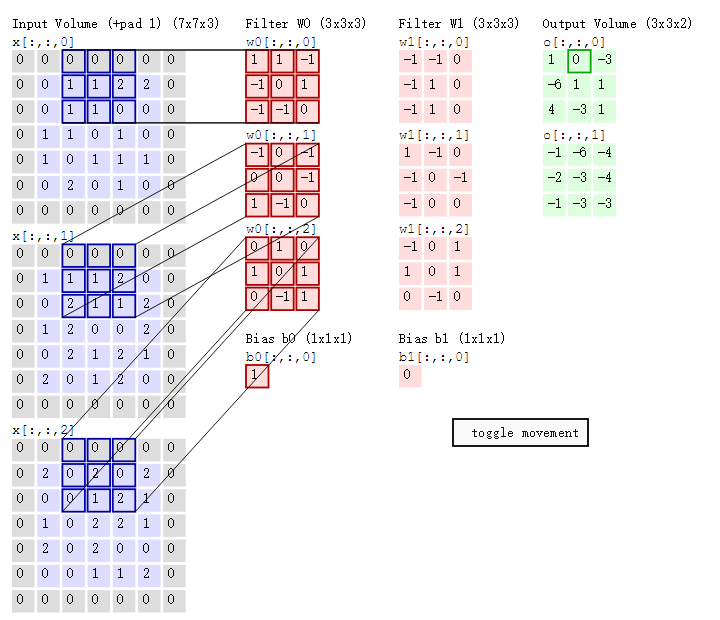


卷积神经网络结构

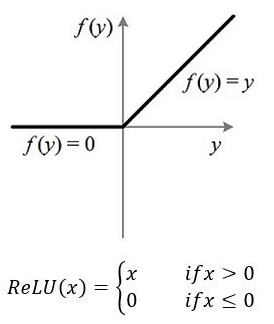
卷积层

在卷积层计算过程中，输入是一定区域大小(width\*height)的数据，和滤波器filter（带着一组固定权重的神经元）做内积后等到新的二维数据。

具体来说，滤波器filter（带着一组固定权重的神经元）通过滑动窗口的方式，对输入图像进行扫描，扫描过程中，对输入图像中的像素值进行点乘，不同的滤波器filter会得到不同的输出数据，比如颜色深浅、轮廓。相当于如果想提取图像的不同特征，则用不同的滤波器filter，提取想要的关于图像的特定信息：颜色深浅或轮廓。



激活函数：



ReLU函数其实是分段线性函数，把所有的负值都变为0，而正值不变，这种操作被成为单侧抑制。可别小看这个简单的操作，正因为有了这单侧抑制，才使得神经网络中的神经元也具有了稀疏激活性。

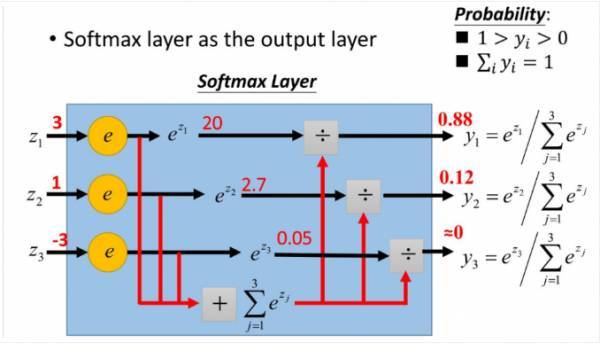
此外，相比于其它激活函数来说，ReLU有以下优势：对于线性函数而言，ReLU的表达能力更强，尤其体现在深度网络中；而对于sigmoid等激活函数而言，ReLU由于非负区间的导数为1，因此可以缓解梯度消失问题(Vanishing Gradient Problem)，使得模型的收敛速度维持在一个稳定状态。这里稍微描述一下什么是梯度消失问题：当梯度小于1时，预测值与真实值之间的误差每传播一层会衰减一次，如果在深层模型中使用sigmoid作为激活函数，这种现象尤为明显，将导致模型收敛停滞不前。

池化层：



上图所展示的是区域最大，即上图左边部分中 左上角2x2的矩阵中6最大，右上角2x2的矩阵中8最大，左下角2x2的矩阵中3最大，右下角2x2的矩阵中4最大，所以得到上图右边部分的结果：6 8 3 4。

全连接层：



softmax直白来说就是将原来输出是3,1,-3通过softmax函数一作用，就映射成为(0,1)的值， 而这些值的累和为1（满足概率的性质），那么我们就可以将它理解成概率。在训练网络时可以通过对预测值和标签值计算交叉熵损失。

# 正则化实验

## 实验介绍

一般而言，正则化是为了防止过拟合或者帮助优化。本实验会给出几种神经网络中最受欢迎的正则化方法，以及用MindSpore实现：提前停止，L2正则化，dropout。此外该实验也增加了Batch Normalization归一化方法优化模型效果。

## 实验环境要求

ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

## 实验总体设计

通过构建加入噪音的cosine模型，加入各种正则化技术形成对比。

提前停止：当验证集的性能开始下降时停止训练。

L2正则化是简单地将L2正则项加在成本函数中。

Dropout 是非常有用和成功的一种技术，会随机删除一些神经元，以在不同批量上训练不同的神经网络架构。

Batch normalization就是通过一定的标准化手段，调整每层神经网络任意神经元的均值和方差，来避免由于不同神经元均值和方差相差过大导致训练网络时出现协变量偏移的问题。

## 实验过程

本节将详细介绍实验的设计与实现。

下载调用包；

超参数设定；

建立数据集；

建立网络及输出。

### 下载调用包

import random

import numpy as np

import matplotlib.pyplot as plt

#导入mindspore框架

import mindspore as ms

#导入mindspore中Neural Networks(nn)模块，包含预先定义的构建块或计算单元来构建神经网络。

from mindspore import nn

#导入mindspore中context模块，用于配置当前执行环境，包括执行模式等特性。

from mindspore import context

#导入参数初始化模块

from mindspore.common.initializer import Normal

from IPython.display import clear\_output

%matplotlib inline

# 设置MindSpore的执行模式和设备

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend')

### 超参数设定

N\_SAMPLES = 40 #样本数

BATCH\_SIZE = 40 #批量大小

NOISE\_RATE = 0.2 #噪声率

INPUT\_DIM = 1 #输入维度

HIDDEN\_DIM = 100 #隐藏层维度

OUTPUT\_DIM = 1 #输出维度

N\_LAYERS = 6 #隐藏层数目

ITERATION = 1500 #最大输入迭代

LEARNING\_RATE = 0.003 #学习率

DROPOUT\_RATE = 0.7

WEIGHT\_DECAY = 1e-4 #L2正则化惩罚数值

MAX\_COUNT = 20 #最早终止参数

ACTIVATION = nn.LeakyReLU #激活函数

#固定结果

def fix\_seed(seed=1):

# reproducible

random.seed(seed)

np.random.seed(seed)

# 小批量样本索引

def sample\_idx(m, n):

A = np.random.permutation(m)

idx = A[:n]

return idx

### 建立数据集

#建立数据集x值，y值

fix\_seed(5)

data\_x = np.linspace(-1, 1, num=int(N\_SAMPLES\*2.5))[:, np.newaxis]

data\_y = np.cos(np.pi\*data\_x)

p = np.random.permutation(len(data\_x))

#建立训练集，测试集，验证集

train\_x, train\_y = data\_x[p[0:N\_SAMPLES]], data\_y[p[0:N\_SAMPLES]]

test\_x, test\_y = data\_x[p[N\_SAMPLES:N\_SAMPLES\*2]], data\_y[p[N\_SAMPLES:N\_SAMPLES\*2]]

validate\_x, validate\_y = data\_x[p[N\_SAMPLES\*2:]], data\_y[p[N\_SAMPLES\*2:]]

#设置y值噪声

noise = np.random.normal(0, NOISE\_RATE, train\_y.shape)

train\_y += noise

### 建立网络

#### 模型定义

#自定义Cosine网络

class CosineNet(nn.Cell):

def \_\_init\_\_(self, batchnorm, dropout):

super(CosineNet, self).\_\_init\_\_()

layers = []

if batchnorm:

layers.append(nn.BatchNorm2d(INPUT\_DIM))

# 初始化隐含层

for l\_n in range(N\_LAYERS):

in\_channels = HIDDEN\_DIM if l\_n > 0 else INPUT\_DIM

# 这里使用1x1Conv代替全连接算子，可以与BatchNorm2d算子配合的更好

conv = nn.Conv2d(in\_channels, HIDDEN\_DIM, kernel\_size=1, pad\_mode='valid', has\_bias=True, weight\_init=Normal(0.01))

layers.append(conv)

if batchnorm:

layers.append(nn.BatchNorm2d(HIDDEN\_DIM))

if dropout:

layers.append(nn.Dropout(DROPOUT\_RATE))

layers.append(ACTIVATION())

self.layers = nn.SequentialCell(layers)

# 初始化输出层

self.flatten = nn.Flatten() # 将(N,C,H,W)4维数据转为(N,C\*H\*W)2维

self.fc = nn.Dense(HIDDEN\_DIM, OUTPUT\_DIM, weight\_init=Normal(0.1), bias\_init='zeros')

def construct(self, x):

# 构建隐含层

x = self.layers(x)

# 构建输出层

x = self.flatten(x)

x = self.fc(x)

return x

#### 针对集中正则化方法，创建不同的训练任务

def build\_fn(batchnorm, dropout, l2):

# 实例化网络、Loss、optimizer

net = CosineNet(batchnorm=batchnorm, dropout=dropout)

loss = nn.loss.MSELoss()

opt = nn.optim.Adam(net.trainable\_params(), learning\_rate=LEARNING\_RATE, weight\_decay=WEIGHT\_DECAY if l2 else 0.0)

# 构建计算loss和训练用的模块

with\_loss = nn.WithLossCell(net, loss)

train\_step = nn.TrainOneStepCell(with\_loss, opt).set\_train()

return train\_step, with\_loss, net

# 针对5种不同设置，创建不同的训练任务

fc\_train, fc\_loss, fc\_predict = build\_fn(batchnorm=False, dropout=False, l2=False) # 默认任务

dropout\_train, dropout\_loss, dropout\_predict = build\_fn(batchnorm=False, dropout=True, l2=False) # 实验dropout功能

bn\_train, bn\_loss, bn\_predict = build\_fn(batchnorm=True, dropout=False, l2=False) # 实验batchnorm功能

l2\_train, l2\_loss, l2\_predict = build\_fn(batchnorm=False, dropout=False, l2=True) # 实验l2 regularization功能

early\_stop\_train, early\_stop\_loss, early\_stop\_predict = build\_fn(batchnorm=False, dropout=False, l2=False) # 实验Early Stop功能

# 辅助函数，用于设置网络是否为train状态，用于batchnorm，dropout等算子判断是否处于train状态。

nets\_train = [fc\_train, dropout\_train, bn\_train, l2\_train, early\_stop\_train]

nets\_loss = [fc\_loss, dropout\_loss, bn\_loss, l2\_loss, early\_stop\_loss]

nets\_predict = [fc\_predict, dropout\_predict, bn\_predict, l2\_predict, early\_stop\_predict]

def set\_train(nets, mode=True):

for net in nets:

net.set\_train(mode=mode)

### 启动训练

#### 将数据转为4维，并转为MindSpore Tensor类型

# 将喂给网络的数据由(N,C)2维转为将(N,C,H,W)4维

data\_xt, data\_yt = ms.Tensor(data\_x.reshape(data\_x.shape + (1, 1)), ms.float32), ms.Tensor(data\_y, ms.float32)

test\_xt, test\_yt = ms.Tensor(test\_x.reshape(test\_x.shape + (1, 1)), ms.float32), ms.Tensor(test\_y, ms.float32)

validate\_xt, validate\_yt = ms.Tensor(validate\_x.reshape(validate\_x.shape + (1, 1)), ms.float32), ms.Tensor(validate\_y, ms.float32)

#### 启动训练并通过plot观察各模型拟合效果

# 设置提前终止(Early Stop)用到的一些指标

early\_stop = False # 为True时，终止相关的训练

min\_val\_loss = 1 # 足够大的初始值，训练过程中用于记录最小的验证loss

count = 0 # 训练迭代过程中，验证loss连续多少次大于min\_val\_loss

for it in range(ITERATION):

# 每个迭代随机从训练集中选择一个batch的样本，当batch\_size==N\_SAMPLES时，仅作了shuffle

mb\_idx = sample\_idx(N\_SAMPLES, BATCH\_SIZE)

x\_batch, y\_batch = train\_x[mb\_idx, :], train\_y[mb\_idx, :]

x\_batch, y\_batch = ms.Tensor(x\_batch.reshape(x\_batch.shape + (1, 1)), ms.float32), ms.Tensor(y\_batch, ms.float32)

set\_train(nets\_train, True) # 将网络设置为train状态

fc\_train(x\_batch, y\_batch)

dropout\_train(x\_batch, y\_batch)

bn\_train(x\_batch, y\_batch)

l2\_train(x\_batch, y\_batch)

# 为True时，终止相关的训练

if not early\_stop:

early\_stop\_train(x\_batch, y\_batch)

if it % 20 == 0:

set\_train(nets\_loss+nets\_predict, False) # 将网络设置为非train状态

# 计算各模型在测试集上的loss

loss\_fc = fc\_loss(test\_xt, test\_yt)

loss\_dropout = dropout\_loss(test\_xt, test\_yt)

loss\_bn = bn\_loss(test\_xt, test\_yt)

loss\_l2 = l2\_loss(test\_xt, test\_yt)

loss\_early\_stop = early\_stop\_loss(test\_xt, test\_yt)

# 计算各模型在全量样本上的预测值，用于评估模型的拟合效果

all\_fc = fc\_predict(data\_xt)

all\_dropout = dropout\_predict(data\_xt)

all\_bn = bn\_predict(data\_xt)

all\_l2 = l2\_predict(data\_xt)

all\_early\_stop = early\_stop\_predict(data\_xt)

# 对于Early Stop任务，当验证集loss连续MAX\_COUNT次大于min\_val\_loss时，终止该任务的训练

if not early\_stop:

val\_loss = early\_stop\_loss(validate\_xt, validate\_yt)

if val\_loss > min\_val\_loss:

count += 1

else:

min\_val\_loss = val\_loss

count = 0

if count == MAX\_COUNT:

early\_stop = True

print('='\*10, 'early stopped', '='\*10)

# 画图

plt.figure(1, figsize=(15,10))

plt.cla()

plt.scatter(train\_x, train\_y, c='magenta', s=50, alpha=0.3, label='train samples')

plt.scatter(test\_x, test\_y, c='cyan', s=50, alpha=0.3, label='test samples')

plt.plot(data\_x, all\_fc.asnumpy(), 'r', label='overfitting')

plt.plot(data\_x, all\_l2.asnumpy(), 'y', label='L2 regularization')

plt.plot(data\_x, all\_early\_stop.asnumpy(), 'k', label='early stopping')

plt.plot(data\_x, all\_dropout.asnumpy(), 'b', label='dropout({})'.format(DROPOUT\_RATE))

plt.plot(data\_x, all\_bn.asnumpy(), 'g', label='batch normalization')

plt.text(-0.1, -1.2, 'overfitting loss=%.4f' % loss\_fc.asnumpy(), fontdict={'size': 20, 'color': 'red'})

plt.text(-0.1, -1.5, 'L2 regularization loss=%.4f' % loss\_l2.asnumpy(), fontdict={'size': 20, 'color': 'y'})

plt.text(-0.1, -1.8, 'early stopping loss=%.4f' % loss\_early\_stop.asnumpy(), fontdict={'size': 20, 'color': 'black'})

plt.text(-0.1, -2.1, 'dropout loss=%.4f' % loss\_dropout.asnumpy(), fontdict={'size': 20, 'color': 'blue'})

plt.text(-0.1, -2.4, 'batch normalization loss=%.4f' % loss\_bn.asnumpy(), fontdict={'size': 20, 'color': 'green'})

plt.legend(loc='upper left');

plt.ylim((-2.5, 2.5));

clear\_output(wait=True)

plt.show()

最终结果如下图：



## 实验总结

本章提供了一个基于华为MindSpore框架的正则化实验。该实验构建简单的cosine模型并且加入正则化技术形成动态图。可以看到：

1. 带有dropout和l2正则化的两个模型在全量数据集上拟合的曲线更平滑，更接近真实的cosine曲线。
2. 不带任何正则化的模型以及带有batchnorm的两个模型对训练数据的拟合程度太高，曲线多曲折，属于过拟合现象。
3. 带有Early Stop的模型处于折中的状态。
4. **FashionMnist分类任务正则化对比实验**
   1. 实验介绍

本实验使用Fashion-MNIST数据集，它是一个替代MNIST手写数字集的图像数据集，由Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自10种类别的共7万个不同商品的正面图片。Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。60000/10000的训练测试数据划分，28x28的灰度图片。通过上述实验我们对比不同正则化技术效果，此实验我们应用正则化技术做案例分析，并对比有无正则化的训练结果。

* 1. 实验环境要求
* ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

* 1. 实验总体设计
  2. 实验过程

本节将详细介绍实验的设计与实现。

数据准备；

导入实验环境；

数据读取和预处理；

训练模型；

观察总结。

* + 1. 数据准备

下载和解压数据集

本实验需要用到的是Fashion-MNIST数据集，由于华为云下载该数据集会花费较多时间，因此可以选择从OBS上下载并解压到notebook的环境中。只下载一次就可。

#从OBS桶下载数据集

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/fashion-mnist.zip

#解压文件

!unzip fashion-mnist.zip

Fashion Mnist有10个标签，如下表所示：

|  |  |
| --- | --- |
| **标签** | **描述** |
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

* + 1. 导入实验环境

导入库

import os

import struct

import sys

from easydict import EasyDict as edict

import matplotlib.pyplot as plt

import numpy as np

import mindspore

import mindspore.dataset as ds

import mindspore.nn as nn

from mindspore import context

from mindspore.nn.metrics import Accuracy

from mindspore.train import Model

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

from mindspore import Tensor

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend')

定义常量

cfg = edict({

'train\_size': 60000, # 训练集大小

'test\_size': 10000, # 测试集大小

'channel': 1, # 图片通道数

'image\_height': 28, # 图片高度

'image\_width': 28, # 图片宽度

'batch\_size': 64,

'num\_classes': 10, # 分类类别

'lr': 0.001, # 学习率

'epoch\_size': 20, # 训练次数

'data\_dir\_train': os.path.join('fashion-mnist', 'train'),

'data\_dir\_test': os.path.join('fashion-mnist', 'test'),

})

### 数据读取和预处理

定义函数用于读取数据

def read\_image(file\_name):

'''

:param file\_name: 文件路径

:return: 训练或者测试数据

如下是训练的图片的二进制格式

[offset] [type] [value] [description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 60000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

........

xxxx unsigned byte ?? pixel

'''

file\_handle = open(file\_name, "rb") # 以二进制打开文档

file\_content = file\_handle.read() # 读取到缓冲区中

head = struct.unpack\_from('>IIII', file\_content, 0) # 取前4个整数，返回一个元组

offset = struct.calcsize('>IIII')

imgNum = head[1] # 图片数

width = head[2] # 宽度

height = head[3] # 高度

bits = imgNum \* width \* height # data一共有60000\*28\*28个像素值

bitsString = '>' + str(bits) + 'B' # fmt格式：'>47040000B'

imgs = struct.unpack\_from(bitsString, file\_content, offset) # 取data数据，返回一个元组

imgs\_array = np.array(imgs, np.float32).reshape((imgNum, width \* height)) # 最后将读取的数据reshape成 【图片数，图片像素】二维数组

return imgs\_array

def read\_label(file\_name):

'''

:param file\_name:

:return:

标签的格式如下：

[offset] [type] [value] [description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 60000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

........

xxxx unsigned byte ?? label

The labels values are 0 to 9.

'''

file\_handle = open(file\_name, "rb") # 以二进制打开文档

file\_content = file\_handle.read() # 读取到缓冲区中

head = struct.unpack\_from('>II', file\_content, 0) # 取前2个整数，返回一个元组

offset = struct.calcsize('>II')

labelNum = head[1] # label数

bitsString = '>' + str(labelNum) + 'B' # fmt格式：'>47040000B'

label = struct.unpack\_from(bitsString, file\_content, offset) # 取data数据，返回一个元组

return np.array(label, np.int32)

def get\_data():

# 文件获取

train\_image = os.path.join(cfg.data\_dir\_train, 'train-images-idx3-ubyte')

test\_image = os.path.join(cfg.data\_dir\_test, "t10k-images-idx3-ubyte")

train\_label = os.path.join(cfg.data\_dir\_train, "train-labels-idx1-ubyte")

test\_label = os.path.join(cfg.data\_dir\_test, "t10k-labels-idx1-ubyte")

# 读取数据

train\_x = read\_image(train\_image)

test\_x = read\_image(test\_image)

train\_y = read\_label(train\_label)

test\_y = read\_label(test\_label)

return train\_x, train\_y, test\_x, test\_y

数据预处理

图像最后一个维度，即通道（颜色），使用reshape()函数将其添加到train\_images和test\_images的维度中。在这种情况下，它是单一颜色，因此通道为1，即“灰度”。为了减少计算量，还需要把图片的像素值进行归一化，八位图像的像素值范围在0-255之间，将所有像素值除以255，使得像素值范围控制在0-1之间。并打印数据集形状和一张图片作为例子。

train\_x, train\_y, test\_x, test\_y = get\_data()

train\_x = train\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

test\_x = test\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

train\_x = train\_x / 255.0

test\_x = test\_x / 255.0

print('训练数据集样本数：', train\_x.shape[0])

print('测试数据集样本数：', test\_y.shape[0])

print('通道数/图像长/宽：', train\_x.shape[1:])

print('一张图像的标签样式：', train\_y[0]) # 一共10类，用0-9的数字表达类别。

plt.figure()

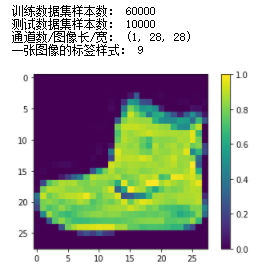
plt.imshow(train\_x[0,0,...])

plt.colorbar()

plt.grid(False)

plt.show()

输出结果：



数据集预处理

在训练之前，需要先对数据集中的数据进行“洗牌”，打乱数据集的顺序。

# 转换数据类型为Dataset

def create\_dataset():

XY\_train = list(zip(train\_x, train\_y))

ds\_train = ds.GeneratorDataset(XY\_train, ['x', 'y'])

ds\_train = ds\_train.shuffle(buffer\_size=1000).batch(cfg.batch\_size, drop\_remainder=True)

XY\_test = list(zip(test\_x, test\_y))

ds\_test = ds.GeneratorDataset(XY\_test, ['x', 'y'])

ds\_test = ds\_test.shuffle(buffer\_size=1000).batch(cfg.batch\_size, drop\_remainder=True)

return ds\_train, ds\_test

### 训练模型

创建卷积神经网络

该实验中，可以选择使用不含正则化的卷积神经网络或者选择加入正则化的卷积神经网络，用于对比结果。

下面这部分用于创建不加入正则化的卷积神经网络，网络结构为：卷积层1🡪卷积层2🡪卷积层3🡪最大池化层🡪全连接层1🡪全连接层2。

# 定义卷积神经网络，无正则化

class ForwardFashion(nn.Cell):

def \_\_init\_\_(self, num\_class=10): # 一共分十类，图片通道数是1

super(ForwardFashion, self).\_\_init\_\_()

self.num\_class = num\_class

self.conv1 = nn.Conv2d(1, 32,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv2 = nn.Conv2d(32, 64,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv3 = nn.Conv2d(64, 128,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.maxpool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.relu = nn.ReLU()

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(128 \* 11 \* 11, 128)

self.fc2 = nn.Dense(128, self.num\_class)

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.conv2(x)

x = self.relu(x)

x = self.conv3(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

return x

下面这部分用于创建加入正则化的卷积神经网络，网络结构为卷积层1🡪dropout层1🡪卷积层2🡪dropout层1🡪卷积层3🡪dropout层1🡪最大池化层🡪 dropout层2🡪全连接层1🡪 dropout层2🡪全连接层2：

# 定义卷积神经网络，有正则化

class ForwardFashionRegularization(nn.Cell):

def \_\_init\_\_(self, num\_class=10): # 一共分十类，图片通道数是1

super(ForwardFashionRegularization, self).\_\_init\_\_()

self.num\_class = num\_class

self.conv1 = nn.Conv2d(1, 32,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv2 = nn.Conv2d(32, 64,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv3 = nn.Conv2d(64, 128,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.maxpool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.relu = nn.ReLU()

self.dropout = nn.Dropout()

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(3200, 128)

self.bn = nn.BatchNorm1d(128)

self.fc2 = nn.Dense(128, self.num\_class)

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.conv2(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.dropout(x)

x = self.conv3(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.dropout(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.bn(x)

x = self.dropout(x)

x = self.fc2(x)

return x

启动训练

为这个模型指定优化器（adam）、损失函数（crossentropy）和度量(accuracy)，然后启动训练，最后进行验证。

def train(Net):

ds\_train, ds\_test = create\_dataset()

# 构建网络

network = Net(cfg.num\_classes)

# 定义模型的损失函数，优化器

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

net\_opt = nn.Adam(network.trainable\_params(), cfg.lr)

# 训练模型

model = Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'acc': Accuracy()})

loss\_cb = LossMonitor()

print("============== Starting Training ==============")

model.train(30, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=True)

# 验证

metric = model.eval(ds\_test)

print(metric)

return model

训练并验证无正则化的网络。

# 训练无正则化的网络

model = train(ForwardFashion)

输出结果：

============== Starting Training ==============

epoch: 1 step: 937, loss is 0.36209568

epoch: 2 step: 937, loss is 0.11113132

epoch: 3 step: 937, loss is 0.107788906

epoch: 4 step: 937, loss is 0.12908919

epoch: 5 step: 937, loss is 0.078461185

epoch: 6 step: 937, loss is 0.18977618

epoch: 7 step: 937, loss is 0.15168177

epoch: 8 step: 937, loss is 0.06739945

epoch: 9 step: 937, loss is 0.14379226

epoch: 10 step: 937, loss is 0.076876596

epoch: 11 step: 937, loss is 0.055951692

epoch: 12 step: 937, loss is 0.022819532

epoch: 13 step: 937, loss is 0.10054826

epoch: 14 step: 937, loss is 0.012528818

epoch: 15 step: 937, loss is 0.0076259132

epoch: 16 step: 937, loss is 0.07877082

epoch: 17 step: 937, loss is 0.031406786

epoch: 18 step: 937, loss is 0.009203883

epoch: 19 step: 937, loss is 0.005287296

epoch: 20 step: 937, loss is 0.0929834

epoch: 21 step: 937, loss is 0.0015465739

epoch: 22 step: 937, loss is 0.03491202

epoch: 23 step: 937, loss is 0.0005662525

epoch: 24 step: 937, loss is 0.010102608

epoch: 25 step: 937, loss is 0.003999765

epoch: 26 step: 937, loss is 0.011775437

epoch: 27 step: 937, loss is 0.080310896

epoch: 28 step: 937, loss is 0.0018242185

epoch: 29 step: 937, loss is 0.007360851

epoch: 30 step: 937, loss is 0.003999797

{'acc': 0.9147636217948718}

训练并验证有正则化的网络。

# 训练有正则化的网络

model = train(ForwardFashionRegularization)

输出结果：

============== Starting Training ==============

epoch: 1 step: 937, loss is 0.29856867

epoch: 2 step: 937, loss is 0.28910726

epoch: 3 step: 937, loss is 0.18035105

epoch: 4 step: 937, loss is 0.2785972

epoch: 5 step: 937, loss is 0.21400028

epoch: 6 step: 937, loss is 0.27920294

epoch: 7 step: 937, loss is 0.17452516

epoch: 8 step: 937, loss is 0.309029

epoch: 9 step: 937, loss is 0.30411178

epoch: 10 step: 937, loss is 0.2842149

epoch: 11 step: 937, loss is 0.22666603

epoch: 12 step: 937, loss is 0.16507925

epoch: 13 step: 937, loss is 0.17004505

epoch: 14 step: 937, loss is 0.23396353

epoch: 15 step: 937, loss is 0.20207018

epoch: 16 step: 937, loss is 0.43118417

epoch: 17 step: 937, loss is 0.23762615

epoch: 18 step: 937, loss is 0.24660718

epoch: 19 step: 937, loss is 0.12197974

epoch: 20 step: 937, loss is 0.22719634

epoch: 21 step: 937, loss is 0.2809552

epoch: 22 step: 937, loss is 0.21124852

epoch: 23 step: 937, loss is 0.2100177

epoch: 24 step: 937, loss is 0.29766798

epoch: 25 step: 937, loss is 0.115716025

epoch: 26 step: 937, loss is 0.41360933

epoch: 27 step: 937, loss is 0.11700327

epoch: 28 step: 937, loss is 0.2552187

epoch: 29 step: 937, loss is 0.14747506

epoch: 30 step: 937, loss is 0.19088028

{'acc': 0.9234775641025641}

预测模型

使用上述训练好的模型对测试数据集进行预测。打印预测结果

# 预测

ds\_test, \_ = create\_dataset()

test\_ = ds\_test.create\_dict\_iterator(output\_numpy=True).\_\_next\_\_()

predictions = model.predict(Tensor(test\_['x']))

predictions = predictions.asnumpy()

for i in range(15):

p\_np = predictions[i, :]

p\_list = p\_np.tolist()

print('第' + str(i) + '个sample预测结果：', p\_list.index(max(p\_list)), ' 真实结果：', test\_['y'][i])

输出结果：

第0个sample预测结果： 3 真实结果： 3

第1个sample预测结果： 5 真实结果： 5

第2个sample预测结果： 6 真实结果： 2

第3个sample预测结果： 4 真实结果： 4

第4个sample预测结果： 3 真实结果： 3

第5个sample预测结果： 3 真实结果： 3

第6个sample预测结果： 7 真实结果： 7

第7个sample预测结果： 8 真实结果： 8

第8个sample预测结果： 3 真实结果： 3

第9个sample预测结果： 5 真实结果： 5

第10个sample预测结果： 9 真实结果： 9

第11个sample预测结果： 2 真实结果： 4

第12个sample预测结果： 6 真实结果： 6

第13个sample预测结果： 5 真实结果： 5

第14个sample预测结果： 0 真实结果： 0

可视化结果

定义可视化函数。

# -------------------定义可视化函数--------------------------------

# 输入预测结果序列，真实标签序列，以及图片序列

# 目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为红色；错：标签为蓝色

def plot\_image(predictions\_array, true\_label, img):

plt.grid(False)

plt.xticks([])

plt.yticks([])

# 显示对应图片

plt.imshow(img, cmap=plt.cm.binary)

# 显示预测结果的颜色，如果对上了是蓝色，否则为红色

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

# 显示对应标签的格式，样式

plt.xlabel('{},{:2.0f}% ({})'.format(class\_names[predicted\_label],

100 \* np.max(predictions\_array),

class\_names[true\_label]), color=color)

# 将预测的结果以柱状图形状显示蓝对红错

def plot\_value\_array(predictions\_array, true\_label):

plt.grid(False)

plt.xticks([])

plt.yticks([])

this\_plot = plt.bar(range(10), predictions\_array, color='#777777')

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

this\_plot[predicted\_label].set\_color('red')

this\_plot[true\_label].set\_color('blue')

import numpy as np

def softmax\_np(x):

x = x - np.max(x)

exp\_x = np.exp(x)

softmax\_x = exp\_x/np.sum(exp\_x)

return softmax\_x

预测结果可视化，输入预测结果序列，真实标签序列，以及图片序列。目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为蓝色；错：标签为红色。最后预测15个图像与标签，将预测的结果以柱状图形状显示蓝对红错。

# 预测15个图像与标签，并展现出来

num\_rows = 5

num\_cols = 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

for i in range(num\_images):

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

pred\_np\_ = predictions[i, :]

pred\_np\_ = softmax\_np(pred\_np\_)

plot\_image(pred\_np\_, test\_['y'][i], test\_['x'][i, 0, ...])

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 2)

plot\_value\_array(pred\_np\_, test\_['y'][i])

plt.show()

输出结果：



### 观察总结

如果神经网络具有较少的层数和神经元数量，则它可能在训练数据集上表现不佳，即可能导致欠拟合问题。但是，如果神经网络具有很多的层数和神经元数量，它可能在训练数据集上表现得很好（高精度），但是可能在测试数据集上表现不好，即可能导致过拟合问题。

不幸的是，没有确定公式来确定模型的正确结构（层数和每层中的神经元数量），搭建网络时将需要不断尝试，来找到表现不错的架构。

虽然对于神经网络，我们还有其他正则化技术，如L1, L2正则化，Dropout是所有正则化技术中最有效的，也是最常用的。Dropout会随机地丢弃层的一些输出特征（神经元），即设置为零。假设在应用Dropout之前层的输出为[0.5, 0.8, 2.2, 0.9, 0.1]，在应用Dropout之后，输出将为[0, 0.8, 2.2, 0, 0.1]。“Dropout率”是层中被置为零的特征（神经元）的分数。一般来说，Dropout率保持在0.2至0.5之间。

请注意，Dropout只在训练阶段使用。在对测试数据集进行预测时，我们不需要从模型中手动移除Dropout层。 在测试阶段，不会应用Dropout。

因此，为了减少本案例中的过拟合问题，让我们应用正则化技术-Dropout来提升测试表现。

* 1. 实验总结

本章提供了一个FashionMnist正则化前后对比实验。该实验选取FashionMnist灰度数据集将模型进行训练预测，当初始模型表现过拟合时，参数量多的模型的性能反而不如参数量少的模型，我们加入正则化技术重新建立新模型并预测。

# 花卉图像分类实验

## 实验介绍

随着电子技术的迅速发展，人们使用便携数码设备（如手机、相机等）获取花卉图像越来越方便，如何自动识别花卉种类受到了广泛的关注。由于花卉所处背景的复杂性，以及花卉自身的类间相似性和类内多样性，利用传统的手工提取特征进行图像分类的方法，并不能很好地解决花卉图像分类这一问题。

本实验为基于卷积神经网络实现的花卉识别实验，与传统图像分类方法不同，卷积神经网络无需人工提取特征，可以根据输入图像，自动学习包含丰富语义信息的特征，得到更为全面的花卉图像特征描述，可以很好地表达图像的不同类别信息。

## 实验环境要求

ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

## 实验总体设计

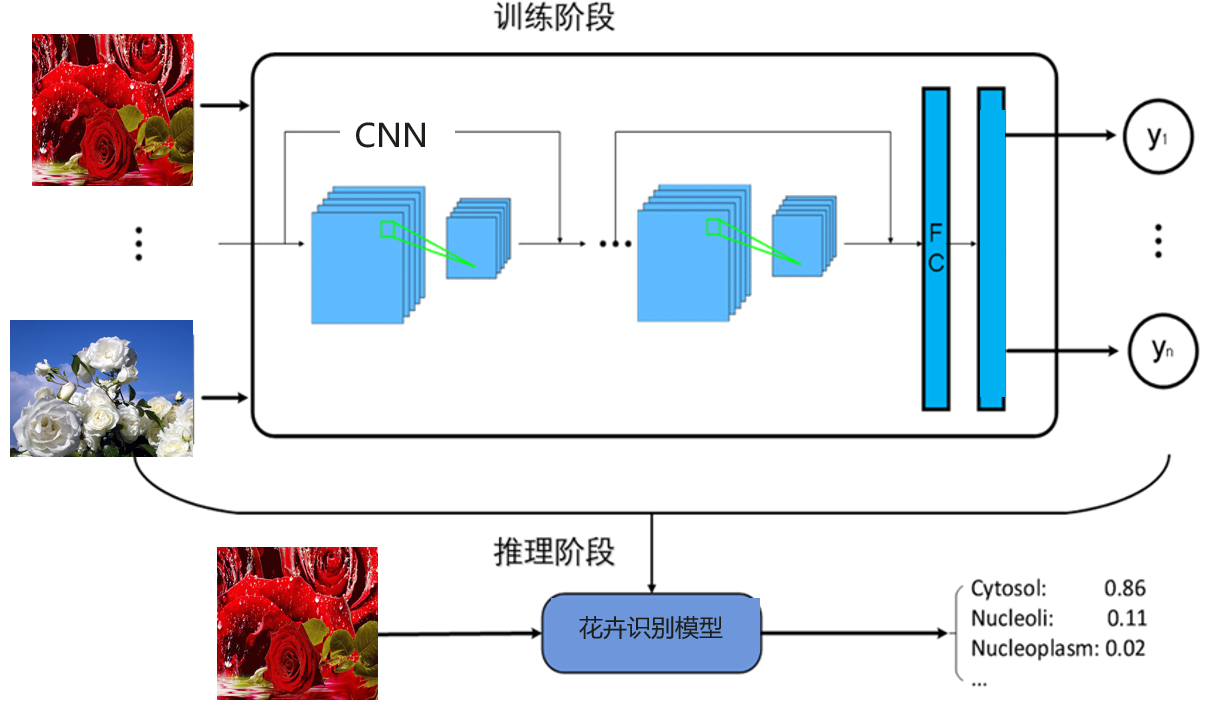
本实验主要介绍如何使用MindSpore进行花卉图像分类实验。定义卷积神经网络，并利于该网络进行花卉分类训练和测试。

### 功能结构

该实验可以划分为数据处理、模型构建、图像识别三个主要的子实验。其中数据处理子实验包括数据集划分、图像预处理两个部分；模型构建子实验主要包括模型定义、模型训练两个部分；图像识别子实验主要包括加载、运行推断模型进行图像特征提取并输出模型识别结果两个部分。

### 体系结构

按照体系结构划分，整个实验的体系结构可以划分为三部分，分别为模型训练、模型保存和模型推理，如图4-1所示。各层侧重点各不相同。训练层运行在安装有MindSpore框架的服务器，最好配置计算加速卡。推断层运行于开发环境，能够支持卷积神经网络的加速。展示层运行于客户端应用程序，能够完成图像选择并实时显示推断层的计算结果。各层之间存在单向依赖关系。推断层需要的网络模型由训练层提供，并根据需要进行必要的格式转换或加速重构。相应的，展示层要显示的元数据需要由推断层计算得到。



## 实验过程

本节将详细介绍实验的设计与实现。

导入实验环境；

数据集获取与预处理；

构建CNN图像识别模型；

图像分类模型验证；

### 导入实验环境

导入相应的模块

Glob模块主要用于查找符合特定规则的文件路径名，类似使用windows下的文件搜索； os模块主要用于处理文件和目录，比如：获取当前目录下文件，删除制定文件，改变目录，查看文件大小等；MindSpore是目前业界流行的深度学习框架之一，在图像，语音，文本，目标检测等领域都有深入的应用，也是该实验的核心，主要用于定义占位符，定义变量，创建卷积神经网络模型；numpy是一个基于python的科学计算包，在该实验中主要用来处理数值运算。

#easydict模块用于以属性的方式访问字典的值

from easydict import EasyDict as edict

#glob模块主要用于查找符合特定规则的文件路径名，类似使用windows下的文件搜索

import glob

#os模块主要用于处理文件和目录

import os

import numpy as np

import matplotlib.pyplot as plt

import mindspore

#导入mindspore框架数据集

import mindspore.dataset as ds

#vision.c\_transforms模块是处理图像增强的高性能模块，用于数据增强图像数据改进训练模型。

import mindspore.dataset.vision.c\_transforms as CV

#c\_transforms模块提供常用操作，包括OneHotOp和TypeCast

import mindspore.dataset.transforms.c\_transforms as C

from mindspore.common import dtype as mstype

from mindspore import context

#导入模块用于初始化截断正态分布

from mindspore.common.initializer import TruncatedNormal

from mindspore import nn

from mindspore.train import Model

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore import Tensor

# 设置MindSpore的执行模式和设备

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

定义变量

cfg = edict({

'data\_path': 'flower\_photos',

'data\_size':3670,

'image\_width': 100, # 图片宽度

'image\_height': 100, # 图片高度

'batch\_size': 32,

'channel': 3, # 图片通道数

'num\_class':5, # 分类类别

'weight\_decay': 0.01,

'lr':0.0001, # 学习率

'dropout\_ratio': 0.5,

'epoch\_size': 400, # 训练次数

'sigma':0.01,

'save\_checkpoint\_steps': 1, # 多少步保存一次模型

'keep\_checkpoint\_max': 1, # 最多保存多少个模型

'output\_directory': './', # 保存模型路径

'output\_prefix': "checkpoint\_classification" # 保存模型文件名字

})

### 数据集获取与预处理

该数据集是开源数据集，总共包括5种花的类型：分别是daisy（雏菊，633张），dandelion（蒲公英，898张），roses（玫瑰，641张），sunflowers（向日葵，699张），tulips（郁金香，799张），保存在5个文件夹当中，总共3670张，大小大概在230M左右。为了在模型部署上线之后进行测试，数据集在这里分成了flower\_train和flower\_test两部分。

数据读取并处理流程如下：

MindSpore的mindspore.dataset提供了ImageFolderDatasetV2函数，可以直接读取文件夹图片数据并映射文件夹名字为其标签(label)。这里我们使用ImageFolderDatasetV2函数 读取'daisy','dandelion','roses','sunflowers','tulips'数据。并将这五类标签映射为： {'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4}

使用RandomCropDecodeResize、HWC2CHW、TypeCast、shuffle进行数据预处理

获取数据集

# 解压数据集，只需要第一次运行时解压，第二次无需再解压

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos.zip

!unzip flower\_photos.zip

数据预处理

#从目录中读取图像的源数据集。

de\_dataset = ds.ImageFolderDataset(cfg.data\_path,

class\_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})

#解码前将输入图像裁剪成任意大小和宽高比。

transform\_img = CV.RandomCropDecodeResize([cfg.image\_width,cfg.image\_height], scale=(0.08, 1.0), ratio=(0.75, 1.333)) #改变尺寸

#转换输入图像；形状（H, W, C）为形状（C, H, W）。

hwc2chw\_op = CV.HWC2CHW()

#转换为给定MindSpore数据类型的Tensor操作。

type\_cast\_op = C.TypeCast(mstype.float32)

#将操作中的每个操作应用到此数据集。

de\_dataset = de\_dataset.map(input\_columns="image", num\_parallel\_workers=8, operations=transform\_img)

de\_dataset = de\_dataset.map(input\_columns="image", operations=hwc2chw\_op, num\_parallel\_workers=8)

de\_dataset = de\_dataset.map(input\_columns="image", operations=type\_cast\_op, num\_parallel\_workers=8)

de\_dataset = de\_dataset.shuffle(buffer\_size=cfg.data\_size)

划分训练集与测试集

按照8:2的比列将数据划分为训练数据集和测试数据集

对训练数据和测试数据分批次（batch）

#划分训练集测试集

(de\_train,de\_test)=de\_dataset.split([0.8,0.2])

#设置每个批处理的行数

#drop\_remainder确定是否删除最后一个可能不完整的批（default=False）。

#如果为True，并且如果可用于生成最后一个批的batch\_size行小于batch\_size行，则这些行将被删除，并且不会传播到子节点。

de\_train=de\_train.batch(cfg.batch\_size, drop\_remainder=True)

#重复此数据集计数次数。

de\_test=de\_test.batch(cfg.batch\_size, drop\_remainder=True)

print('训练数据集数量：',de\_train.get\_dataset\_size()\*cfg.batch\_size)#get\_dataset\_size()获取批处理的大小。

print('测试数据集数量：',de\_test.get\_dataset\_size()\*cfg.batch\_size)

data\_next=de\_dataset.create\_dict\_iterator(output\_numpy=True).\_\_next\_\_()

print('通道数/图像长/宽：', data\_next['image'].shape)

print('一张图像的标签样式：', data\_next['label']) # 一共5类，用0-4的数字表达类别。

plt.figure()

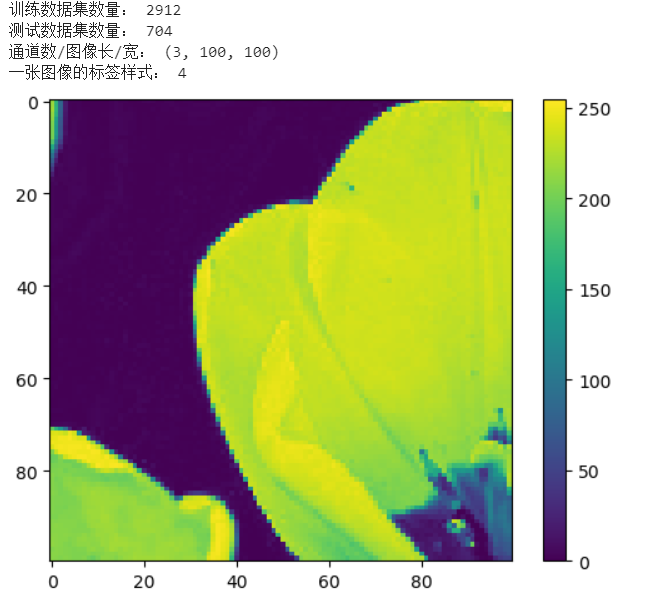
plt.imshow(data\_next['image'][0,...])

plt.colorbar()

plt.grid(False)

plt.show()

输出结果：



### 构建CNN图像识别模型

花卉图像数据集准备完成，接下来我们就需要构建训练模型，本实验采用的是CNN神经网络算法。

定义图像识别模型

# 定义CNN图像识别网络

class Identification\_Net(nn.Cell):

def \_\_init\_\_(self, num\_class=5,channel=3,dropout\_ratio=0.5,trun\_sigma=0.01): # 一共分五类，图片通道数是3

super(Identification\_Net, self).\_\_init\_\_()

self.num\_class = num\_class

self.channel = channel

self.dropout\_ratio = dropout\_ratio

#设置卷积层

self.conv1 = nn.Conv2d(self.channel, 32,

kernel\_size=5, stride=1, padding=0,

has\_bias=True, pad\_mode="same",

weight\_init=TruncatedNormal(sigma=trun\_sigma),bias\_init='zeros')

#设置ReLU激活函数

self.relu = nn.ReLU()

#设置最大池化层

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2,pad\_mode="valid")

self.conv2 = nn.Conv2d(32, 64,

kernel\_size=5, stride=1, padding=0,

has\_bias=True, pad\_mode="same",

weight\_init=TruncatedNormal(sigma=trun\_sigma),bias\_init='zeros')

self.conv3 = nn.Conv2d(64, 128,

kernel\_size=3, stride=1, padding=0,

has\_bias=True, pad\_mode="same",

weight\_init=TruncatedNormal(sigma=trun\_sigma),bias\_init='zeros')

self.conv4 = nn.Conv2d(128, 128,

kernel\_size=3, stride=1, padding=0,

has\_bias=True, pad\_mode="same",

weight\_init=TruncatedNormal(sigma=trun\_sigma), bias\_init='zeros')

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(6\*6\*128, 1024,weight\_init =TruncatedNormal(sigma=trun\_sigma),bias\_init = 0.1)

self.dropout = nn.Dropout(self.dropout\_ratio)

self.fc2 = nn.Dense(1024, 512, weight\_init=TruncatedNormal(sigma=trun\_sigma), bias\_init=0.1)

self.fc3 = nn.Dense(512, self.num\_class, weight\_init=TruncatedNormal(sigma=trun\_sigma), bias\_init=0.1)

#构建模型

def construct(self, x):

x = self.conv1(x)

#print(x.shape)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv2(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv3(x)

x = self.max\_pool2d(x)

x = self.conv4(x)

x = self.max\_pool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

#print(x.shape)

x = self.dropout(x)

x = self.fc2(x)

x = self.relu(x)

x = self.dropout(x)

x = self.fc3(x)

return x

模型训练、测试、预测

net=Identification\_Net(num\_class=cfg.num\_class, channel=cfg.channel, dropout\_ratio=cfg.dropout\_ratio)

#计算softmax交叉熵。

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

#opt

fc\_weight\_params = list(filter(lambda x: 'fc' in x.name and 'weight' in x.name, net.trainable\_params()))

other\_params=list(filter(lambda x: 'fc' not in x.name or 'weight' not in x.name, net.trainable\_params()))

group\_params = [{'params': fc\_weight\_params, 'weight\_decay': cfg.weight\_decay},

{'params': other\_params},

{'order\_params': net.trainable\_params()}]

#设置Adam优化器

net\_opt = nn.Adam(group\_params, learning\_rate=cfg.lr, weight\_decay=0.0)

#net\_opt = nn.Adam(params=net.trainable\_params(), learning\_rate=cfg.lr, weight\_decay=0.1)

model = Model(net, loss\_fn=net\_loss, optimizer=net\_opt, metrics={"acc"})

loss\_cb = LossMonitor(per\_print\_times=de\_train.get\_dataset\_size()\*10)

config\_ck = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps,

keep\_checkpoint\_max=cfg.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix=cfg.output\_prefix, directory=cfg.output\_directory, config=config\_ck)

print("============== Starting Training ==============")

model.train(cfg.epoch\_size, de\_train, callbacks=[loss\_cb, ckpoint\_cb], dataset\_sink\_mode=True)

# 使用测试集评估模型，打印总体准确率

metric = model.eval(de\_test)

print(metric)

输出结果：

============== Starting Training ==============

epoch: 10 step: 91, loss is 1.0415859

epoch: 20 step: 91, loss is 0.9116502

epoch: 30 step: 91, loss is 0.77244914

epoch: 40 step: 91, loss is 0.8834233

epoch: 50 step: 91, loss is 0.7412845

…

epoch: 380 step: 91, loss is 0.08321401

epoch: 390 step: 91, loss is 0.096963525

epoch: 400 step: 91, loss is 0.40712065

{'acc': 0.9161931818181818}

### 图像分类模型验证

验证之前训练出来的模型的性能。

掌握利图像识别模型验证方法。

加载训练模型

#加载模型

import os

CKPT = os.path.join(cfg.output\_directory,cfg.output\_prefix+'-'+str(cfg.epoch\_size)+'\_'+str(de\_train.get\_dataset\_size())+'.ckpt')

net = Identification\_Net(num\_class=cfg.num\_class, channel=cfg.channel, dropout\_ratio=cfg.dropout\_ratio)

load\_checkpoint(CKPT, net=net)

model = Model(net)

验证推理

# 预测

class\_names = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}

test\_ = de\_test.create\_dict\_iterator().\_\_next\_\_()

test = Tensor(test\_['image'], mindspore.float32)

predictions = model.predict(test)

predictions = predictions.asnumpy()

true\_label = test\_['label'].asnumpy()

#显示预测结果

for i in range(10):

p\_np = predictions[i, :]

pre\_label = np.argmax(p\_np)

print('第' + str(i) + '个sample预测结果：', class\_names[pre\_label], ' 真实结果：', class\_names[true\_label[i]])

输出结果：

第0个sample预测结果： sunflowers 真实结果： sunflowers

第1个sample预测结果： roses 真实结果： roses

第2个sample预测结果： roses 真实结果： roses

第3个sample预测结果： roses 真实结果： daisy

第4个sample预测结果： tulips 真实结果： tulips

第5个sample预测结果： tulips 真实结果： tulips

第6个sample预测结果： tulips 真实结果： tulips

第7个sample预测结果： dandelion 真实结果： dandelion

第8个sample预测结果： sunflowers 真实结果： sunflowers

第9个sample预测结果： tulips 真实结果： tulips

## 实验总结

本章提供了一个基于华为MindSpore框架的图像识别实验。该实验演示了如何利用华为云ModelArts完成图像识别任务。本章对实验实验做了详尽的剖析。阐明了整个实验功能、结构与流程是如何设计的，详细解释了如何解析数据、如何构建深度学习模型、如何保存模型等内容。部署后的实验多个类别图片下进行测试，结果表明实验实验具有较快的推断速度和较好的识别性能。读者可以在该实验实验的基础上开发更有针对性的应用实验。

# LeNet的手写数字识别实验

## 实验介绍

LeNet5 + MINST被誉为深度学习领域的“Hello world”。本实验主要介绍使用MindSpore在MNIST数据集上开发和训练一个LeNet5模型，并验证模型精度。

了解如何使用MindSpore进行简单卷积神经网络的开发。

了解如何使用MindSpore进行简单图片分类任务的训练。

了解如何使用MindSpore进行简单图片分类任务的验证。

## 实验环境要求

ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

## 实验准备

数据集准备

MNIST是一个手写数字数据集，训练集包含60000张手写数字，测试集包含10000张手写数字，共10类。

方法1：从MNIST官网下载数据集到本地，MNIST数据集的官网<http://yann.lecun.com/exdb/mnist/>

方法2：从华为云OBS中下载MNIST数据集并解压。 <https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/MNIST.zip>

本实验采用方法2下载数据集。

## 实验总体设计

## 实验过程

本节将详细介绍实验的设计与实现。

导入实验环境；

数据处理；

定义模型；

训练和推理。

### 导入实验环境

import os

# os.environ['DEVICE\_ID'] = '0'

import numpy as np

import mindspore as ms

#导入mindspore中context模块，用于配置当前执行环境，包括执行模式等特性。

import mindspore.context as context

#c\_transforms模块提供常用操作，包括OneHotOp和TypeCast

import mindspore.dataset.transforms.c\_transforms as C

#vision.c\_transforms模块是处理图像增强的高性能模块，用于数据增强图像数据改进训练模型。

import mindspore.dataset.vision.c\_transforms as CV

from mindspore import nn

from mindspore.train import Model

from mindspore.train.callback import LossMonitor

# 设置MindSpore的执行模式和设备

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend') # Ascend, CPU, GPU

### 数据处理

获取数据集

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/MNIST.zip

!unzip MNIST.zip

对数据进行预处理。

def create\_dataset(data\_dir, training=True, batch\_size=32, resize=(32, 32),

rescale=1/(255\*0.3081), shift=-0.1307/0.3081, buffer\_size=64):

data\_train = os.path.join(data\_dir, 'train') # 训练集信息

data\_test = os.path.join(data\_dir, 'test') # 测试集信息

ds = ms.dataset.MnistDataset(data\_train if training else data\_test)

#将操作中的每个操作应用到此数据集。

ds = ds.map(input\_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale, shift), CV.HWC2CHW()])

ds = ds.map(input\_columns=["label"], operations=C.TypeCast(ms.int32))

# When `dataset\_sink\_mode=True` on Ascend, append `ds = ds.repeat(num\_epochs) to the end

ds = ds.shuffle(buffer\_size=buffer\_size).batch(batch\_size, drop\_remainder=True)

return ds

对其中几张图片进行可视化，可以看到图片中的手写数字，图片的大小为32x32。

import matplotlib.pyplot as plt

ds = create\_dataset('MNIST', training=False)

data = ds.create\_dict\_iterator().\_\_next\_\_()

images = data['image'].asnumpy()

labels = data['label'].asnumpy()

#显示前4张图片以及对应标签

for i in range(1, 5):

plt.subplot(2, 2, i)

plt.imshow(images[i][0])

plt.title('Number: %s' % labels[i])

plt.xticks([])

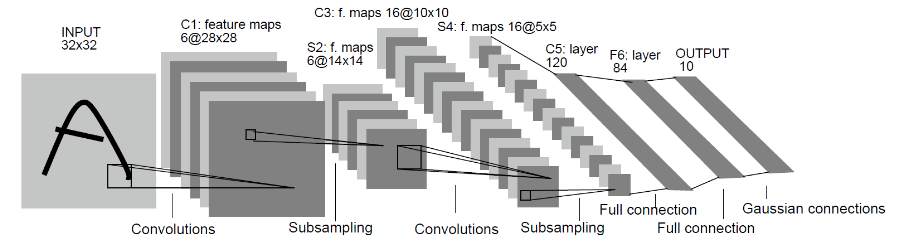
plt.show()

输出：



### 定义模型

MindSpore model\_zoo中提供了多种常见的模型，可以直接使用。LeNet5模型结构如下图所示：



图片来源于http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

#定义LeNet5模型

class LeNet5(nn.Cell):

def \_\_init\_\_(self):

super(LeNet5, self).\_\_init\_\_()

#设置卷积网络（输入输出通道数，卷积核尺寸，步长，填充方式）

self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad\_mode='valid')

self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad\_mode='valid')

self.relu = nn.ReLU()

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(400, 120)

self.fc2 = nn.Dense(120, 84)

self.fc3 = nn.Dense(84, 10)

#构建网络

def construct(self, x):

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

x = self.pool(x)

x = self.relu(self.conv2(x))

x = self.pool(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.fc2(x)

x = self.fc3(x)

return x

### 训练

使用MNIST数据集对上述定义的LeNet5模型进行训练。训练策略如下表所示，可以调整训练策略并查看训练效果，要求验证精度大于95%。

|  |  |  |  |
| --- | --- | --- | --- |
| **batch size** | **number of epochs** | **learning rate** | **optimizer** |
| 32 | 3 | 0.01 | Momentum 0.9 |

def train(data\_dir, lr=0.01, momentum=0.9, num\_epochs=3):

ds\_train = create\_dataset(data\_dir)

ds\_eval = create\_dataset(data\_dir, training=False)

net = LeNet5()

#计算softmax交叉熵。

loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

#设置Momentum优化器

opt = nn.Momentum(net.trainable\_params(), lr, momentum)

loss\_cb = LossMonitor(per\_print\_times=ds\_train.get\_dataset\_size())

model = Model(net, loss, opt, metrics={'acc', 'loss'})

# dataset\_sink\_mode can be True when using Ascend

model.train(num\_epochs, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=True)

metrics = model.eval(ds\_eval, dataset\_sink\_mode=True)

print('Metrics:', metrics)

train('MNIST')

输出结果：

epoch: 1 step 1875, loss is 0.23394052684307098

epoch: 2 step 1875, loss is 0.4737345278263092

epoch: 3 step 1875, loss is 0.07734094560146332

Metrics: {'loss': 0.10531254443608654, 'acc': 0.9701522435897436}

## 实验总结

本实验展示了如何使用MindSpore进行手写数字识别，以及开发和训练LeNet5模型。通过对LeNet5模型做几代的训练，然后使用训练后的LeNet5模型对手写数字进行识别，识别准确率大于95%。即LeNet5学习到了如何进行手写数字识别。

# 图像识别全流程代码实践

## 实验介绍

图像分类在我们的日常生活中广泛使用，比如拍照识物，还有手机的AI拍照，在学术界，每年也有很多图像分类的比赛，本实验将会利用一个开源数据集来帮助大家学习如何构建自己的图像识别模型。本实验会使用MindSpore来构建图像识别模型。

## 实验环境要求

ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

## 实验总体设计

该实验主要步骤包括

1. 数据集获取
2. 导入实验环境
3. 读取数据集
4. 模型构建训练
5. 模型预测
6. 模型保存

注解：其中导入实验环境、数据集准备请参考第四章花卉图像分类实验指导。

## 实验过程

本节将详细介绍实验的设计与实现。

数据集获取；

导入实验环境；

读取数据集；

模型构建训练；

模型预测；

模型保存；

### 数据集获取

该数据集是开源数据集，总共包括5种花的类型：分别是daisy（雏菊，633张），dandelion（蒲公英，898张），roses（玫瑰，641张），sunflowers（向日葵，699张），tulips（郁金香，799张），保存在5个文件夹当中，总共3670张，大小大概在230M左右。为了在模型部署上线之后进行测试，数据集在这里分成了flower\_train和flower\_test两部分。

在ModelArts平台输入代码自动获取数据。

#ModelArts平台输入代码会自动下载数据，下载完成之后不需要二次运行，不然会报错。

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_train.zip

!unzip flower\_photos\_train.zip

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_test.zip

!unzip flower\_photos\_test.zip

### 导入实验环境

导入相应的模块

os模块主要用于处理文件和目录，比如：获取当前目录下文件，删除制定文件，改变目录，查看文件大小等；MindSpore是目前业界最流行的深度学习框架，在图像，语音，文本，目标检测等领域都有深入的应用，也是该实验的核心，主要用于定义占位符，定义变量，创建卷积神经网络模型；numpy是一个基于python的科学计算包，在该实验中主要用来处理数值运算。

from easydict import EasyDict as edict

# 字典访问，用来存储超参数

import os

# os模块主要用于处理文件和目录

import numpy as np

# 科学计算库

import matplotlib.pyplot as plt

# 绘图库

import mindspore

# MindSpore库

import mindspore.dataset as ds

# 数据集处理模块

from mindspore.dataset.vision import c\_transforms as vision

# 图像增强模块

from mindspore import context

# 环境设置模块

import mindspore.nn as nn

# 神经网络模块

from mindspore.train import Model

# 模型编译

from mindspore.nn.optim.momentum import Momentum

# 动量优化器

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor

# 模型保存设置

from mindspore import Tensor

# 张量

from mindspore.train.serialization import export

# 模型导出

from mindspore.train.loss\_scale\_manager import FixedLossScaleManager

# 损失值平滑处理

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

# 模型加载

import mindspore.ops as ops

# 常见算子操作

# 设置MindSpore的执行模式和设备

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

定义变量

cfg = edict({

'data\_path': 'flower\_photos\_train', #训练数据集，如果是zip文件需要解压

'test\_path':'flower\_photos\_test', #测试数据集，如果是zip文件需要解压

'data\_size': 3616,

'HEIGHT': 224, # 图片高度

'WIDTH': 224, # 图片宽度

'\_R\_MEAN': 123.68, # CIFAR10的均值

'\_G\_MEAN': 116.78,

'\_B\_MEAN': 103.94,

'\_R\_STD': 1, # 自定义的标准差

'\_G\_STD': 1,

'\_B\_STD':1,

'\_RESIZE\_SIDE\_MIN': 256, # 图像增强resize最小值

'\_RESIZE\_SIDE\_MAX': 512,

'batch\_size': 32, # 批次大小

'num\_class': 5, # 分类类别

'epoch\_size': 150, # 训练次数

'loss\_scale\_num':1024,

'prefix': 'resnet-ai', # 模型保存的名称

'directory': './model\_resnet', # 模型保存的路径

'save\_checkpoint\_steps': 10, # 每隔10步保存ckpt

})

### 读取数据集

# 数据处理

def read\_data(path,config,usage="train"):

# 从目录中读取图像的源数据集。

dataset = ds.ImageFolderDataset(path,

class\_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})

# define map operations

# 图像解码算子

decode\_op = vision.Decode()

# 图像正则化算子

normalize\_op = vision.Normalize(mean=[cfg.\_R\_MEAN, cfg.\_G\_MEAN, cfg.\_B\_MEAN], std=[cfg.\_R\_STD, cfg.\_G\_STD, cfg.\_B\_STD])

# 图像调整大小算子

resize\_op = vision.Resize(cfg.\_RESIZE\_SIDE\_MIN)

# 图像裁剪算子

center\_crop\_op = vision.CenterCrop((cfg.HEIGHT, cfg.WIDTH))

# 图像随机水平翻转算子

horizontal\_flip\_op = vision.RandomHorizontalFlip()

# 图像通道数转换算子

channelswap\_op = vision.HWC2CHW()

# 图像随机裁剪解码编码调整大小算子

random\_crop\_decode\_resize\_op = vision.RandomCropDecodeResize((cfg.HEIGHT, cfg.WIDTH), (0.5, 1.0), (1.0, 1.0), max\_attempts=100)

# 只对训练集做的预处理操作

if usage == 'train':

dataset = dataset.map(input\_columns="image", operations=random\_crop\_decode\_resize\_op)

dataset = dataset.map(input\_columns="image", operations=horizontal\_flip\_op)

# 只对测试集做的预处理操作

else:

dataset = dataset.map(input\_columns="image", operations=decode\_op)

dataset = dataset.map(input\_columns="image", operations=resize\_op)

dataset = dataset.map(input\_columns="image", operations=center\_crop\_op)

# 对全部数据集做的预处理操作

dataset = dataset.map(input\_columns="image", operations=normalize\_op)

dataset = dataset.map(input\_columns="image", operations=channelswap\_op)

# 对训练集做的批次处理

if usage == 'train':

dataset = dataset.shuffle(buffer\_size=10000) # 10000 as in imageNet train script

dataset = dataset.batch(cfg.batch\_size, drop\_remainder=True)

# 对测试集做的批次处理

else:

dataset = dataset.batch(1, drop\_remainder=True)

# 数据增强

dataset = dataset.repeat(1)

dataset.map\_model = 4

return dataset

# 查看训练集和测试集的数量

de\_train = read\_data(cfg.data\_path,cfg,usage="train")

de\_test = read\_data(cfg.test\_path,cfg,usage="test")

print('训练数据集数量：',de\_train.get\_dataset\_size()\*cfg.batch\_size) # get\_dataset\_size()获取批处理的大小。

print('测试数据集数量：',de\_test.get\_dataset\_size())

# 查看训练集的样图

data\_next = de\_train.create\_dict\_iterator(output\_numpy=True).\_\_next\_\_()

print('通道数/图像长/宽：', data\_next['image'][0,...].shape)

print('一张图像的标签样式：', data\_next['label'][0]) # 一共5类，用0-4的数字表达类别。

plt.figure()

plt.imshow(data\_next['image'][0,0,...])

plt.colorbar()

plt.grid(False)

plt.show()

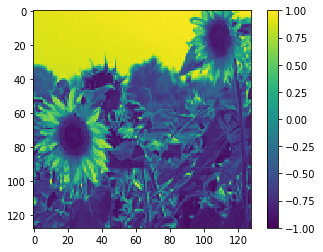
输出：

训练数据集数量： 3616

测试数据集数量： 32

通道数/图像长/宽： (3, 224, 224)

一张图像的标签样式： 3



### 模型构建训练

定义模型

自定义动态学习率

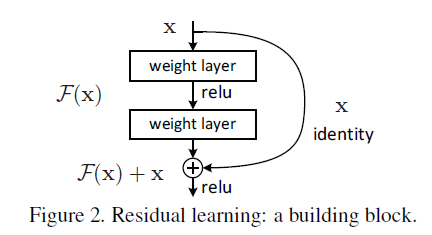
模型训练

模型评估

定义模型

残差块（Residual Block）

将前面若干层的输出跳过中间层作为后几层的输入部分，也就是说后面特征层会有前几层的部分线性贡献。此种设计是为了克服随着网络层数加深而产生的学习效率变低和准确率无法有效提升的问题。

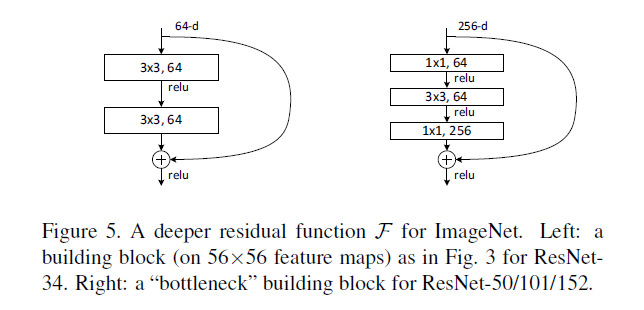


如果维度相同：

如果维度不同，则采用：

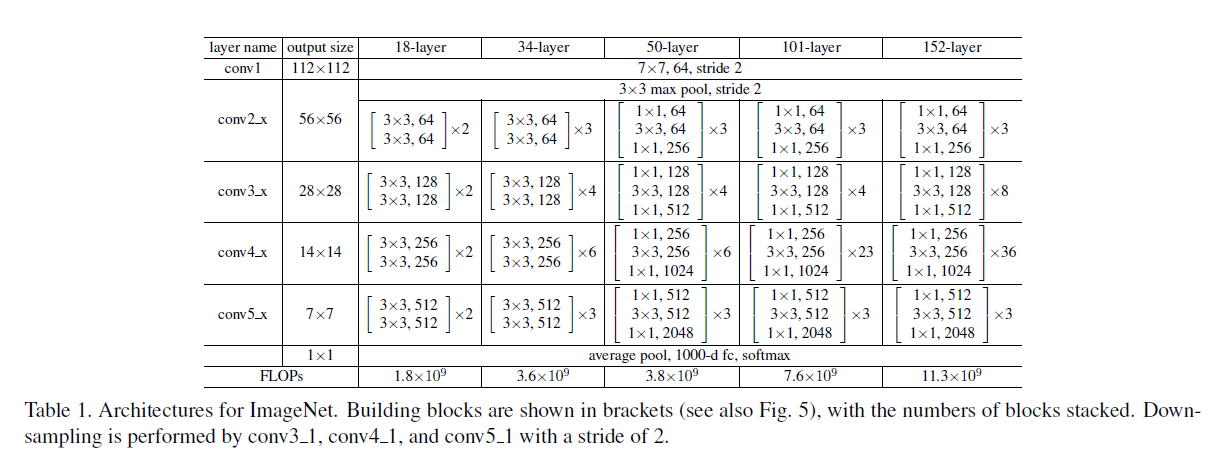
瓶颈（BottleNeck）模块：

瓶颈（BottleNeck）模块，思路和Inception一样，通过1x1 conv来巧妙地缩减或扩张feature map维度从而使3x3 conv的filters数目不受外界即上一层输入的影响，自然它的输出也不会影响到下一层module。



ResNet50模型构建

ResNet50有两个基本的块，分别名为Conv Block和Identity Block，其中Conv Block输入和输出的维度是不一样的，所以不能连续串联，它的作用是改变网络的维度；Identity Block输入维度和输出维度相同，可以串联，用于加深网络。



定义模型

"""ResNet."""

# 定义权重初始化函数

def \_weight\_variable(shape, factor=0.01):

init\_value = np.random.randn(\*shape).astype(np.float32) \* factor

return Tensor(init\_value)

# 定义3X3卷积函数

def \_conv3x3(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 3, 3)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=3, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 定义1X1卷积层函数

def \_conv1x1(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 1, 1)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=1, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 定义7X7卷积层函数

def \_conv7x7(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 7, 7)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=7, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 定义Batch Norm层函数

def \_bn(channel):

return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,

gamma\_init=1, beta\_init=0, moving\_mean\_init=0, moving\_var\_init=1)

# 定义最后一层的Batch Norm函数

def \_bn\_last(channel):

return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,

gamma\_init=0, beta\_init=0, moving\_mean\_init=0, moving\_var\_init=1)

# 定义全连接层函数

def \_fc(in\_channel, out\_channel):

weight\_shape = (out\_channel, in\_channel)

weight = \_weight\_variable(weight\_shape)

return nn.Dense(in\_channel, out\_channel, has\_bias=True, weight\_init=weight, bias\_init=0)

# 构建残差模块

class ResidualBlock(nn.Cell):

"""

ResNet V1 residual block definition.

Args:

in\_channel (int): Input channel.

out\_channel (int): Output channel.

stride (int): Stride size for the first convolutional layer. Default: 1.

Returns:

Tensor, output tensor.

Examples:

>>> ResidualBlock(3, 256, stride=2)

"""

expansion = 4 # conv2\_x--conv5\_x中，前两层的卷积核的个数是第三层（也就是输出通道）的4分之一。

def \_\_init\_\_(self, in\_channel, out\_channel, stride=1):

super(ResidualBlock, self).\_\_init\_\_()

# 前两层的卷积核个数等于输出通道的四分之一

channel = out\_channel // self.expansion

# 第一层卷积

self.conv1 = \_conv1x1(in\_channel, channel, stride=1)

self.bn1 = \_bn(channel)

# 第二层卷积

self.conv2 = \_conv3x3(channel, channel, stride=stride)

self.bn2 = \_bn(channel)

# 第三层卷积，其中卷积核个数等于输出通道

self.conv3 = \_conv1x1(channel, out\_channel, stride=1)

self.bn3 = \_bn\_last(out\_channel)

# Relu激活层

self.relu = nn.ReLU()

self.down\_sample = False

# 当步长不为1、或输出通道不等于输入通道时，进行图像下采样，用来调整通道数

if stride != 1 or in\_channel != out\_channel:

self.down\_sample = True

self.down\_sample\_layer = None

# 用1X1卷积调整通道数

if self.down\_sample:

self.down\_sample\_layer = nn.SequentialCell([\_conv1x1(in\_channel, out\_channel, stride), # 1X1卷积

\_bn(out\_channel)]) # Batch Norm

# 加法算子

self.add = ops.Add()

# 构建残差块

def construct(self, x):

# 输入

identity = x

# 第一层卷积 1X1

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

# 第二层卷积 3X3

out = self.conv2(out)

out = self.bn2(out)

out = self.relu(out)

# 第三层卷积 1X1

out = self.conv3(out)

out = self.bn3(out)

# 改变网络的维度

if self.down\_sample:

identity = self.down\_sample\_layer(identity)

# 加上残差

out = self.add(out, identity)

# Relu激活

out = self.relu(out)

return out

# 构建残差网络

class ResNet(nn.Cell):

"""

ResNet architecture.

Args:

block (Cell): Block for network.

layer\_nums (list): Numbers of block in different layers.

in\_channels (list): Input channel in each layer.

out\_channels (list): Output channel in each layer.

strides (list): Stride size in each layer.

num\_classes (int): The number of classes that the training images are belonging to.

Returns:

Tensor, output tensor.

Examples:

>>> ResNet(ResidualBlock,

>>> [3, 4, 6, 3],

>>> [64, 256, 512, 1024],

>>> [256, 512, 1024, 2048],

>>> [1, 2, 2, 2],

>>> 10)

"""

# 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长，图像类别数

def \_\_init\_\_(self, block, layer\_nums, in\_channels, out\_channels, strides, num\_classes):

super(ResNet, self).\_\_init\_\_()

# 报错信息，不用管

if not len(layer\_nums) == len(in\_channels) == len(out\_channels) == 4:

raise ValueError("the length of layer\_num, in\_channels, out\_channels list must be 4!")

# 第一层卷积，卷积核7X7，输入通道3，输出通道64，步长2

self.conv1 = \_conv7x7(3, 64, stride=2)

self.bn1 = \_bn(64)

self.relu = ops.ReLU()

# 3X3池化层，步长2

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode="same")

# conv2\_x残差块

self.layer1 = self.\_make\_layer(block,

layer\_nums[0],

in\_channel=in\_channels[0],

out\_channel=out\_channels[0],

stride=strides[0])

# conv3\_x残差块

self.layer2 = self.\_make\_layer(block,

layer\_nums[1],

in\_channel=in\_channels[1],

out\_channel=out\_channels[1],

stride=strides[1])

# conv4\_x残差块

self.layer3 = self.\_make\_layer(block,

layer\_nums[2],

in\_channel=in\_channels[2],

out\_channel=out\_channels[2],

stride=strides[2])

# conv5\_x残差块

self.layer4 = self.\_make\_layer(block,

layer\_nums[3],

in\_channel=in\_channels[3],

out\_channel=out\_channels[3],

stride=strides[3])

# 均值算子

self.mean = ops.ReduceMean(keep\_dims=True)

# Flatten层

self.flatten = nn.Flatten()

# 输出层

self.end\_point = \_fc(out\_channels[3], num\_classes)

# 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长

def \_make\_layer(self, block, layer\_num, in\_channel, out\_channel, stride):

"""

Make stage network of ResNet.

Args:

block (Cell): Resnet block.

layer\_num (int): Layer number.

in\_channel (int): Input channel.

out\_channel (int): Output channel.

stride (int): Stride size for the first convolutional layer.

Returns:

SequentialCell, the output layer.

Examples:

>>> \_make\_layer(ResidualBlock, 3, 128, 256, 2)

"""

# 搭建convn\_x的残差块

layers = []

resnet\_block = block(in\_channel, out\_channel, stride=stride)

layers.append(resnet\_block)

for \_ in range(1, layer\_num):

resnet\_block = block(out\_channel, out\_channel, stride=1)

layers.append(resnet\_block)

return nn.SequentialCell(layers)

# 构建 ResNet网络

def construct(self, x):

x = self.conv1(x) # 第一层卷积7X7，步长为2

x = self.bn1(x) # 第一层的Batch Norm

x = self.relu(x) # Rule激活层

c1 = self.maxpool(x) # 最大池化3X3，步长为2

c2 = self.layer1(c1) # conv2\_x残差块

c3 = self.layer2(c2) # conv3\_x残差块

c4 = self.layer3(c3) # conv4\_x残差块

c5 = self.layer4(c4) # conv5\_x残差块

out = self.mean(c5, (2, 3)) # 平均池化层

out = self.flatten(out) # Flatten层

out = self.end\_point(out) # 输出层

return out

# 构建ResNet50 网络

def resnet50(class\_num=10):

"""

Get ResNet50 neural network.

Args:

class\_num (int): Class number.

Returns:

Cell, cell instance of ResNet50 neural network.

Examples:

>>> net = resnet50(10)

"""

return ResNet(ResidualBlock, # 残差块

[3, 4, 6, 3], # 残差块数量

[64, 256, 512, 1024], # 输入通道

[256, 512, 1024, 2048], # 输出通道

[1, 2, 2, 2], # 步长

class\_num) # 输出类别数

# 构建ResNet101 网络

def resnet101(class\_num=1001):

"""

Get ResNet101 neural network.

Args:

class\_num (int): Class number.

Returns:

Cell, cell instance of ResNet101 neural network.

Examples:

>>> net = resnet101(1001)

"""

return ResNet(ResidualBlock,

[3, 4, 23, 3],

[64, 256, 512, 1024],

[256, 512, 1024, 2048],

[1, 2, 2, 2],

class\_num)

定义学习率函数

def get\_lr(global\_step,

total\_epochs,

steps\_per\_epoch,

lr\_init=0.01,

lr\_max=0.1,

warmup\_epochs=5):

"""

Generate learning rate array.

Args:

global\_step (int): Initial step of training.

total\_epochs (int): Total epoch of training.

steps\_per\_epoch (float): Steps of one epoch.

lr\_init (float): Initial learning rate. Default: 0.01.

lr\_max (float): Maximum learning rate. Default: 0.1.

warmup\_epochs (int): The number of warming up epochs. Default: 5.

Returns:

np.array, learning rate array.

"""

lr\_each\_step = []

total\_steps = steps\_per\_epoch \* total\_epochs

warmup\_steps = steps\_per\_epoch \* warmup\_epochs

if warmup\_steps != 0:

inc\_each\_step = (float(lr\_max) - float(lr\_init)) / float(warmup\_steps)

else:

inc\_each\_step = 0

for i in range(int(total\_steps)):

if i < warmup\_steps:

lr = float(lr\_init) + inc\_each\_step \* float(i)

else:

base = ( 1.0 - (float(i) - float(warmup\_steps)) / (float(total\_steps) - float(warmup\_steps)) )

lr = float(lr\_max) \* base \* base

if lr < 0.0:

lr = 0.0

lr\_each\_step.append(lr)

current\_step = global\_step

lr\_each\_step = np.array(lr\_each\_step).astype(np.float32)

learning\_rate = lr\_each\_step[current\_step:]

return learning\_rate

开始训练

完成数据预处理、网络定义、损失函数和优化器定义之后，开始模型训练。模型训练包含2层迭代，数据集的多伦迭代epoch和一轮数据集内按分组从数据集中抽取数据，输入网络计算得到损失函数，然后通过优化器计算和更新训练参数的梯度。

# 构建ResNet50网络，输出类别数为5，对应5种花的类别

net=resnet50(class\_num=cfg.num\_class)

# 设置Softmax交叉熵损失函数

loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

# 设置动态学习率

train\_step\_size = de\_train.get\_dataset\_size()

lr = Tensor(get\_lr(global\_step=0, total\_epochs=cfg.epoch\_size, steps\_per\_epoch=train\_step\_size))

# 设置动量优化器

opt = Momentum(net.trainable\_params(), lr, momentum=0.9, weight\_decay=1e-4, loss\_scale=cfg.loss\_scale\_num)

# 损失值平滑，解决训练过程中梯度过小的问题

loss\_scale = FixedLossScaleManager(cfg.loss\_scale\_num, False)

# 模型编译，输入网络结构，损失函数，优化器，损失值平滑，以及模型评估标准

model = Model(net, loss\_fn=loss, optimizer=opt, loss\_scale\_manager=loss\_scale, metrics={'acc'})

# 损失值监控

loss\_cb = LossMonitor(per\_print\_times=train\_step\_size)

# 模型保存参数，设置每隔多少步保存一次模型，最多保存几个模型

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps, keep\_checkpoint\_max=1)

# 模型保存，设置模型保存的名称，路径，以及保存参数

ckpoint\_cb = ModelCheckpoint(prefix=cfg.prefix, directory=cfg.directory, config=ckpt\_config)

print("============== Starting Training ==============")

# 训练模型，设置训练次数，训练集，回调函数，是否采用数据下沉模式（只可应用于Ascend 和GPU，可加快训练速度）

model.train(cfg.epoch\_size, de\_train, callbacks=[loss\_cb,ckpoint\_cb], dataset\_sink\_mode=True)

# 训练时长大约15-20分钟

# 使用测试集进行模型评估，输出测试集的准确率

metric = model.eval(de\_test)

print(metric)

输出：

============== Starting Training ==============

epoch: 1 step: 113, loss is 2.1889174

epoch: 2 step: 113, loss is 2.1905

epoch: 3 step: 113, loss is 1.1549394

epoch: 4 step: 113, loss is 1.2237203

epoch: 5 step: 113, loss is 0.8866976

epoch: 6 step: 113, loss is 0.8851016

epoch: 7 step: 113, loss is 0.84188724

epoch: 8 step: 113, loss is 0.6292048

epoch: 9 step: 113, loss is 0.98489845

epoch: 10 step: 113, loss is 0.6488789

...

epoch: 146 step: 113, loss is 0.010150566

epoch: 147 step: 113, loss is 0.006538092

epoch: 148 step: 113, loss is 0.03851804

epoch: 149 step: 113, loss is 0.023275778

epoch: 150 step: 113, loss is 0.005423387

{'acc': 0.9038461538461539}

### 模型预测

# 模型预测，从测试集中取10个样本进行测试，输出预测结果和真实标签

class\_names = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}

for i in range(10):

test\_ = de\_test.create\_dict\_iterator().\_\_next\_\_()

test = Tensor(test\_['image'], mindspore.float32)

# 模型预测

predictions = model.predict(test)

predictions = predictions.asnumpy()

true\_label = test\_['label'].asnumpy()

# 显示预测结果

p\_np = predictions[0, :]

pre\_label = np.argmax(p\_np)

print('第' + str(i) + '个sample预测结果：', class\_names[pre\_label], ' 真实结果：', class\_names[true\_label[0]])

输出：

第0个sample预测结果： dandelion 真实结果： dandelion

第1个sample预测结果： sunflowers 真实结果： sunflowers

第2个sample预测结果： daisy 真实结果： dandelion

第3个sample预测结果： daisy 真实结果： daisy

第4个sample预测结果： daisy 真实结果： daisy

第5个sample预测结果： sunflowers 真实结果： sunflowers

第6个sample预测结果： tulips 真实结果： tulips

第7个sample预测结果： daisy 真实结果： daisy

第8个sample预测结果： roses 真实结果： roses

第9个sample预测结果： tulips 真实结果： daisy

### 模型保存

保存模型为onnx格式

# 创建flower文件夹，将模型保存至此文件夹下

if not os.path.exists('./flowers/'):

os.mkdir('./flowers/')

# 加载ckpt模型，注意如果此行报错，可将cfg.directory,后的代码改为已存在的ckpt文件，例如'resnet-ai\_2-150\_113.ckpt'

param\_dict = load\_checkpoint(os.path.join(cfg.directory,cfg.prefix+'-'+str(cfg.epoch\_size)+'\_'+str(train\_step\_size)+'.ckpt'))

# 设置ResNet50网络

resnet=resnet50(class\_num=cfg.num\_class)

# 加载模型参数到ResNet50网络内

load\_param\_into\_net(resnet, param\_dict)

# 这里 x 用来指定导出模型的输入shape以及数据类型

x = np.random.uniform(-1.0, 1.0, size = [1, 3, cfg.HEIGHT, cfg.WIDTH]).astype(np.float32)

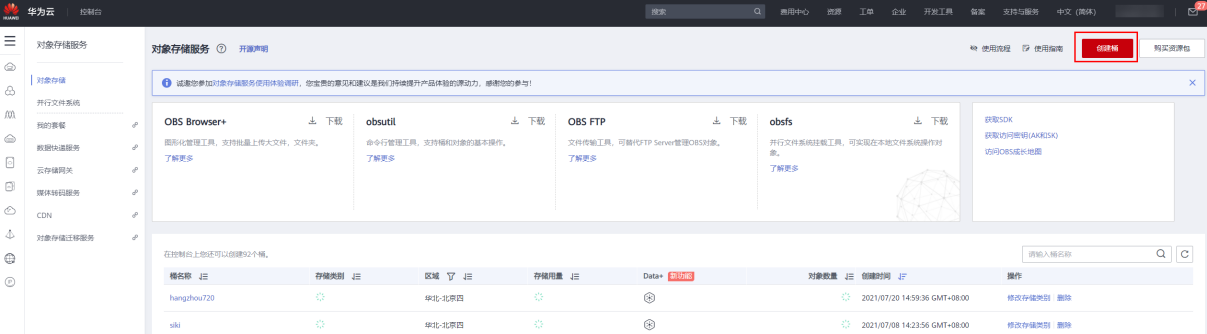
# 导出ONNX模型，设置网络，网络的输入，模型名称，保存格式

export(resnet, Tensor(x), file\_name = './flowers/best\_model.onnx', file\_format = 'ONNX')

保存模型到obs桶里面

在下方链接处创建OBS桶，区域选择华北-北京四，自定义桶名称：

[https://console.huaweicloud.com/console/?region=cn-north-4#/obs/create](https://console.huaweicloud.com/console/?region=cn-north-4" \l "/obs/create)



创建桶



自定义桶名称

OBS桶创建完成之后，在ModelArts的Notebook运行以下代码，将onnx模型保存至自己创建的OBS桶内。可以通过如下链接查看模型是否保存成功：

[https://console.huaweicloud.com/console/?region=cn-north-4#/obs/manager/buckets](https://console.huaweicloud.com/console/?region=cn-north-4" \l "/obs/manager/buckets)

#创建桶，修改成自定义的桶名

import moxing

moxing.file.copy\_parallel(src\_url='./flowers/best\_model.onnx', dst\_url='obs://guidian23/flower/onnx/best\_model.onnx') # 将guidian23修改成自己的桶名称

配置文件insert\_op\_conf.cfg和customize\_service.py的下载链接：<https://zhuanyejianshe.obs.cn-north-4.myhuaweicloud.com/chuangxinshijianke/cv-nlp/flower_recognition.zip>

## 实验总结

本章提供了一个基于华为MindSpore框架的花卉图像识别实验。该实验演示了如何利用华为云ModelArts完成图像识别任务。通过MindSpore来构建图像识别模型。

# 前沿网络案例- DeepLabv3

## 实验介绍

语义分割是像素级的物体识别。目标是用对应的类（class）标记图像中的每个像素。因为我们要预测图像中的每个像素，所以此任务通常被称为密集预测。语义分割有着广泛的应用场景，例如自动驾驶，人机交互，医学图像诊断，计算摄影学和增强现实等。

本实验使用MindSpore深度学习框架构建DeepLabv3网络模型在PASCAL VOC2012数据集上进行图像语义分割。图像的语义分割是将输入图像中的每个像素分配一个语义类别，以得到像素级的密集分类。

## 实验环境要求

ModelArts平台：Mindspore

若选择在ModelArts平台快速搭建，可参考文末附录：ModelArts开发环境搭建。

## 背景知识

### 网络介绍

语义分割网络介绍

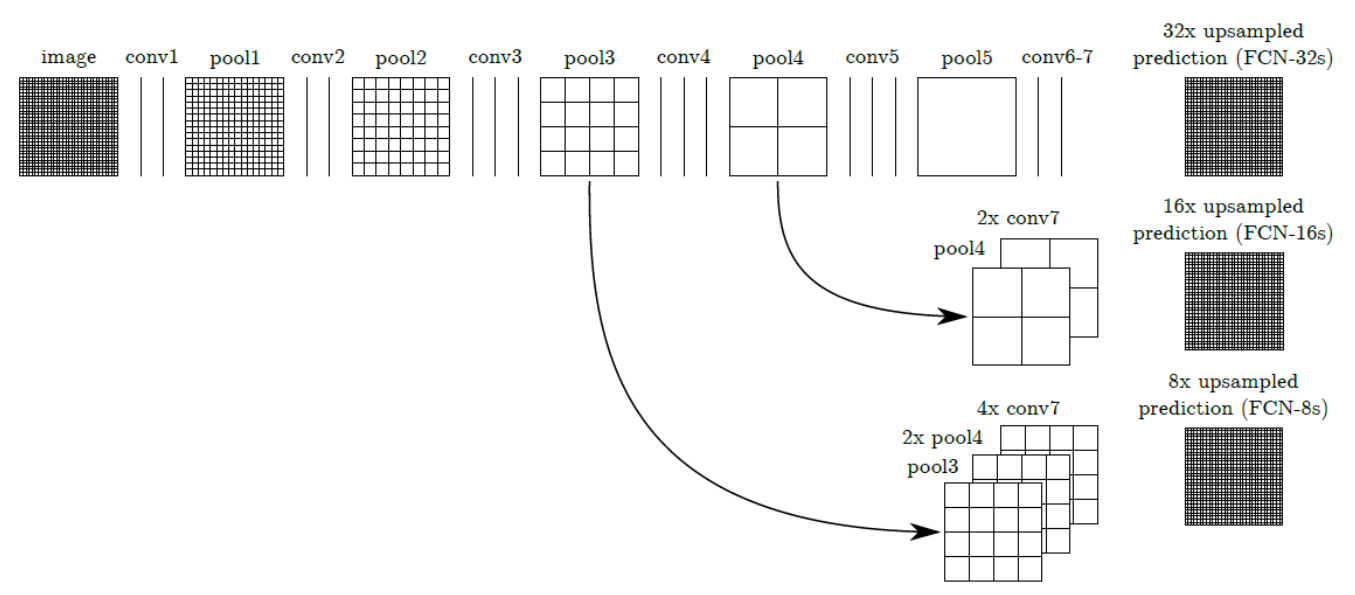
一般的语义分割架构可以被认为是一个编码器-解码器网络。编码器通常是一个预训练的分类网络，像 VGG、ResNet，然后是一个解码器网络。这些架构不同的地方主要在于解码器网络。解码器的任务是将编码器学习到的可判别特征（较低分辨率）从语义上投影到像素空间（较高分辨率），以获得密集分类。语义分割不仅需要在像素级有判别能力，还需要有能将编码器在不同阶段学到的可判别特征投影到像素空间的机制。不同的架构采用不同的机制（跳跃连接、金字塔池化等）作为解码机制的一部分。

FCN网络介绍

自2014年 [Long等人](https://arxiv.org/abs/1411.4038)首次使用全卷积神经网络 (FCN)对自然图像进行端到端分割，语义分割才产生了大的突破。FCN网络将当前分类网络（AlexNet, VGG net等）修改为全卷积网络，通过对分割任务进行微调，将它们学习的表征转移到网络中。然后，定义了一种新的架构，将深的、粗糙的网络层的语义信息和浅的、精细的网络层的表层信息结合起来，来生成精确和详细的分割。FCN网络结构：FCN网络结构图如下所示。网络为卷积层和池化层的堆叠，特征图大小不断地减小，最后通过上采样的方式恢复图像分辨率。网络分为FCN-32s、FCN-16s、FCN-8s，分别代表上采样32、16、8倍的网络。以FCN-8s为例，将pool3输出的特征图上采样8倍、pool4输出的特征图上采样16倍、conv7的特征图上采样32倍。最后将三者的结果拼接在一起，以使用多个尺度的特征图。综合浅层特征的位置信息和深层特征的语义信息，这样在分割时就有足够的上下文信息(context information)，同时也有目标的细节信息。

FCN问题：在CNN中，低层的卷积中空间位置信息较为准确，但是感受野较小，只能关注附近像素点的信息。高层的卷积中感受野扩大，可以获取全局信息，但是目标空间位置信息损失严重，不能精确定位目标。而图像分割相较于分类任务需要更多的上下文信息和更精确的目标定位信息。

论文：<https://arxiv.org/abs/1411.4038>



DeepLabv1网络介绍

近来，深度卷积网络（DCNN）在高级视觉任务（图像分类和目标检测）中展示了优异的性能。DeepLabv1结合DCNN 和概率图模型来解决像素级分类任务（即语义分割）。其关键特点：

1. 提出 空洞卷积（atrous convolution）。
2. 在最后两个最大池化操作中不降低特征图的分辨率，并在倒数第二个最大池化之后的卷积中使用空洞卷积。
3. 使用 CRF（条件随机场） 作为后处理，恢复边界细节，达到准确定位效果。
4. 附加输入图像和前四个最大池化层的每个输出到一个两层卷积，然后拼接到主网络的最后一层，达到 多尺度预测效果。

DeepLabv2关键特点：

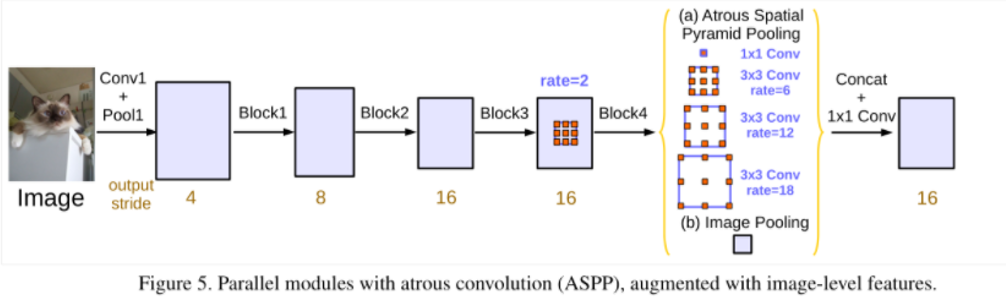
1. 强调上采样过滤器的卷积，或“空洞卷积”，在密集预测任务中是一个强大的工具。空洞卷积允许显式地控制在深度卷积神经网络中计算的特征响应的分辨率。它还允许有效地扩大过滤器的视野，在不增加参数数量或计算量的情况下引入更大的上下文。
2. 提出了一种空洞空间金字塔池化（ASPP）的多尺度鲁棒分割方法。ASPP 使用多个采样率的过滤器和有效的视野探测传入的卷积特征层，从而在多个尺度上捕获目标和图像上下文。
3. 结合 DCNNs 方法和概率图形模型，改进了目标边界的定位。DCNNs 中常用的最大池化和下采样的组合实现了不变性，但对定位精度有一定的影响。通过将 DCNN 最后一层的响应与一个全连接条件随机场(CRF)相结合来克服这个问题。

DeepLabv3网络介绍

相较于之前的DeepLab有很大的改进，在PASCAL VOC2012图像语义分割基准上获得了state-of-art的性能（论文参考：<http://arxiv.org/abs/1706.05587>）。

其关键特点：

1. 为了解决多尺度目标的分割问题，串行/并行设计了能够捕捉多尺度上下文的模块，模块中采用不同的空洞率。
2. 增强了先前提出的空洞空间金字塔池化（ASPP）模块，增加了图像级特征来编码全局上下文，使得模块可以在多尺度下探测卷积特征。并在没有 CRF 作为后处理的情况下显著提升了性能。



DeepLabv3 使用 ResNet 作为主干网络。

## 实验步骤

本实验采用VOCaug数据集训练得到的DeepLabv3 Checkpoint，使用VOC2012训练数据集对模型进行微调训练，最后在VOC2012测试数据对模型进行测试。

准备数据集

[VOC2012](http://host.robots.ox.ac.uk:8080/pascal/VOC/voc2012/)（Visual Object Classes Challenge 2012）数据集来自2012年的PASCAL VOC大赛。PASCAL VOC 大赛是一项世界级的计算机视觉挑战赛，该挑战赛由 Mark Everingham、Luc Van Gool、Chris Williams、John Winn 和 Andrew Zisserman 发起，并在 2005 至 2012 年期间举办，比赛项目包括图像分类（Object Classification）、目标检测（Object Detection）、目标分割（Object Segmentation）、人体关节点识别（Human Layout）、动作识别（Action Classification）几大类。

PASCAL VOC2012数据集是针对计算机视觉任务中监督学习的数据集，它主要有四个大类别，分别是人、常见动物、交通车辆、室内家具用品，其中又细分20 个小类（加背景 21 类）。

详细类别如下所示：

Person: person

Animal: bird, cat, cow, dog, horse, sheep

Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train

Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

VOC原始数据集包含以下目录结构：

VOC

├── Annotations # 目标检测任务标签，xml 形式，文件名与图片名对应

├── ImageSets # 存放不同任务训练和测试数据的编号，可根据编号在JPEGImages文件中找到参与训练和测试的数据图片。（有些任务编号直接带标签）

| ├── Action # 行为识别（包含数据编号和标签）,格式.txt

| ├── Layout # 人体部位识别（包含数据编号和标签）,格式.txt

| ├── Main # 分类（包含数据编号和标签）,格式.txt

| └──Segmentation # 语义分割（包含数据编号，无对应标签，标签参考文件SegmentationClass）。

| ├── trainval.txt

| ├── val.txt

| └── train.txt

├── JPEGImages # 数据集所有原图（彩色三通道），格式.jpg

├── SegmentationClass # 语义分割标签图（彩色三通道）,格式.png

└── SegmentationObject # 实例分割标签图（彩色三通道）,格式.png

这里使用ModelArts的Moxing将储存在OBS的VOC2012数据集下载到Notebook环境中。

import moxing as mox

mox.file.copy\_parallel(src\_url="obs://ascend-professional-construction-dataset/deep-learning/deeplabv3-mindspore/VOC2012", dst\_url="./VOC2012")

准备预训练模型

为了节省训练时间，本实验采用fine-tune的训练方式，将预训练的checkpoint文件（deeplab\_v3\_s8-300\_11.ckpt）使用ModelArts的Moxing拷贝至执行容器内。

该模型为VOCaug数据集训练得到。VOCaug数据集是VOC2012数据集和SBD数据集的集合。SBD数据属于VOC2012数据集，但是VOC2012数据集的训练或者验证图片的标签图非常少。但是SBD给出的很多，所以可以综合这两个数据集得到更加多的带标签数据。

|  |  |  |
| --- | --- | --- |
| **数据集名称** | **训练** | **测试** |
| VOC2012数据集 | 1464 | 1449 |
| SBD数据集 | 8498 | 2857 |
| VOCaug数据集 | 8829\* | \ |

**\***VOCaug数据集8829样例个数已经去重。

使用ModelArts的Moxing将储存在OBS的预训练的DeepLabv3模型deeplab\_v3\_s8-300\_11.ckpt下载到Notebook环境中。

mox.file.copy\_parallel(src\_url="obs://ascend-professional-construction-dataset/deep-learning/deeplabv3-mindspore/ckpt", dst\_url="./ckpt")

构建数据读取相关函数

以下代码主要构建一个数据集分割类，通过这个类可以完成生成Mindrecord文件、预处理数据集等操作。

import os

import numpy as np

import scipy.io

import pickle

from PIL import Image

import shutil

import cv2

from mindspore.mindrecord import FileWriter

import mindspore.dataset as de

cv2.setNumThreads(0)

class SegDataset:

def \_\_init\_\_(self,

image\_mean,

image\_std,

data\_file='',

batch\_size=32,

crop\_size=512,

max\_scale=2.0,

min\_scale=0.5,

ignore\_label=255,

num\_classes=21,

num\_readers=2,

num\_parallel\_calls=4,

shard\_id=None,

shard\_num=None):

self.data\_file = data\_file

self.batch\_size = batch\_size

self.crop\_size = crop\_size

self.image\_mean = np.array(image\_mean, dtype=np.float32)

self.image\_std = np.array(image\_std, dtype=np.float32)

self.max\_scale = max\_scale

self.min\_scale = min\_scale

self.ignore\_label = ignore\_label

self.num\_classes = num\_classes

self.num\_readers = num\_readers

self.num\_parallel\_calls = num\_parallel\_calls

self.shard\_id = shard\_id

self.shard\_num = shard\_num

self.voc\_img\_dir = os.path.join(self.data\_file,'JPEGImages')

self.voc\_anno\_dir = os.path.join(self.data\_file,'SegmentationClass')

self.voc\_train\_lst = os.path.join(self.data\_file,'ImageSets/Segmentation/train.txt')

self.voc\_val\_lst = os.path.join(self.data\_file,'ImageSets/Segmentation/val.txt')

self.voc\_anno\_gray\_dir = os.path.join(self.data\_file,'SegmentationClassGray')

self.mindrecord\_save = os.path.join(self.data\_file,'VOC\_mindrecord')

assert max\_scale > min\_scale

def preprocess\_(self, image, label):

# bgr image

image\_out = cv2.imdecode(np.frombuffer(image, dtype=np.uint8), cv2.IMREAD\_COLOR)

label\_out = cv2.imdecode(np.frombuffer(label, dtype=np.uint8), cv2.IMREAD\_GRAYSCALE)

sc = np.random.uniform(self.min\_scale, self.max\_scale)

new\_h, new\_w = int(sc \* image\_out.shape[0]), int(sc \* image\_out.shape[1])

image\_out = cv2.resize(image\_out, (new\_w, new\_h), interpolation=cv2.INTER\_CUBIC)

label\_out = cv2.resize(label\_out, (new\_w, new\_h), interpolation=cv2.INTER\_NEAREST)

image\_out = (image\_out - self.image\_mean) / self.image\_std

h\_, w\_ = max(new\_h, self.crop\_size), max(new\_w, self.crop\_size)

pad\_h, pad\_w = h\_ - new\_h, w\_ - new\_w

if pad\_h > 0 or pad\_w > 0:

image\_out = cv2.copyMakeBorder(image\_out, 0, pad\_h, 0, pad\_w, cv2.BORDER\_CONSTANT, value=0)

label\_out = cv2.copyMakeBorder(label\_out, 0, pad\_h, 0, pad\_w, cv2.BORDER\_CONSTANT, value=self.ignore\_label)

offset\_h = np.random.randint(0, h\_ - self.crop\_size + 1)

offset\_w = np.random.randint(0, w\_ - self.crop\_size + 1)

image\_out = image\_out[offset\_h: offset\_h + self.crop\_size, offset\_w: offset\_w + self.crop\_size, :]

label\_out = label\_out[offset\_h: offset\_h + self.crop\_size, offset\_w: offset\_w+self.crop\_size]

if np.random.uniform(0.0, 1.0) > 0.5:

image\_out = image\_out[:, ::-1, :]

label\_out = label\_out[:, ::-1]

image\_out = image\_out.transpose((2, 0, 1))

image\_out = image\_out.copy()

label\_out = label\_out.copy()

return image\_out, label\_out

def get\_gray\_dataset(self):

if os.path.exists(self.voc\_anno\_gray\_dir):

print('the gray file is already exists！')

return

os.makedirs(self.voc\_anno\_gray\_dir)

# convert voc color png to gray png

print('converting voc color png to gray png ...')

for ann in os.listdir(self.voc\_anno\_dir):

ann\_im = Image.open(os.path.join(self.voc\_anno\_dir, ann))

ann\_im = Image.fromarray(np.array(ann\_im))

ann\_im.save(os.path.join(self.voc\_anno\_gray\_dir, ann))

print('converting done')

def get\_mindrecord\_dataset(self, is\_training,num\_shards=1, shuffle=True):

datas = []

if is\_training:

data\_lst = self.voc\_train\_lst

self.mindrecord\_save = os.path.join(self.mindrecord\_save,'train')

else:

data\_lst = self.voc\_val\_lst

self.mindrecord\_save = os.path.join(self.mindrecord\_save,'eval')

if os.path.exists(self.mindrecord\_save):

#shutil.rmtree(self.mindrecord\_save)

print('mindrecord file is already exists！')

self.mindrecord\_save = os.path.join(self.mindrecord\_save,'VOC\_mindrecord')

return

with open(data\_lst) as f:

lines = f.readlines()

if shuffle:

np.random.shuffle(lines)

print('creating mindrecord dataset...')

os.makedirs(self.mindrecord\_save)

self.mindrecord\_save = os.path.join(self.mindrecord\_save,'VOC\_mindrecord')

print('number of samples:', len(lines))

seg\_schema = {"file\_name": {"type": "string"}, "label": {"type": "bytes"}, "data": {"type": "bytes"}}

writer = FileWriter(file\_name=self.mindrecord\_save, shard\_num=num\_shards)

writer.add\_schema(seg\_schema, "seg\_schema")

cnt = 0

for l in lines:

id\_ = l.strip()

img\_path = os.path.join(self.voc\_img\_dir, id\_ + '.jpg')

label\_path = os.path.join(self.voc\_anno\_gray\_dir, id\_ + '.png')

sample\_ = {"file\_name": img\_path.split('/')[-1]}

with open(img\_path, 'rb') as f:

sample\_['data'] = f.read()

with open(label\_path, 'rb') as f:

sample\_['label'] = f.read()

datas.append(sample\_)

cnt += 1

if cnt % 1000 == 0:

writer.write\_raw\_data(datas)

print('number of samples written:', cnt)

datas = []

if datas:

writer.write\_raw\_data(datas)

writer.commit()

print('number of samples written:', cnt)

print('Create Mindrecord Done')

def get\_dataset(self, repeat=1):

data\_set = de.MindDataset(dataset\_files=self.mindrecord\_save, columns\_list=["data", "label"],

shuffle=True, num\_parallel\_workers=self.num\_readers,

num\_shards=self.shard\_num, shard\_id=self.shard\_id)

transforms\_list = self.preprocess\_

data\_set = data\_set.map(operations=transforms\_list, input\_columns=["data", "label"],

output\_columns=["data", "label"],

num\_parallel\_workers=self.num\_parallel\_calls)

data\_set = data\_set.shuffle(buffer\_size=self.batch\_size \* 10)

data\_set = data\_set.batch(self.batch\_size, drop\_remainder=True)

data\_set = data\_set.repeat(repeat)

return data\_set

构建网络

以下代码主要完成DeepLabv3主体网络的构建，通过定义多个resnet cell结构组成整体的DeepLabv3网络。最终的网络是以类的方式进行定义，通过实例化即可创建对应的网络对象。

import mindspore.nn as nn

from mindspore.ops import operations as P

def conv1x1(in\_planes, out\_planes, stride=1):

return nn.Conv2d(in\_planes, out\_planes, kernel\_size=1, stride=stride, weight\_init='xavier\_uniform')

def conv3x3(in\_planes, out\_planes, stride=1, dilation=1, padding=1):

return nn.Conv2d(in\_planes, out\_planes, kernel\_size=3, stride=stride, pad\_mode='pad', padding=padding,

dilation=dilation, weight\_init='xavier\_uniform')

class Resnet(nn.Cell):

def \_\_init\_\_(self, block, block\_num, output\_stride, use\_batch\_statistics=True):

super(Resnet, self).\_\_init\_\_()

self.inplanes = 64

self.conv1 = nn.Conv2d(3, self.inplanes, kernel\_size=7, stride=2, pad\_mode='pad', padding=3,

weight\_init='xavier\_uniform')

self.bn1 = nn.BatchNorm2d(self.inplanes, use\_batch\_statistics=use\_batch\_statistics)

self.relu = nn.ReLU()

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode='same')

self.layer1 = self.\_make\_layer(block, 64, block\_num[0], use\_batch\_statistics=use\_batch\_statistics)

self.layer2 = self.\_make\_layer(block, 128, block\_num[1], stride=2, use\_batch\_statistics=use\_batch\_statistics)

if output\_stride == 16:

self.layer3 = self.\_make\_layer(block, 256, block\_num[2], stride=2,

use\_batch\_statistics=use\_batch\_statistics)

self.layer4 = self.\_make\_layer(block, 512, block\_num[3], stride=1, base\_dilation=2, grids=[1, 2, 4],

use\_batch\_statistics=use\_batch\_statistics)

elif output\_stride == 8:

self.layer3 = self.\_make\_layer(block, 256, block\_num[2], stride=1, base\_dilation=2,

use\_batch\_statistics=use\_batch\_statistics)

self.layer4 = self.\_make\_layer(block, 512, block\_num[3], stride=1, base\_dilation=4, grids=[1, 2, 4],

use\_batch\_statistics=use\_batch\_statistics)

def \_make\_layer(self, block, planes, blocks, stride=1, base\_dilation=1, grids=None, use\_batch\_statistics=True):

if stride != 1 or self.inplanes != planes \* block.expansion:

downsample = nn.SequentialCell([

conv1x1(self.inplanes, planes \* block.expansion, stride),

nn.BatchNorm2d(planes \* block.expansion, use\_batch\_statistics=use\_batch\_statistics)

])

if grids is None:

grids = [1] \* blocks

layers = [

block(self.inplanes, planes, stride, downsample, dilation=base\_dilation \* grids[0],

use\_batch\_statistics=use\_batch\_statistics)

]

self.inplanes = planes \* block.expansion

for i in range(1, blocks):

layers.append(

block(self.inplanes, planes, dilation=base\_dilation \* grids[i],

use\_batch\_statistics=use\_batch\_statistics))

return nn.SequentialCell(layers)

def construct(self, x):

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

out = self.maxpool(out)

out = self.layer1(out)

out = self.layer2(out)

out = self.layer3(out)

out = self.layer4(out)

return out

class Bottleneck(nn.Cell):

expansion = 4

def \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None, dilation=1, use\_batch\_statistics=True):

super(Bottleneck, self).\_\_init\_\_()

self.conv1 = conv1x1(inplanes, planes)

self.bn1 = nn.BatchNorm2d(planes, use\_batch\_statistics=use\_batch\_statistics)

self.conv2 = conv3x3(planes, planes, stride, dilation, dilation)

self.bn2 = nn.BatchNorm2d(planes, use\_batch\_statistics=use\_batch\_statistics)

self.conv3 = conv1x1(planes, planes \* self.expansion)

self.bn3 = nn.BatchNorm2d(planes \* self.expansion, use\_batch\_statistics=use\_batch\_statistics)

self.relu = nn.ReLU()

self.downsample = downsample

self.add = P.TensorAdd()

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)

out = self.relu(out)

out = self.conv3(out)

out = self.bn3(out)

if self.downsample is not None:

identity = self.downsample(x)

out = self.add(out, identity)

out = self.relu(out)

return out

class ASPP(nn.Cell):

def \_\_init\_\_(self, atrous\_rates, phase='train', in\_channels=2048, num\_classes=21,

use\_batch\_statistics=True):

super(ASPP, self).\_\_init\_\_()

self.phase = phase

out\_channels = 256

self.aspp1 = ASPPConv(in\_channels, out\_channels, atrous\_rates[0], use\_batch\_statistics=use\_batch\_statistics)

self.aspp2 = ASPPConv(in\_channels, out\_channels, atrous\_rates[1], use\_batch\_statistics=use\_batch\_statistics)

self.aspp3 = ASPPConv(in\_channels, out\_channels, atrous\_rates[2], use\_batch\_statistics=use\_batch\_statistics)

self.aspp4 = ASPPConv(in\_channels, out\_channels, atrous\_rates[3], use\_batch\_statistics=use\_batch\_statistics)

self.aspp\_pooling = ASPPPooling(in\_channels, out\_channels)

self.conv1 = nn.Conv2d(out\_channels \* (len(atrous\_rates) + 1), out\_channels, kernel\_size=1,

weight\_init='xavier\_uniform')

self.bn1 = nn.BatchNorm2d(out\_channels, use\_batch\_statistics=use\_batch\_statistics)

self.relu = nn.ReLU()

self.conv2 = nn.Conv2d(out\_channels, num\_classes, kernel\_size=1, weight\_init='xavier\_uniform', has\_bias=True)

self.concat = P.Concat(axis=1)

self.drop = nn.Dropout(0.3)

def construct(self, x):

x1 = self.aspp1(x)

x2 = self.aspp2(x)

x3 = self.aspp3(x)

x4 = self.aspp4(x)

x5 = self.aspp\_pooling(x)

x = self.concat((x1, x2))

x = self.concat((x, x3))

x = self.concat((x, x4))

x = self.concat((x, x5))

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

if self.phase == 'train':

x = self.drop(x)

x = self.conv2(x)

return x

class ASPPPooling(nn.Cell):

def \_\_init\_\_(self, in\_channels, out\_channels, use\_batch\_statistics=True):

super(ASPPPooling, self).\_\_init\_\_()

self.conv = nn.SequentialCell([

nn.Conv2d(in\_channels, out\_channels, kernel\_size=1, weight\_init='xavier\_uniform'),

nn.BatchNorm2d(out\_channels, use\_batch\_statistics=use\_batch\_statistics),

nn.ReLU()

])

self.shape = P.Shape()

def construct(self, x):

size = self.shape(x)

out = nn.AvgPool2d(size[2])(x)

out = self.conv(out)

out = P.ResizeNearestNeighbor((size[2], size[3]), True)(out)

return out

class ASPPConv(nn.Cell):

def \_\_init\_\_(self, in\_channels, out\_channels, atrous\_rate=1, use\_batch\_statistics=True):

super(ASPPConv, self).\_\_init\_\_()

if atrous\_rate == 1:

conv = nn.Conv2d(in\_channels, out\_channels, kernel\_size=1, has\_bias=False, weight\_init='xavier\_uniform')

else:

conv = nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, pad\_mode='pad', padding=atrous\_rate,

dilation=atrous\_rate, weight\_init='xavier\_uniform')

bn = nn.BatchNorm2d(out\_channels, use\_batch\_statistics=use\_batch\_statistics)

relu = nn.ReLU()

self.aspp\_conv = nn.SequentialCell([conv, bn, relu])

def construct(self, x):

out = self.aspp\_conv(x)

return out

class DeepLabV3(nn.Cell):

def \_\_init\_\_(self, phase='train', num\_classes=21, output\_stride=16, freeze\_bn=False):

super(DeepLabV3, self).\_\_init\_\_()

use\_batch\_statistics = not freeze\_bn

self.resnet = Resnet(Bottleneck, [3, 4, 23, 3], output\_stride=output\_stride,

use\_batch\_statistics=use\_batch\_statistics)

self.aspp = ASPP([1, 6, 12, 18], phase, 2048, num\_classes,

use\_batch\_statistics=use\_batch\_statistics)

self.shape = P.Shape()

def construct(self, x):

size = self.shape(x)

out = self.resnet(x)

out = self.aspp(out)

out = P.ResizeBilinear((size[2], size[3]), True)(out)

return out

定义不同的学习率

以下代码定义不同的学习率函数。

#定义不同的学习率

def cosine\_lr(base\_lr, decay\_steps, total\_steps):

for i in range(total\_steps):

step\_ = min(i, decay\_steps)

yield base\_lr \* 0.5 \* (1 + np.cos(np.pi \* step\_ / decay\_steps))

def poly\_lr(base\_lr, decay\_steps, total\_steps, end\_lr=0.0001, power=0.9):

for i in range(total\_steps):

step\_ = min(i, decay\_steps)

yield (base\_lr - end\_lr) \* ((1.0 - step\_ / decay\_steps) \*\* power) + end\_lr

def exponential\_lr(base\_lr, decay\_steps, decay\_rate, total\_steps, staircase=False):

for i in range(total\_steps):

if staircase:

power\_ = i // decay\_steps

else:

power\_ = float(i) / decay\_steps

yield base\_lr \* (decay\_rate \*\* power\_)

定义损失函数

以下代码定义了损失函数。

from mindspore import Tensor

import mindspore.common.dtype as mstype

import mindspore.nn as nn

from mindspore.ops import operations as P

class SoftmaxCrossEntropyLoss(nn.Cell):

def \_\_init\_\_(self, num\_cls=21, ignore\_label=255):

super(SoftmaxCrossEntropyLoss, self).\_\_init\_\_()

self.one\_hot = P.OneHot(axis=-1)

self.on\_value = Tensor(1.0, mstype.float32)

self.off\_value = Tensor(0.0, mstype.float32)

self.cast = P.Cast()

self.ce = nn.SoftmaxCrossEntropyWithLogits()

self.not\_equal = P.NotEqual()

self.num\_cls = num\_cls

self.ignore\_label = ignore\_label

self.mul = P.Mul()

self.sum = P.ReduceSum(False)

self.div = P.RealDiv()

self.transpose = P.Transpose()

self.reshape = P.Reshape()

def construct(self, logits, labels):

labels\_int = self.cast(labels, mstype.int32)

labels\_int = self.reshape(labels\_int, (-1,))

logits\_ = self.transpose(logits, (0, 2, 3, 1))

logits\_ = self.reshape(logits\_, (-1, self.num\_cls))

weights = self.not\_equal(labels\_int, self.ignore\_label)

weights = self.cast(weights, mstype.float32)

one\_hot\_labels = self.one\_hot(labels\_int, self.num\_cls, self.on\_value, self.off\_value)

loss = self.ce(logits\_, one\_hot\_labels)

loss = self.mul(weights, loss)

loss = self.div(self.sum(loss), self.sum(weights))

return loss

构建训练网络的函数

"""train DeepLabv3."""

import os

import sys

sys.path.insert(0,'./deeplabv3/deeplabv3\_2/') # your code path

from easydict import EasyDict as edict

import shutil

# import moxing as mox

from mindspore import context

from mindspore.train.model import ParallelMode, Model

import mindspore.nn as nn

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.communication.management import init, get\_rank, get\_group\_size

from mindspore.train.callback import LossMonitor, TimeMonitor

from mindspore.train.loss\_scale\_manager import FixedLossScaleManager

from mindspore.common import set\_seed

set\_seed(1)

context.set\_context(mode=context.GRAPH\_MODE, enable\_auto\_mixed\_precision=True, save\_graphs=False,

device\_target="Ascend")

class BuildTrainNetwork(nn.Cell):

def \_\_init\_\_(self, network, criterion):

super(BuildTrainNetwork, self).\_\_init\_\_()

self.network = network

self.criterion = criterion

def construct(self, input\_data, label):

output = self.network(input\_data)

net\_loss = self.criterion(output, label)

return net\_loss

def train(args):

# init multicards training

if args.is\_distributed:

init()

args.rank = get\_rank()

args.group\_size = get\_group\_size()

parallel\_mode = ParallelMode.DATA\_PARALLEL

context.set\_auto\_parallel\_context(parallel\_mode=parallel\_mode, gradients\_mean=True, device\_num=args.group\_size)

# dataset

dataset = SegDataset(image\_mean=args.image\_mean,

image\_std=args.image\_std,

data\_file=args.data\_file,

batch\_size=args.batch\_size,

crop\_size=args.crop\_size,

max\_scale=args.max\_scale,

min\_scale=args.min\_scale,

ignore\_label=args.ignore\_label,

num\_classes=args.num\_classes,

num\_readers=2,

num\_parallel\_calls=4,

shard\_id=args.rank,

shard\_num=args.group\_size)

dataset.get\_gray\_dataset()

dataset.get\_mindrecord\_dataset(is\_training=True)

dataset = dataset.get\_dataset(repeat=1)

# network

if args.model == 'deeplab\_v3\_s16':

network = DeepLabV3('train', args.num\_classes, 16, args.freeze\_bn)

elif args.model == 'deeplab\_v3\_s8':

network = DeepLabV3('train', args.num\_classes, 8, args.freeze\_bn)

else:

raise NotImplementedError('model [{:s}] not recognized'.format(args.model))

# loss

loss\_ = SoftmaxCrossEntropyLoss(args.num\_classes, args.ignore\_label)

loss\_.add\_flags\_recursive(fp32=True)

train\_net = BuildTrainNetwork(network, loss\_)

# load pretrained model

param\_dict = load\_checkpoint(args.ckpt\_file)

load\_param\_into\_net(train\_net, param\_dict)

# optimizer

iters\_per\_epoch = dataset.get\_dataset\_size()

total\_train\_steps = iters\_per\_epoch \* args.train\_epochs

if args.lr\_type == 'cos':

lr\_iter = cosine\_lr(args.base\_lr, total\_train\_steps, total\_train\_steps)

elif args.lr\_type == 'poly':

lr\_iter = poly\_lr(args.base\_lr, total\_train\_steps, total\_train\_steps, end\_lr=0.0, power=0.9)

elif args.lr\_type == 'exp':

lr\_iter = exponential\_lr(args.base\_lr, args.lr\_decay\_step, args.lr\_decay\_rate,

total\_train\_steps, staircase=True)

else:

raise ValueError('unknown learning rate type')

opt = nn.Momentum(params=train\_net.trainable\_params(), learning\_rate=lr\_iter, momentum=0.9, weight\_decay=0.0001,

loss\_scale=args.loss\_scale)

# loss scale

manager\_loss\_scale = FixedLossScaleManager(args.loss\_scale, drop\_overflow\_update=False)

model = Model(train\_net, optimizer=opt, amp\_level="O3", loss\_scale\_manager=manager\_loss\_scale)

# callback for saving ckpts

time\_cb = TimeMonitor(data\_size=iters\_per\_epoch)

loss\_cb = LossMonitor()

cbs = [time\_cb, loss\_cb]

if args.rank == 0:

config\_ck = CheckpointConfig(save\_checkpoint\_steps=iters\_per\_epoch,

keep\_checkpoint\_max=args.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix=args.model, directory=args.train\_dir, config=config\_ck)

cbs.append(ckpoint\_cb)

model.train(args.train\_epochs, dataset, callbacks=cbs,dataset\_sink\_mode=True)

设定相关参数并训练网络

#设定相关参数

cfg = edict({

"batch\_size": 16,

"crop\_size": 513,

"image\_mean": [103.53, 116.28, 123.675],

"image\_std": [57.375, 57.120, 58.395],

"min\_scale": 0.5,

"max\_scale": 2.0,

"ignore\_label": 255,

"num\_classes": 21,

"train\_epochs" : 3,

"lr\_type": 'cos',

"base\_lr": 0.0,

"lr\_decay\_step": 3\*91,

"lr\_decay\_rate" :0.1,

"loss\_scale": 2048,

"model": 'deeplab\_v3\_s8',

'rank': 0,

'group\_size':1,

'keep\_checkpoint\_max':1,

'train\_dir': 'model',

'is\_distributed':False,

'freeze\_bn':True

})

if os.path.exists(cfg.train\_dir):

shutil.rmtree(cfg.train\_dir)

data\_path = './VOC2012'

cfg.data\_file = data\_path

ckpt\_path = './ckpt/deeplab\_v3\_s8-300\_11.ckpt'

cfg.ckpt\_file = ckpt\_path

train(cfg)

Fine-tune好的模型将放在”./model”文件夹中，默认的模型名为” deeplab\_v3\_s8-3\_91.ckpt”。

输出：

converting voc color png to gray png ...

converting done

creating mindrecord dataset...

number of samples: 1464

number of samples written: 1000

number of samples written: 1464

Create Mindrecord Done

epoch: 1 step: 91, loss is 0.0029648119

epoch time: 220090.707 ms, per step time: 2418.579 ms

epoch: 2 step: 91, loss is 0.004314194

epoch time: 47422.051 ms, per step time: 521.121 ms

epoch: 3 step: 91, loss is 0.0039475723

epoch time: 47430.172 ms, per step time: 521.211 ms

验证网络

以下代码构建验证网络。

"""eval DeepLabv3."""

import os

import sys

sys.path.insert(0,'./deeplabv3/deeplabv3\_2/') # your code path

from easydict import EasyDict as edict

from PIL import Image

import PIL

import matplotlib.pyplot as plt

import matplotlib as mpl

import matplotlib.colors as colors

import numpy as np

import cv2

# import moxing as mox

from mindspore import Tensor

import mindspore.common.dtype as mstype

import mindspore.nn as nn

from mindspore import context

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", save\_graphs=False)

def cal\_hist(a, b, n):

k = (a >= 0) & (a < n)

return np.bincount(n \* a[k].astype(np.int32) + b[k], minlength=n \*\* 2).reshape(n, n)

def resize\_long(img, long\_size=513):

h, w, \_ = img.shape

if h > w:

new\_h = long\_size

new\_w = int(1.0 \* long\_size \* w / h)

else:

new\_w = long\_size

new\_h = int(1.0 \* long\_size \* h / w)

imo = cv2.resize(img, (new\_w, new\_h))

return imo

class BuildEvalNetwork(nn.Cell):

def \_\_init\_\_(self, network):

super(BuildEvalNetwork, self).\_\_init\_\_()

self.network = network

self.softmax = nn.Softmax(axis=1)

def construct(self, input\_data):

output = self.network(input\_data)

output = self.softmax(output)

return output

def pre\_process(args, img\_, crop\_size=513):

# resize

img\_ = resize\_long(img\_, crop\_size)

resize\_h, resize\_w, \_ = img\_.shape

# mean, std

image\_mean = np.array(args.image\_mean)

image\_std = np.array(args.image\_std)

img\_ = (img\_ - image\_mean) / image\_std

# pad to crop\_size

pad\_h = crop\_size - img\_.shape[0]

pad\_w = crop\_size - img\_.shape[1]

if pad\_h > 0 or pad\_w > 0:

img\_ = cv2.copyMakeBorder(img\_, 0, pad\_h, 0, pad\_w, cv2.BORDER\_CONSTANT, value=0)

# hwc to chw

img\_ = img\_.transpose((2, 0, 1))

return img\_, resize\_h, resize\_w

def eval\_batch(args, eval\_net, img\_lst, crop\_size=513, flip=True):

result\_lst = []

batch\_size = len(img\_lst)

batch\_img = np.zeros((args.batch\_size, 3, crop\_size, crop\_size), dtype=np.float32)

resize\_hw = []

for l in range(batch\_size):

img\_ = img\_lst[l]

img\_, resize\_h, resize\_w = pre\_process(args, img\_, crop\_size)

batch\_img[l] = img\_

resize\_hw.append([resize\_h, resize\_w])

batch\_img = np.ascontiguousarray(batch\_img)

net\_out = eval\_net(Tensor(batch\_img, mstype.float32))

net\_out = net\_out.asnumpy()

if flip:

batch\_img = batch\_img[:, :, :, ::-1]

net\_out\_flip = eval\_net(Tensor(batch\_img, mstype.float32))

net\_out += net\_out\_flip.asnumpy()[:, :, :, ::-1]

for bs in range(batch\_size):

probs\_ = net\_out[bs][:, :resize\_hw[bs][0], :resize\_hw[bs][1]].transpose((1, 2, 0))

ori\_h, ori\_w = img\_lst[bs].shape[0], img\_lst[bs].shape[1]

probs\_ = cv2.resize(probs\_, (ori\_w, ori\_h))

result\_lst.append(probs\_)

return result\_lst

def eval\_batch\_scales(args, eval\_net, img\_lst, scales,

base\_crop\_size=513, flip=True):

sizes\_ = [int((base\_crop\_size - 1) \* sc) + 1 for sc in scales]

probs\_lst = eval\_batch(args, eval\_net, img\_lst, crop\_size=sizes\_[0], flip=flip)

#print(sizes\_)

for crop\_size\_ in sizes\_[1:]:

probs\_lst\_tmp = eval\_batch(args, eval\_net, img\_lst, crop\_size=crop\_size\_, flip=flip)

for pl, \_ in enumerate(probs\_lst):

probs\_lst[pl] += probs\_lst\_tmp[pl]

result\_msk = []

for i in probs\_lst:

result\_msk.append(i.argmax(axis=2))

return result\_msk

# The color source: print(list(colors.cnames.keys()))

#print(list(colors.cnames.keys()))

num\_class = {0: 'background', 1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat', 5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat',

9: 'chair', 10: 'cow', 11: 'diningtable', 12: 'dog', 13: 'horse', 14: 'motorbike', 15: 'person', 16: 'pottedplant',

17: 'sheep', 18: 'sofa', 19: 'train', 20: 'tvmonitor', 21: 'edge'}

num\_color = {0:'aliceblue', 1:'grey', 2:'red', 3:'green', 4:'darkorange', 5:'lime', 6:'bisque',

7:'black', 8:'blanchedalmond', 9:'blue', 10:'blueviolet', 11:'brown', 12:'burlywood', 13:'cadetblue',

14:'darkorange', 15:'tan', 16:'darkviolet', 17:'cornflowerblue', 18:'yellow', 19:'crimson', 20:'darkcyan'}

color\_dic = [num\_color[k] for k in sorted(num\_color.keys())]

bounds = list(range(21))

cmap = mpl.colors.ListedColormap(color\_dic)

norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

def num\_to\_ClassAndColor(num\_list):

color\_ = []

class\_ = []

for num in num\_list:

color\_.append(num\_class[num])

class\_.append(num\_color[num])

return color\_,class\_

def net\_eval(args):

# network

if args.model == 'deeplab\_v3\_s16':

network = DeepLabV3('eval', args.num\_classes, 16, args.freeze\_bn)

elif args.model == 'deeplab\_v3\_s8':

network = DeepLabV3('eval', args.num\_classes, 8, args.freeze\_bn)

else:

raise NotImplementedError('model [{:s}] not recognized'.format(args.model))

eval\_net = BuildEvalNetwork(network)

# load model

param\_dict = load\_checkpoint(args.ckpt\_file)

load\_param\_into\_net(eval\_net, param\_dict)

eval\_net.set\_train(False)

# data list

with open(args.data\_lst) as f:

img\_lst = f.readlines()

# evaluate

hist = np.zeros((args.num\_classes, args.num\_classes))

batch\_img\_lst = []

batch\_msk\_lst = []

bi = 0

image\_num = 0

for i, line in enumerate(img\_lst):

id\_ = line.strip()

img\_path = os.path.join(cfg.voc\_img\_dir, id\_ + '.jpg')

msk\_path = os.path.join(cfg.voc\_anno\_gray\_dir, id\_ + '.png')

img\_ = cv2.imread(img\_path)

msk\_ = cv2.imread(msk\_path, cv2.IMREAD\_GRAYSCALE)

batch\_img\_lst.append(img\_)

batch\_msk\_lst.append(msk\_)

if args.if\_png:

batch\_res = eval\_batch\_scales(args, eval\_net, batch\_img\_lst, scales=args.scales,

base\_crop\_size=args.crop\_size, flip=args.flip)

height ,weight = batch\_res[0].shape

batch\_msk\_lst[0][batch\_msk\_lst[0]==args.ignore\_label] = 0

plt.figure(figsize=(3 \* weight/1024\*10, 2 \* height/1024\*10))

plt.subplot(1,3,1)

image = Image.open(img\_path)

plt.imshow(image)

plt.subplot(1,3,2)

plt.imshow(image)

plt.imshow(batch\_res[0],alpha=0.8,interpolation='none', cmap=cmap, norm=norm)

plt.subplot(1,3,3)

plt.imshow(image)

plt.imshow(batch\_msk\_lst[0],alpha=0.8,interpolation='none', cmap=cmap, norm=norm)

plt.show()

prediction\_num = np.unique(batch\_res[0])

real\_num = np.unique(batch\_msk\_lst[0])

prediction\_color,prediction\_class = num\_to\_ClassAndColor(prediction\_num)

print('prediction num:',prediction\_num)

print('prediction color:',prediction\_color)

print('prediction class:',prediction\_class)

real\_color,real\_class = num\_to\_ClassAndColor(real\_num)

print('groundtruth num:',real\_num)

print('groundtruth color:',real\_color)

print('groundtruth class:',real\_class)

batch\_img\_lst = []

batch\_msk\_lst = []

if i < args.num\_png-1:

continue

else:

return

bi += 1

if bi == args.batch\_size:

batch\_res = eval\_batch\_scales(args, eval\_net, batch\_img\_lst, scales=args.scales,

base\_crop\_size=args.crop\_size, flip=args.flip)

for mi in range(args.batch\_size):

hist += cal\_hist(batch\_msk\_lst[mi].flatten(), batch\_res[mi].flatten(), args.num\_classes)

bi = 0

batch\_img\_lst = []

batch\_msk\_lst = []

if (i+1)%100 == 0:

print('processed {} images'.format(i+1))

image\_num = i

if bi > 0:

batch\_res = eval\_batch\_scales(args, eval\_net, batch\_img\_lst, scales=args.scales,

base\_crop\_size=args.crop\_size, flip=args.flip)

for mi in range(bi):

hist += cal\_hist(batch\_msk\_lst[mi].flatten(), batch\_res[mi].flatten(), args.num\_classes)

if (i+1) % 100 == 0:

print('processed {} images'.format(image\_num + 1))

iu = np.diag(hist) / (hist.sum(1) + hist.sum(0) - np.diag(hist))

print('mean IoU', np.nanmean(iu))

以下代码是从数据集读取图片进行网络验证，不显示图片。

# test 1

cfg = edict({

"batch\_size": 1,

"crop\_size": 513,

"image\_mean": [103.53, 116.28, 123.675],

"image\_std": [57.375, 57.120, 58.395],

"scales": [1.0], # [0.5,0.75,1.0,1.25,1.75]

'flip': True,

'ignore\_label': 255,

'num\_classes':21,

'model': 'deeplab\_v3\_s8',

'freeze\_bn': True,

'if\_png':False,

'num\_png':10

})

data\_path = './VOC2012'

# if not os.path.exists(data\_path):

# mox.file.copy\_parallel(src\_url="s3://share-course/dataset/voc2012\_raw/", dst\_url=data\_path)

cfg.data\_file = data\_path

# dataset

dataset = SegDataset(image\_mean=cfg.image\_mean,

image\_std=cfg.image\_std,

data\_file=cfg.data\_file)

dataset.get\_gray\_dataset()

cfg.data\_lst = os.path.join(cfg.data\_file,'ImageSets/Segmentation/val.txt')

cfg.voc\_img\_dir = os.path.join(cfg.data\_file,'JPEGImages')

cfg.voc\_anno\_gray\_dir = os.path.join(cfg.data\_file,'SegmentationClassGray')

ckpt\_path = './model'

# if not os.path.exists(ckpt\_path):

# mox.file.copy\_parallel(src\_url="s3://yyq-3/DATA/code/deeplabv3/model", dst\_url=ckpt\_path) #if yours model had saved

cfg.ckpt\_file = os.path.join(ckpt\_path,'deeplab\_v3\_s8-3\_91.ckpt')

print('loading checkpoing:',cfg.ckpt\_file)

net\_eval(cfg)

输出：

the gray file is already exists！

loading checkpoing: ./model/deeplab\_v3\_s8-3\_91.ckpt

processed 100 images

processed 200 images

processed 300 images

processed 400 images

processed 500 images

processed 600 images

processed 700 images

processed 800 images

processed 900 images

processed 1000 images

processed 1100 images

processed 1200 images

processed 1300 images

processed 1400 images

mean IoU 0.7759366796455764

以下代码进行网络验证，显示最终图片和结果。

# test 2

cfg = edict({

"batch\_size": 1,

"crop\_size": 513,

"image\_mean": [103.53, 116.28, 123.675],

"image\_std": [57.375, 57.120, 58.395],

"scales": [1.0], # [0.5,0.75,1.0,1.25,1.75]

'flip': True,

'ignore\_label': 255,

'num\_classes':21,

'model': 'deeplab\_v3\_s8',

'freeze\_bn': True,

'if\_png':True,

'num\_png':3

})

# import moxing as mox

data\_path = './VOC2012'

# if not os.path.exists(data\_path):

# mox.file.copy\_parallel(src\_url="s3://share-course/dataset/voc2012\_raw/", dst\_url=data\_path)

cfg.data\_file = data\_path

# dataset

dataset = SegDataset(image\_mean=cfg.image\_mean,

image\_std=cfg.image\_std,

data\_file=cfg.data\_file)

dataset.get\_gray\_dataset()

cfg.data\_lst = os.path.join(cfg.data\_file,'ImageSets/Segmentation/val.txt')

cfg.voc\_img\_dir = os.path.join(cfg.data\_file,'JPEGImages')

cfg.voc\_anno\_gray\_dir = os.path.join(cfg.data\_file,'SegmentationClassGray')

ckpt\_path = './model'

# if not os.path.exists(ckpt\_path):

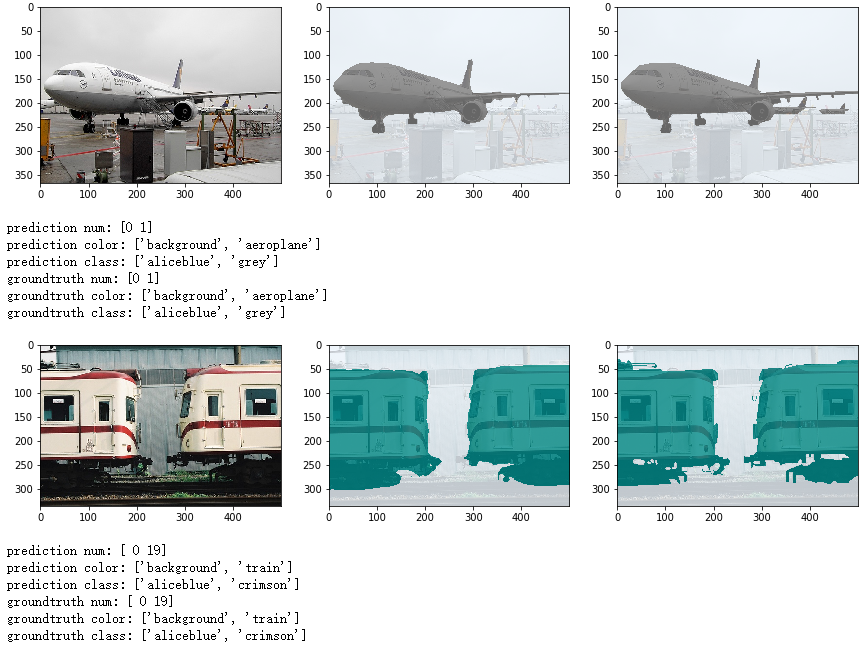
# mox.file.copy\_parallel(src\_url="s3://yyq-3/DATA/code/deeplabv3/model", dst\_url=ckpt\_path) #if yours model had saved

cfg.ckpt\_file = os.path.join(ckpt\_path,'deeplab\_v3\_s8-3\_91.ckpt')

print('loading checkpoing:',cfg.ckpt\_file)

net\_eval(cfg)

输出：



## 实验总结

本实验主要介绍如何使用MindSpore在voc2012数据集上训练和推理DeepLabv3网络模型，从而实现图像语义分割任务。通过本实验学员将了解如何处理图像分割数据标签，定义和训练卷积神经网络等的基本操作。

# 前沿网络案例-YOLOV3

## 实验介绍

目标检测是很多计算机视觉应用的基础，它结合了目标分类和定位两个任务。深度学习用于目标检测的算法从思路上来看，可以分为两大类，一类是two stage的方法，也就是把整个分为两部分，生成候选框和识别框内物体，例如R-CNN系列；另一类是one stage的方法，把整个流程统一在一起，直接给出检测结果，主要包含SSD,YOLO系列。目标检测的backbone一般是基于ImageNet预训练的图像分类网络。图像分类问题只关注分类和感受视野，不用关注物体定位，但是目标检测领域同时很关注空间信息。如果下采样过多，会导致最后的feature map很小，小目标很容易漏掉。很多基础架构网络，比如ResNet、FPN等神经网络提取图像的上下文信息，不断在特征提取方向优化。

本实验主要介绍使用MindSpore开发和训练Yolov3模型。本实验实现了目标检测（人、脸、口罩）。

## 实验环境要求

ModelArts平台：Mindspore

关于如何在ModelArts平台快速搭建环境，可参考文末附录：ModelArts开发环境搭建。

OBS桶：用于保存、导出模型文件等

首先点击“创建桶”，文件夹的创建可在实验过程中对应代码的具体步骤进行；创建桶链接： [https://storage.huaweicloud.com/obs/?region=cn-north-4#/obs/manager/buckets](https://storage.huaweicloud.com/obs/?region=cn-north-4" \l "/obs/manager/buckets)

## 实验准备

本实验所用数据集可参考下方链接进行下载，方便本地预览；由于此数据集过大，本实验将在代码中利用华为云ModelArts的Moxing库直接从OBS桶内导入，不必下载至本地后上传至云上训练环境。

数据集下载链接：<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/mask_detection_500.tar.gz>

数据集格式如下：

- train: 训练数据集.

- \*.jpg: 训练集图片

- \*.xml：训练集标签

- test: 测试数据集.

- \*.jpg: 测试集图片

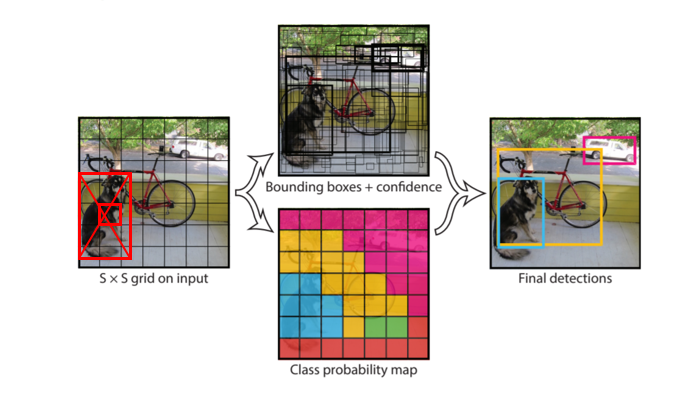
数据集包含三类，分别为：人（person），脸（face）、口罩（mask）。

## 背景知识

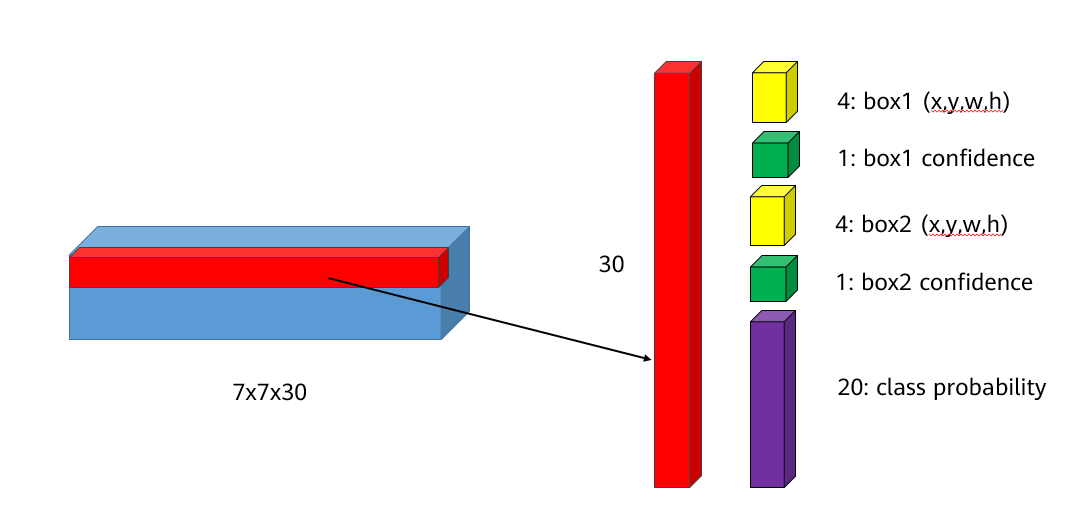
### 网络介绍

YOLO网络介绍

YOLO是单阶段方法的开山之作。它将检测任务表述成一个统一的、端到端的回归问题，并且以只处理一次图片同时得到位置和分类而得名。

YOLOV1是典型的目标检测one stage方法，用回归的方法去做目标检测，执行速度快，达到非常高效的检测。YOLOV1的基本思想是把一副图片，首先reshape成448×448大小（由于网络中使用了全连接层，所以图片的尺寸需固定大小输入到CNN中），然后将其划分成SxS个单元格（原文中S=7），如果目标中心点在某个单元格内，该单元格就负责预测该目标。输出层的大小为7x7，通道数为30。7x7可以看作将原图分为7x7的网格，而每个格子中有30个数。这三十个数分别对应了两组（意味着每个网格尝试着预测两个边界框）的“位置信息+置信度”以及20个类别（VOC数据集中有20个类别）。

YOLOv1预测示意图



YOLOv1输出层

YOLOV2，选择了5个锚作为召回率和模型复杂度之间的良好折衷。其关键特点：

1）Batch Normalization: YOLOv1没有使用BN层，而YOLOv2在每一层卷积层后都使用了BN层，BN层通过训练数据学习每一层每个神经元的缩放比例，进行标准化。BN层可以帮助网络进行训练，原论文指出，卷积层加了BN层后就可以不用dropout了，使用BN层后可以提高2%的mAP(平均准确率，mean average precision)。顺便一提的是，卷积层后加了BN层时，卷积层可以不使用偏置值。

2）High Resolution Classifier: 对YOLOV2，预训练之后，在ImageNet数据集上，用448\*448大小的图片对分类网络进行微调，大约10个epoches，其目的是让网络先学习一下高分辨率的图片，之后再应用到检测网络中，这个举措使得mAP提升大概4%。

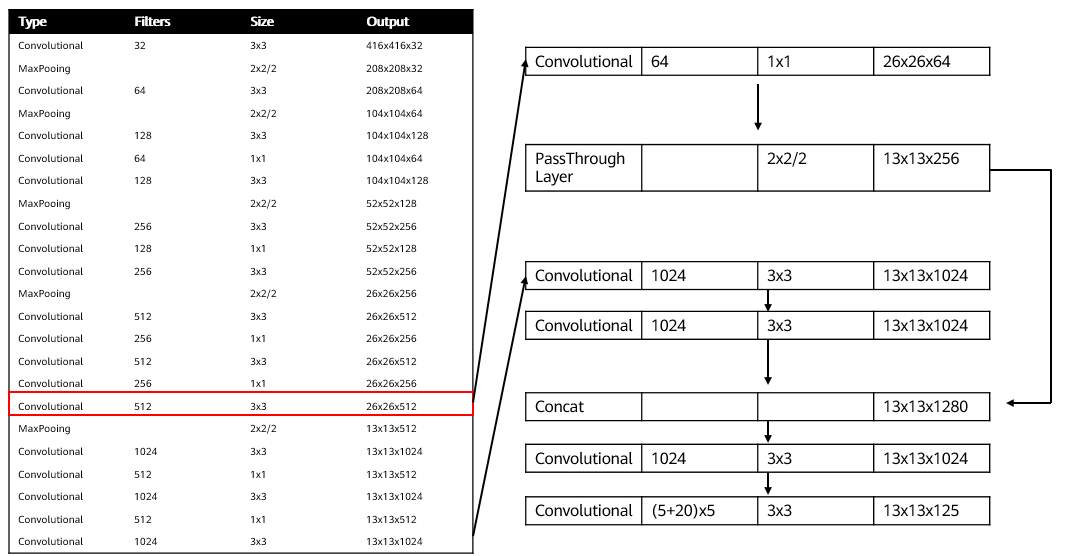
3）Convolutional With Anchor Boxes: YOLOv1并没有使用锚点，而是直接预测x,y,w,h，且每个网格预测两个边界框的形式总觉得很奇怪（因为同一个网格中的两个边界框并没有什么不同）。而YOLOv2引用了Faster RCNN和SSD模型中的锚点，预测的位置是相对预置的锚点的。论文指出通过使用锚点，mAP下降了0.3%的mAP，但是召回率增加了7%，虽然mAP下降了，但是更高的召回率意味着模型的上限更高。

4）Dimension Cluster: 对网络来说，如果能够选择合适的anchor尺寸，网络更加容易学习并且预测出更好的结果，在论文中作者使用k-means算法在训练集上的边界框中自动选择合适的box dimensions。

5）Direct location prediction: 作者在论文中提到，需要对x,y,w,h进行归一化（在输出层代表位置信息的部分使用sigmoid激活函数）。此外，置信度同样也需要进行归一化（输出层代码置信度的位置加sigmoid激活函数）。这样可以使得网络在训练过程中更加稳定。通过Dimension Clusters和Direct location prediction可以使模型提高5%的mAP。

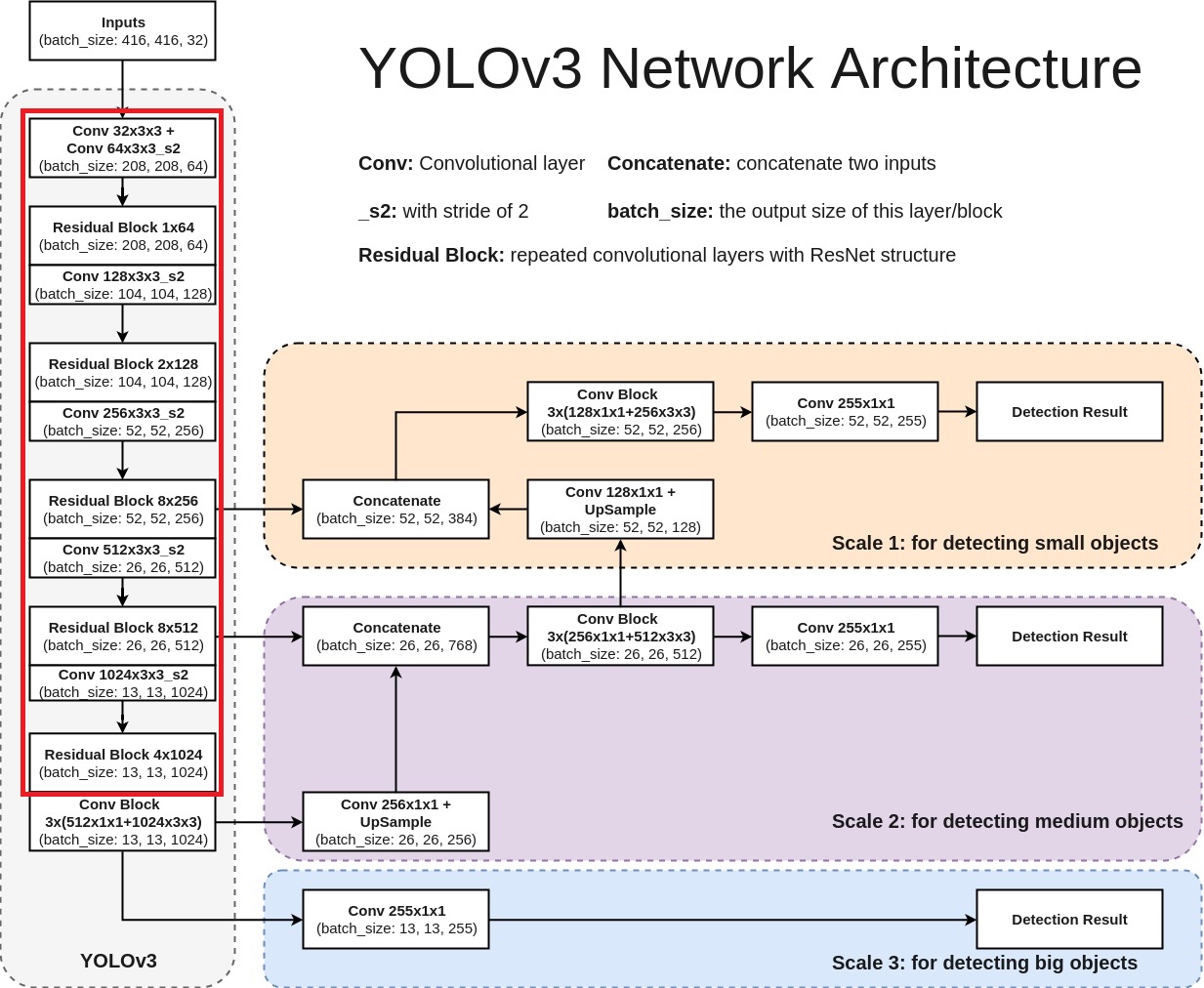
6）Fine-Grained Features:在13\*13特征图上进行目标检测，对于一些大的目标是足够的，但是对于小物体的检测还需要细粒度的特征，为此YOLOV2添加一个passthrough layer，将浅层的特征和深层的特征，两个不同尺寸的特征按通道维度拼接起来。值得一提的是，原论文中，作者写的是PassThrough层将26x26x512的特征图变为13x13x2048的特征图，但是实际上，在作者的代码实现中，在PassThrough层前使用了1x1的卷积将512维的通道数减少到了64维。因此，实际上，PassThrough层的输入为26x26x64，输出为13x13x256。而YOLOv2的最终网络结构如8-3所示。

7） Multi-Scale training：从上面的结构图可以看到，YOLOv2相比YOLOv1，去掉了全连接层，所有带参数的网络层均为卷积层和BN层（在表格中没画出来，每个卷积层后面都会跟一个BN层）。卷积层和BN层都不会受到输入图像大小的影响（如果网络有全连接层，输入图像的大小必须是一致的）。因此，作者提出，在训练模型时可以使用不同尺度的图像进行训练来保证模型对大目标和小目标都能达到不错的效果。由于网络的输入图像的大小为输出大小的32倍，因此，作者使用了多个尺寸为32倍数的图像对网络进行训练。输入图像的大小为{320，352，...，608}，每十个batch换一组尺寸。



YOLOv2网络结构图

YOLOv3相比YOLOv2最大的改进点在于借鉴了SSD的多尺度判别，即在不同大小的特征图上进行预测。对于网络前几层的大尺寸特征图，可以有效地检测出小目标，对于网络最后的小尺寸特征图可以有效地检测出大目标。此外，YOLOv3的backbone选择了DarkNet53网络，网络结构更深，特征提取能力更强了。YOLOv3的网络结构如下图所示，左侧中的红色框部分为去掉输出层的DarkNet53网络：



YOLOv3网络结构图

对于YOLOv3网络结构，有以下几点需要注意的：

1. 由于网络较深，使用了残差结构。
2. DarkNet53网络用步长为2的卷积代替了池化层。
3. 所有的网络层不包含全连接层，因此，输入图像的大小也是可以调整的（有些地方看到的可能是608x608，其实是一样的），而输入图像的大小同样是最小的输出特征图的32倍。
4. YOLOv3分别在三个尺寸的特征图进行了预测，每个尺寸的特征图使用了3个锚点。因此，输出层的维度计算方法为：(4+1+80)x3=255，因此，最后一层1x1的卷积层的数量为255。
5. 13 x 13的特征图会通过上采样层和之前的26 x 26的特征图在通道维度拼接在一起，26 x 26的特征图再经过上采样和52 x 52的特征图拼接。

本实验采用了基于Darknet-53的YOLOV3。

## 实验步骤

导入数据，定义数据处理相关函数

#导入数据

import moxing as mox

mox.file.copy\_parallel(src\_url="obs://ascend-professional-construction-dataset/deep-learning/yolov3-mindspore/",dst\_url="./")

以下代码用于定义相关数据集预处理的函数，如：生成Mindrecord文件，生成数据集对象等。

"""YOLOv3 dataset"""

from \_\_future\_\_ import division

import os

from xml.dom.minidom import parse

import xml.dom.minidom

import numpy as np

from matplotlib.colors import rgb\_to\_hsv, hsv\_to\_rgb

from PIL import Image

import mindspore.dataset as de

from mindspore.mindrecord import FileWriter

import mindspore.dataset.vision.c\_transforms as C

def preprocess\_fn(image, box, file, is\_training):

"""Preprocess function for dataset."""

config\_anchors = []

temp = ConfigYOLOV3ResNet18.anchor\_scales

for i in temp:

config\_anchors+=list(i)

anchors = np.array([float(x) for x in config\_anchors]).reshape(-1, 2)

do\_hsv = False

max\_boxes = ConfigYOLOV3ResNet18.\_NUM\_BOXES

num\_classes = ConfigYOLOV3ResNet18.num\_classes

def \_rand(a=0., b=1.):

return np.random.rand() \* (b - a) + a

def \_preprocess\_true\_boxes(true\_boxes, anchors, in\_shape=None):

"""Get true boxes."""

num\_layers = anchors.shape[0] // 3

anchor\_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]

true\_boxes = np.array(true\_boxes, dtype='float32')

input\_shape = np.array(in\_shape, dtype='int32')

boxes\_xy = (true\_boxes[..., 0:2] + true\_boxes[..., 2:4]) // 2.

boxes\_wh = true\_boxes[..., 2:4] - true\_boxes[..., 0:2]

true\_boxes[..., 0:2] = boxes\_xy / input\_shape[::-1]

true\_boxes[..., 2:4] = boxes\_wh / input\_shape[::-1]

grid\_shapes = [input\_shape // 32, input\_shape // 16, input\_shape // 8]

y\_true = [np.zeros((grid\_shapes[l][0], grid\_shapes[l][1], len(anchor\_mask[l]),

5 + num\_classes), dtype='float32') for l in range(num\_layers)]

anchors = np.expand\_dims(anchors, 0)

anchors\_max = anchors / 2.

anchors\_min = -anchors\_max

valid\_mask = boxes\_wh[..., 0] >= 1

wh = boxes\_wh[valid\_mask]

if len(wh) >= 1:

wh = np.expand\_dims(wh, -2)

boxes\_max = wh / 2.

boxes\_min = -boxes\_max

intersect\_min = np.maximum(boxes\_min, anchors\_min)

intersect\_max = np.minimum(boxes\_max, anchors\_max)

intersect\_wh = np.maximum(intersect\_max - intersect\_min, 0.)

intersect\_area = intersect\_wh[..., 0] \* intersect\_wh[..., 1]

box\_area = wh[..., 0] \* wh[..., 1]

anchor\_area = anchors[..., 0] \* anchors[..., 1]

iou = intersect\_area / (box\_area + anchor\_area - intersect\_area)

best\_anchor = np.argmax(iou, axis=-1)

for t, n in enumerate(best\_anchor):

for l in range(num\_layers):

if n in anchor\_mask[l]:

i = np.floor(true\_boxes[t, 0] \* grid\_shapes[l][1]).astype('int32')

j = np.floor(true\_boxes[t, 1] \* grid\_shapes[l][0]).astype('int32')

k = anchor\_mask[l].index(n)

c = true\_boxes[t, 4].astype('int32')

y\_true[l][j, i, k, 0:4] = true\_boxes[t, 0:4]

y\_true[l][j, i, k, 4] = 1.

y\_true[l][j, i, k, 5 + c] = 1.

pad\_gt\_box0 = np.zeros(shape=[ConfigYOLOV3ResNet18.\_NUM\_BOXES, 4], dtype=np.float32)

pad\_gt\_box1 = np.zeros(shape=[ConfigYOLOV3ResNet18.\_NUM\_BOXES, 4], dtype=np.float32)

pad\_gt\_box2 = np.zeros(shape=[ConfigYOLOV3ResNet18.\_NUM\_BOXES, 4], dtype=np.float32)

mask0 = np.reshape(y\_true[0][..., 4:5], [-1])

gt\_box0 = np.reshape(y\_true[0][..., 0:4], [-1, 4])

gt\_box0 = gt\_box0[mask0 == 1]

pad\_gt\_box0[:gt\_box0.shape[0]] = gt\_box0

mask1 = np.reshape(y\_true[1][..., 4:5], [-1])

gt\_box1 = np.reshape(y\_true[1][..., 0:4], [-1, 4])

gt\_box1 = gt\_box1[mask1 == 1]

pad\_gt\_box1[:gt\_box1.shape[0]] = gt\_box1

mask2 = np.reshape(y\_true[2][..., 4:5], [-1])

gt\_box2 = np.reshape(y\_true[2][..., 0:4], [-1, 4])

gt\_box2 = gt\_box2[mask2 == 1]

pad\_gt\_box2[:gt\_box2.shape[0]] = gt\_box2

return y\_true[0], y\_true[1], y\_true[2], pad\_gt\_box0, pad\_gt\_box1, pad\_gt\_box2

def \_infer\_data(img\_data, input\_shape, box):

w, h = img\_data.size

input\_h, input\_w = input\_shape

scale = min(float(input\_w) / float(w), float(input\_h) / float(h))

nw = int(w \* scale)

nh = int(h \* scale)

img\_data = img\_data.resize((nw, nh), Image.BICUBIC)

new\_image = np.zeros((input\_h, input\_w, 3), np.float32)

new\_image.fill(128)

img\_data = np.array(img\_data)

if len(img\_data.shape) == 2:

img\_data = np.expand\_dims(img\_data, axis=-1)

img\_data = np.concatenate([img\_data, img\_data, img\_data], axis=-1)

dh = int((input\_h - nh) / 2)

dw = int((input\_w - nw) / 2)

new\_image[dh:(nh + dh), dw:(nw + dw), :] = img\_data

new\_image /= 255.

new\_image = np.transpose(new\_image, (2, 0, 1))

new\_image = np.expand\_dims(new\_image, 0)

return new\_image, np.array([h, w], np.float32), box

def \_data\_aug(image, box, is\_training, jitter=0.3, hue=0.1, sat=1.5, val=1.5, image\_size=(352, 640)):

"""Data augmentation function."""

if not isinstance(image, Image.Image):

image = Image.fromarray(image)

iw, ih = image.size

ori\_image\_shape = np.array([ih, iw], np.int32)

h, w = image\_size

if not is\_training:

return \_infer\_data(image, image\_size, box)

flip = \_rand() < .5

# correct boxes

box\_data = np.zeros((max\_boxes, 5))

flag =0

while True:

# Prevent the situation that all boxes are eliminated

new\_ar = float(w) / float(h) \* \_rand(1 - jitter, 1 + jitter) / \

\_rand(1 - jitter, 1 + jitter)

scale = \_rand(0.25, 2)

if new\_ar < 1:

nh = int(scale \* h)

nw = int(nh \* new\_ar)

else:

nw = int(scale \* w)

nh = int(nw / new\_ar)

dx = int(\_rand(0, w - nw))

dy = int(\_rand(0, h - nh))

flag = flag + 1

if len(box) >= 1:

t\_box = box.copy()

np.random.shuffle(t\_box)

t\_box[:, [0, 2]] = t\_box[:, [0, 2]] \* float(nw) / float(iw) + dx

t\_box[:, [1, 3]] = t\_box[:, [1, 3]] \* float(nh) / float(ih) + dy

if flip:

t\_box[:, [0, 2]] = w - t\_box[:, [2, 0]]

t\_box[:, 0:2][t\_box[:, 0:2] < 0] = 0

t\_box[:, 2][t\_box[:, 2] > w] = w

t\_box[:, 3][t\_box[:, 3] > h] = h

box\_w = t\_box[:, 2] - t\_box[:, 0]

box\_h = t\_box[:, 3] - t\_box[:, 1]

t\_box = t\_box[np.logical\_and(box\_w > 1, box\_h > 1)] # discard invalid box

if len(t\_box) >= 1:

box = t\_box

break

box\_data[:len(box)] = box

# resize image

image = image.resize((nw, nh), Image.BICUBIC)

# place image

new\_image = Image.new('RGB', (w, h), (128, 128, 128))

new\_image.paste(image, (dx, dy))

image = new\_image

# flip image or not

if flip:

image = image.transpose(Image.FLIP\_LEFT\_RIGHT)

# convert image to gray or not

gray = \_rand() < .25

if gray:

image = image.convert('L').convert('RGB')

# when the channels of image is 1

image = np.array(image)

if len(image.shape) == 2:

image = np.expand\_dims(image, axis=-1)

image = np.concatenate([image, image, image], axis=-1)

# distort image

hue = \_rand(-hue, hue)

sat = \_rand(1, sat) if \_rand() < .5 else 1 / \_rand(1, sat)

val = \_rand(1, val) if \_rand() < .5 else 1 / \_rand(1, val)

image\_data = image / 255.

if do\_hsv:

x = rgb\_to\_hsv(image\_data)

x[..., 0] += hue

x[..., 0][x[..., 0] > 1] -= 1

x[..., 0][x[..., 0] < 0] += 1

x[..., 1] \*= sat

x[..., 2] \*= val

x[x > 1] = 1

x[x < 0] = 0

image\_data = hsv\_to\_rgb(x) # numpy array, 0 to 1

image\_data = image\_data.astype(np.float32)

# preprocess bounding boxes

bbox\_true\_1, bbox\_true\_2, bbox\_true\_3, gt\_box1, gt\_box2, gt\_box3 = \

\_preprocess\_true\_boxes(box\_data, anchors, image\_size)

return image\_data, bbox\_true\_1, bbox\_true\_2, bbox\_true\_3, \

ori\_image\_shape, gt\_box1, gt\_box2, gt\_box3

if is\_training:

images, bbox\_1, bbox\_2, bbox\_3, image\_shape, gt\_box1, gt\_box2, gt\_box3 = \_data\_aug(image, box, is\_training)

return images, bbox\_1, bbox\_2, bbox\_3, gt\_box1, gt\_box2, gt\_box3

images, shape, anno = \_data\_aug(image, box, is\_training)

return images, shape, anno, file

def xy\_local(collection,element):

xy = collection.getElementsByTagName(element)[0]

xy = xy.childNodes[0].data

return xy

def filter\_valid\_data(image\_dir):

"""Filter valid image file, which both in image\_dir and anno\_path."""

label\_id={'person':0, 'face':1, 'mask':2}

all\_files = os.listdir(image\_dir)

image\_dict = {}

image\_files=[]

for i in all\_files:

if (i[-3:]=='jpg' or i[-4:]=='jpeg') and i not in image\_dict:

image\_files.append(i)

label=[]

xml\_path = os.path.join(image\_dir,i[:-3]+'xml')

if not os.path.exists(xml\_path):

label=[[0,0,0,0,0]]

image\_dict[i]=label

continue

DOMTree = xml.dom.minidom.parse(xml\_path)

collection = DOMTree.documentElement

# 在集合中获取所有框

object\_ = collection.getElementsByTagName("object")

for m in object\_:

temp=[]

name = m.getElementsByTagName('name')[0]

class\_num = label\_id[name.childNodes[0].data]

bndbox = m.getElementsByTagName('bndbox')[0]

xmin = xy\_local(bndbox,'xmin')

ymin = xy\_local(bndbox,'ymin')

xmax = xy\_local(bndbox,'xmax')

ymax = xy\_local(bndbox,'ymax')

temp.append(int(xmin))

temp.append(int(ymin))

temp.append(int(xmax))

temp.append(int(ymax))

temp.append(class\_num)

label.append(temp)

image\_dict[i]=label

return image\_files, image\_dict

def data\_to\_mindrecord\_byte\_image(image\_dir, mindrecord\_dir, prefix, file\_num):

"""Create MindRecord file by image\_dir and anno\_path."""

mindrecord\_path = os.path.join(mindrecord\_dir, prefix)

writer = FileWriter(mindrecord\_path, file\_num)

image\_files, image\_anno\_dict = filter\_valid\_data(image\_dir)

yolo\_json = {

"image": {"type": "bytes"},

"annotation": {"type": "int32", "shape": [-1, 5]},

"file": {"type": "string"},

}

writer.add\_schema(yolo\_json, "yolo\_json")

for image\_name in image\_files:

image\_path = os.path.join(image\_dir, image\_name)

with open(image\_path, 'rb') as f:

img = f.read()

annos = np.array(image\_anno\_dict[image\_name],dtype=np.int32)

#print(annos.shape)

row = {"image": img, "annotation": annos, "file": image\_name}

writer.write\_raw\_data([row])

writer.commit()

def create\_yolo\_dataset(mindrecord\_dir, batch\_size=32, repeat\_num=1, device\_num=1, rank=0,

is\_training=True, num\_parallel\_workers=8):

"""Creatr YOLOv3 dataset with MindDataset."""

ds = de.MindDataset(mindrecord\_dir, columns\_list=["image", "annotation","file"], num\_shards=device\_num, shard\_id=rank,

num\_parallel\_workers=num\_parallel\_workers, shuffle=is\_training)

decode = C.Decode()

ds = ds.map(operations=decode, input\_columns=["image"])

compose\_map\_func = (lambda image, annotation, file: preprocess\_fn(image, annotation,file, is\_training))

if is\_training:

hwc\_to\_chw = C.HWC2CHW()

ds = ds.map(operations=compose\_map\_func, input\_columns=["image", "annotation","file"],

output\_columns=["image", "bbox\_1", "bbox\_2", "bbox\_3", "gt\_box1", "gt\_box2", "gt\_box3"],

column\_order=["image", "bbox\_1", "bbox\_2", "bbox\_3", "gt\_box1", "gt\_box2", "gt\_box3"],

num\_parallel\_workers=num\_parallel\_workers)

ds = ds.map(operations=hwc\_to\_chw, input\_columns=["image"], num\_parallel\_workers=num\_parallel\_workers)

ds = ds.batch(batch\_size, drop\_remainder=True)

ds = ds.repeat(repeat\_num)

else:

ds = ds.map(operations=compose\_map\_func, input\_columns=["image", "annotation","file"],

output\_columns=["image", "image\_shape", "annotation","file"],

column\_order=["image", "image\_shape", "annotation","file"],

num\_parallel\_workers=num\_parallel\_workers)

return ds

定义网络

以下代码实现生成YOLO网络所需的相关类。

"""YOLOv3 based on ResNet18."""

import numpy as np

import mindspore as ms

import mindspore.nn as nn

from mindspore import context, Tensor

from mindspore.context import ParallelMode

from mindspore.parallel.\_auto\_parallel\_context import auto\_parallel\_context

from mindspore.communication.management import get\_group\_size

from mindspore.common.initializer import TruncatedNormal

from mindspore.ops import operations as P

from mindspore.ops import functional as F

from mindspore.ops import composite as C

def weight\_variable():

"""Weight variable."""

return TruncatedNormal(0.02)

class \_conv2d(nn.Cell):

"""Create Conv2D with padding."""

def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, stride=1):

super(\_conv2d, self).\_\_init\_\_()

self.conv = nn.Conv2d(in\_channels, out\_channels,

kernel\_size=kernel\_size, stride=stride, padding=0, pad\_mode='same',

weight\_init=weight\_variable())

def construct(self, x):

x = self.conv(x)

return x

def \_fused\_bn(channels, momentum=0.99):

"""Get a fused batchnorm."""

return nn.BatchNorm2d(channels, momentum=momentum)

def \_conv\_bn\_relu(in\_channel,

out\_channel,

ksize,

stride=1,

padding=0,

dilation=1,

alpha=0.1,

momentum=0.99,

pad\_mode="same"):

"""Get a conv2d batchnorm and relu layer."""

return nn.SequentialCell(

[nn.Conv2d(in\_channel,

out\_channel,

kernel\_size=ksize,

stride=stride,

padding=padding,

dilation=dilation,

pad\_mode=pad\_mode),

nn.BatchNorm2d(out\_channel, momentum=momentum),

nn.LeakyReLU(alpha)]

)

class BasicBlock(nn.Cell):

"""

ResNet basic block.

Args:

in\_channels (int): Input channel.

out\_channels (int): Output channel.

stride (int): Stride size for the initial convolutional layer. Default:1.

momentum (float): Momentum for batchnorm layer. Default:0.1.

Returns:

Tensor, output tensor.

Examples:

BasicBlock(3,256,stride=2,down\_sample=True).

"""

expansion = 1

def \_\_init\_\_(self,

in\_channels,

out\_channels,

stride=1,

momentum=0.99):

super(BasicBlock, self).\_\_init\_\_()

self.conv1 = \_conv2d(in\_channels, out\_channels, 3, stride=stride)

self.bn1 = \_fused\_bn(out\_channels, momentum=momentum)

self.conv2 = \_conv2d(out\_channels, out\_channels, 3)

self.bn2 = \_fused\_bn(out\_channels, momentum=momentum)

self.relu = P.ReLU()

self.down\_sample\_layer = None

self.downsample = (in\_channels != out\_channels)

if self.downsample:

self.down\_sample\_layer = \_conv2d(in\_channels, out\_channels, 1, stride=stride)

self.add = P.TensorAdd()

def construct(self, x):

identity = x

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

x = self.conv2(x)

x = self.bn2(x)

if self.downsample:

identity = self.down\_sample\_layer(identity)

out = self.add(x, identity)

out = self.relu(out)

return out

class ResNet(nn.Cell):

"""

ResNet network.

Args:

block (Cell): Block for network.

layer\_nums (list): Numbers of different layers.

in\_channels (int): Input channel.

out\_channels (int): Output channel.

num\_classes (int): Class number. Default:100.

Returns:

Tensor, output tensor.

Examples:

ResNet(ResidualBlock,

[3, 4, 6, 3],

[64, 256, 512, 1024],

[256, 512, 1024, 2048],

100).

"""

def \_\_init\_\_(self,

block,

layer\_nums,

in\_channels,

out\_channels,

strides=None,

num\_classes=80):

super(ResNet, self).\_\_init\_\_()

if not len(layer\_nums) == len(in\_channels) == len(out\_channels) == 4:

raise ValueError("the length of "

"layer\_num, inchannel, outchannel list must be 4!")

self.conv1 = \_conv2d(3, 64, 7, stride=2)

self.bn1 = \_fused\_bn(64)

self.relu = P.ReLU()

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode='same')

self.layer1 = self.\_make\_layer(block,

layer\_nums[0],

in\_channel=in\_channels[0],

out\_channel=out\_channels[0],

stride=strides[0])

self.layer2 = self.\_make\_layer(block,

layer\_nums[1],

in\_channel=in\_channels[1],

out\_channel=out\_channels[1],

stride=strides[1])

self.layer3 = self.\_make\_layer(block,

layer\_nums[2],

in\_channel=in\_channels[2],

out\_channel=out\_channels[2],

stride=strides[2])

self.layer4 = self.\_make\_layer(block,

layer\_nums[3],

in\_channel=in\_channels[3],

out\_channel=out\_channels[3],

stride=strides[3])

self.num\_classes = num\_classes

if num\_classes:

self.reduce\_mean = P.ReduceMean(keep\_dims=True)

self.end\_point = nn.Dense(out\_channels[3], num\_classes, has\_bias=True,

weight\_init=weight\_variable(),

bias\_init=weight\_variable())

self.squeeze = P.Squeeze(axis=(2, 3))

def \_make\_layer(self, block, layer\_num, in\_channel, out\_channel, stride):

"""

Make Layer for ResNet.

Args:

block (Cell): Resnet block.

layer\_num (int): Layer number.

in\_channel (int): Input channel.

out\_channel (int): Output channel.

stride (int): Stride size for the initial convolutional layer.

Returns:

SequentialCell, the output layer.

Examples:

\_make\_layer(BasicBlock, 3, 128, 256, 2).

"""

layers = []

resblk = block(in\_channel, out\_channel, stride=stride)

layers.append(resblk)

for \_ in range(1, layer\_num - 1):

resblk = block(out\_channel, out\_channel, stride=1)

layers.append(resblk)

resblk = block(out\_channel, out\_channel, stride=1)

layers.append(resblk)

return nn.SequentialCell(layers)

def construct(self, x):

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

c1 = self.maxpool(x)

c2 = self.layer1(c1)

c3 = self.layer2(c2)

c4 = self.layer3(c3)

c5 = self.layer4(c4)

out = c5

if self.num\_classes:

out = self.reduce\_mean(c5, (2, 3))

out = self.squeeze(out)

out = self.end\_point(out)

return c3, c4, out

def resnet18(class\_num=10):

"""

Get ResNet18 neural network.

Args:

class\_num (int): Class number.

Returns:

Cell, cell instance of ResNet18 neural network.

Examples:

resnet18(100).

"""

return ResNet(BasicBlock,

[2, 2, 2, 2],

[64, 64, 128, 256],

[64, 128, 256, 512],

[1, 2, 2, 2],

num\_classes=class\_num)

class YoloBlock(nn.Cell):

"""

YoloBlock for YOLOv3.

Args:

in\_channels (int): Input channel.

out\_chls (int): Middle channel.

out\_channels (int): Output channel.

Returns:

Tuple, tuple of output tensor,(f1,f2,f3).

Examples:

YoloBlock(1024, 512, 255).

"""

def \_\_init\_\_(self, in\_channels, out\_chls, out\_channels):

super(YoloBlock, self).\_\_init\_\_()

out\_chls\_2 = out\_chls \* 2

self.conv0 = \_conv\_bn\_relu(in\_channels, out\_chls, ksize=1)

self.conv1 = \_conv\_bn\_relu(out\_chls, out\_chls\_2, ksize=3)

self.conv2 = \_conv\_bn\_relu(out\_chls\_2, out\_chls, ksize=1)

self.conv3 = \_conv\_bn\_relu(out\_chls, out\_chls\_2, ksize=3)

self.conv4 = \_conv\_bn\_relu(out\_chls\_2, out\_chls, ksize=1)

self.conv5 = \_conv\_bn\_relu(out\_chls, out\_chls\_2, ksize=3)

self.conv6 = nn.Conv2d(out\_chls\_2, out\_channels, kernel\_size=1, stride=1, has\_bias=True)

def construct(self, x):

c1 = self.conv0(x)

c2 = self.conv1(c1)

c3 = self.conv2(c2)

c4 = self.conv3(c3)

c5 = self.conv4(c4)

c6 = self.conv5(c5)

out = self.conv6(c6)

return c5, out

class YOLOv3(nn.Cell):

"""

YOLOv3 Network.

Note:

backbone = resnet18.

Args:

feature\_shape (list): Input image shape, [N,C,H,W].

backbone\_shape (list): resnet18 output channels shape.

backbone (Cell): Backbone Network.

out\_channel (int): Output channel.

Returns:

Tensor, output tensor.

Examples:

YOLOv3(feature\_shape=[1,3,416,416],

backbone\_shape=[64, 128, 256, 512, 1024]

backbone=darknet53(),

out\_channel=255).

"""

def \_\_init\_\_(self, feature\_shape, backbone\_shape, backbone, out\_channel):

super(YOLOv3, self).\_\_init\_\_()

self.out\_channel = out\_channel

self.net = backbone

self.backblock0 = YoloBlock(backbone\_shape[-1], out\_chls=backbone\_shape[-2], out\_channels=out\_channel)

self.conv1 = \_conv\_bn\_relu(in\_channel=backbone\_shape[-2], out\_channel=backbone\_shape[-2]//2, ksize=1)

self.upsample1 = P.ResizeNearestNeighbor((feature\_shape[2]//16, feature\_shape[3]//16))

self.backblock1 = YoloBlock(in\_channels=backbone\_shape[-2]+backbone\_shape[-3],

out\_chls=backbone\_shape[-3],

out\_channels=out\_channel)

self.conv2 = \_conv\_bn\_relu(in\_channel=backbone\_shape[-3], out\_channel=backbone\_shape[-3]//2, ksize=1)

self.upsample2 = P.ResizeNearestNeighbor((feature\_shape[2]//8, feature\_shape[3]//8))

self.backblock2 = YoloBlock(in\_channels=backbone\_shape[-3]+backbone\_shape[-4],

out\_chls=backbone\_shape[-4],

out\_channels=out\_channel)

self.concat = P.Concat(axis=1)

def construct(self, x):

# input\_shape of x is (batch\_size, 3, h, w)

# feature\_map1 is (batch\_size, backbone\_shape[2], h/8, w/8)

# feature\_map2 is (batch\_size, backbone\_shape[3], h/16, w/16)

# feature\_map3 is (batch\_size, backbone\_shape[4], h/32, w/32)

feature\_map1, feature\_map2, feature\_map3 = self.net(x)

con1, big\_object\_output = self.backblock0(feature\_map3)

con1 = self.conv1(con1)

ups1 = self.upsample1(con1)

con1 = self.concat((ups1, feature\_map2))

con2, medium\_object\_output = self.backblock1(con1)

con2 = self.conv2(con2)

ups2 = self.upsample2(con2)

con3 = self.concat((ups2, feature\_map1))

\_, small\_object\_output = self.backblock2(con3)

return big\_object\_output, medium\_object\_output, small\_object\_output

class DetectionBlock(nn.Cell):

"""

YOLOv3 detection Network. It will finally output the detection result.

Args:

scale (str): Character, scale.

config (Class): YOLOv3 config.

Returns:

Tuple, tuple of output tensor,(f1,f2,f3).

Examples:

DetectionBlock(scale='l',stride=32).

"""

def \_\_init\_\_(self, scale, config):

super(DetectionBlock, self).\_\_init\_\_()

self.config = config

if scale == 's':

idx = (0, 1, 2)

elif scale == 'm':

idx = (3, 4, 5)

elif scale == 'l':

idx = (6, 7, 8)

else:

raise KeyError("Invalid scale value for DetectionBlock")

self.anchors = Tensor([self.config.anchor\_scales[i] for i in idx], ms.float32)

self.num\_anchors\_per\_scale = 3

self.num\_attrib = 4 + 1 + self.config.num\_classes

self.ignore\_threshold = 0.5

self.lambda\_coord = 1

self.sigmoid = nn.Sigmoid()

self.reshape = P.Reshape()

self.tile = P.Tile()

self.concat = P.Concat(axis=-1)

self.input\_shape = Tensor(tuple(config.img\_shape[::-1]), ms.float32)

def construct(self, x):

num\_batch = P.Shape()(x)[0]

grid\_size = P.Shape()(x)[2:4]

# Reshape and transpose the feature to [n, 3, grid\_size[0], grid\_size[1], num\_attrib]

prediction = P.Reshape()(x, (num\_batch,

self.num\_anchors\_per\_scale,

self.num\_attrib,

grid\_size[0],

grid\_size[1]))

prediction = P.Transpose()(prediction, (0, 3, 4, 1, 2))

range\_x = range(grid\_size[1])

range\_y = range(grid\_size[0])

grid\_x = P.Cast()(F.tuple\_to\_array(range\_x), ms.float32)

grid\_y = P.Cast()(F.tuple\_to\_array(range\_y), ms.float32)

# Tensor of shape [grid\_size[0], grid\_size[1], 1, 1] representing the coordinate of x/y axis for each grid

grid\_x = self.tile(self.reshape(grid\_x, (1, 1, -1, 1, 1)), (1, grid\_size[0], 1, 1, 1))

grid\_y = self.tile(self.reshape(grid\_y, (1, -1, 1, 1, 1)), (1, 1, grid\_size[1], 1, 1))

# Shape is [grid\_size[0], grid\_size[1], 1, 2]

grid = self.concat((grid\_x, grid\_y))

box\_xy = prediction[:, :, :, :, :2]

box\_wh = prediction[:, :, :, :, 2:4]

box\_confidence = prediction[:, :, :, :, 4:5]

box\_probs = prediction[:, :, :, :, 5:]

box\_xy = (self.sigmoid(box\_xy) + grid) / P.Cast()(F.tuple\_to\_array((grid\_size[1], grid\_size[0])), ms.float32)

box\_wh = P.Exp()(box\_wh) \* self.anchors / self.input\_shape

box\_confidence = self.sigmoid(box\_confidence)

box\_probs = self.sigmoid(box\_probs)

if self.training:

return grid, prediction, box\_xy, box\_wh

return box\_xy, box\_wh, box\_confidence, box\_probs

class Iou(nn.Cell):

"""Calculate the iou of boxes."""

def \_\_init\_\_(self):

super(Iou, self).\_\_init\_\_()

self.min = P.Minimum()

self.max = P.Maximum()

def construct(self, box1, box2):

box1\_xy = box1[:, :, :, :, :, :2]

box1\_wh = box1[:, :, :, :, :, 2:4]

box1\_mins = box1\_xy - box1\_wh / F.scalar\_to\_array(2.0)

box1\_maxs = box1\_xy + box1\_wh / F.scalar\_to\_array(2.0)

box2\_xy = box2[:, :, :, :, :, :2]

box2\_wh = box2[:, :, :, :, :, 2:4]

box2\_mins = box2\_xy - box2\_wh / F.scalar\_to\_array(2.0)

box2\_maxs = box2\_xy + box2\_wh / F.scalar\_to\_array(2.0)

intersect\_mins = self.max(box1\_mins, box2\_mins)

intersect\_maxs = self.min(box1\_maxs, box2\_maxs)

intersect\_wh = self.max(intersect\_maxs - intersect\_mins, F.scalar\_to\_array(0.0))

intersect\_area = P.Squeeze(-1)(intersect\_wh[:, :, :, :, :, 0:1]) \* \

P.Squeeze(-1)(intersect\_wh[:, :, :, :, :, 1:2])

box1\_area = P.Squeeze(-1)(box1\_wh[:, :, :, :, :, 0:1]) \* P.Squeeze(-1)(box1\_wh[:, :, :, :, :, 1:2])

box2\_area = P.Squeeze(-1)(box2\_wh[:, :, :, :, :, 0:1]) \* P.Squeeze(-1)(box2\_wh[:, :, :, :, :, 1:2])

iou = intersect\_area / (box1\_area + box2\_area - intersect\_area)

return iou

class YoloLossBlock(nn.Cell):

"""

YOLOv3 Loss block cell. It will finally output loss of the scale.

Args:

scale (str): Three scale here, 's', 'm' and 'l'.

config (Class): The default config of YOLOv3.

Returns:

Tensor, loss of the scale.

Examples:

YoloLossBlock('l', ConfigYOLOV3ResNet18()).

"""

def \_\_init\_\_(self, scale, config):

super(YoloLossBlock, self).\_\_init\_\_()

self.config = config

if scale == 's':

idx = (0, 1, 2)

elif scale == 'm':

idx = (3, 4, 5)

elif scale == 'l':

idx = (6, 7, 8)

else:

raise KeyError("Invalid scale value for DetectionBlock")

self.anchors = Tensor([self.config.anchor\_scales[i] for i in idx], ms.float32)

self.ignore\_threshold = Tensor(self.config.ignore\_threshold, ms.float32)

self.concat = P.Concat(axis=-1)

self.iou = Iou()

self.cross\_entropy = P.SigmoidCrossEntropyWithLogits()

self.reduce\_sum = P.ReduceSum()

self.reduce\_max = P.ReduceMax(keep\_dims=False)

self.input\_shape = Tensor(tuple(config.img\_shape[::-1]), ms.float32)

def construct(self, grid, prediction, pred\_xy, pred\_wh, y\_true, gt\_box):

object\_mask = y\_true[:, :, :, :, 4:5]

class\_probs = y\_true[:, :, :, :, 5:]

grid\_shape = P.Shape()(prediction)[1:3]

grid\_shape = P.Cast()(F.tuple\_to\_array(grid\_shape[::-1]), ms.float32)

pred\_boxes = self.concat((pred\_xy, pred\_wh))

true\_xy = y\_true[:, :, :, :, :2] \* grid\_shape - grid

true\_wh = y\_true[:, :, :, :, 2:4]

true\_wh = P.Select()(P.Equal()(true\_wh, 0.0),

P.Fill()(P.DType()(true\_wh), P.Shape()(true\_wh), 1.0),

true\_wh)

true\_wh = P.Log()(true\_wh / self.anchors \* self.input\_shape)

box\_loss\_scale = 2 - y\_true[:, :, :, :, 2:3] \* y\_true[:, :, :, :, 3:4]

gt\_shape = P.Shape()(gt\_box)

gt\_box = P.Reshape()(gt\_box, (gt\_shape[0], 1, 1, 1, gt\_shape[1], gt\_shape[2]))

iou = self.iou(P.ExpandDims()(pred\_boxes, -2), gt\_box) # [batch, grid[0], grid[1], num\_anchor, num\_gt]

best\_iou = self.reduce\_max(iou, -1) # [batch, grid[0], grid[1], num\_anchor]

ignore\_mask = best\_iou < self.ignore\_threshold

ignore\_mask = P.Cast()(ignore\_mask, ms.float32)

ignore\_mask = P.ExpandDims()(ignore\_mask, -1)

ignore\_mask = F.stop\_gradient(ignore\_mask)

xy\_loss = object\_mask \* box\_loss\_scale \* self.cross\_entropy(prediction[:, :, :, :, :2], true\_xy)

wh\_loss = object\_mask \* box\_loss\_scale \* 0.5 \* P.Square()(true\_wh - prediction[:, :, :, :, 2:4])

confidence\_loss = self.cross\_entropy(prediction[:, :, :, :, 4:5], object\_mask)

confidence\_loss = object\_mask \* confidence\_loss + (1 - object\_mask) \* confidence\_loss \* ignore\_mask

class\_loss = object\_mask \* self.cross\_entropy(prediction[:, :, :, :, 5:], class\_probs)

# Get smooth loss

xy\_loss = self.reduce\_sum(xy\_loss, ())

wh\_loss = self.reduce\_sum(wh\_loss, ())

confidence\_loss = self.reduce\_sum(confidence\_loss, ())

class\_loss = self.reduce\_sum(class\_loss, ())

loss = xy\_loss + wh\_loss + confidence\_loss + class\_loss

return loss / P.Shape()(prediction)[0]

class yolov3\_resnet18(nn.Cell):

"""

ResNet based YOLOv3 network.

Args:

config (Class): YOLOv3 config.

Returns:

Cell, cell instance of ResNet based YOLOv3 neural network.

Examples:

yolov3\_resnet18(80, [1,3,416,416]).

"""

def \_\_init\_\_(self, config):

super(yolov3\_resnet18, self).\_\_init\_\_()

self.config = config

# YOLOv3 network

self.feature\_map = YOLOv3(feature\_shape=self.config.feature\_shape,

backbone=ResNet(BasicBlock,

self.config.backbone\_layers,

self.config.backbone\_input\_shape,

self.config.backbone\_shape,

self.config.backbone\_stride,

num\_classes=None),

backbone\_shape=self.config.backbone\_shape,

out\_channel=self.config.out\_channel)

# prediction on the default anchor boxes

self.detect\_1 = DetectionBlock('l', self.config)

self.detect\_2 = DetectionBlock('m', self.config)

self.detect\_3 = DetectionBlock('s', self.config)

def construct(self, x):

big\_object\_output, medium\_object\_output, small\_object\_output = self.feature\_map(x)

output\_big = self.detect\_1(big\_object\_output)

output\_me = self.detect\_2(medium\_object\_output)

output\_small = self.detect\_3(small\_object\_output)

return output\_big, output\_me, output\_small

class YoloWithLossCell(nn.Cell):

""""

Provide YOLOv3 training loss through network.

Args:

network (Cell): The training network.

config (Class): YOLOv3 config.

Returns:

Tensor, the loss of the network.

"""

def \_\_init\_\_(self, network, config):

super(YoloWithLossCell, self).\_\_init\_\_()

self.yolo\_network = network

self.config = config

self.loss\_big = YoloLossBlock('l', self.config)

self.loss\_me = YoloLossBlock('m', self.config)

self.loss\_small = YoloLossBlock('s', self.config)

def construct(self, x, y\_true\_0, y\_true\_1, y\_true\_2, gt\_0, gt\_1, gt\_2):

yolo\_out = self.yolo\_network(x)

loss\_l = self.loss\_big(yolo\_out[0][0], yolo\_out[0][1], yolo\_out[0][2], yolo\_out[0][3], y\_true\_0, gt\_0)

loss\_m = self.loss\_me(yolo\_out[1][0], yolo\_out[1][1], yolo\_out[1][2], yolo\_out[1][3], y\_true\_1, gt\_1)

loss\_s = self.loss\_small(yolo\_out[2][0], yolo\_out[2][1], yolo\_out[2][2], yolo\_out[2][3], y\_true\_2, gt\_2)

return loss\_l + loss\_m + loss\_s

class TrainingWrapper(nn.Cell):

"""

Encapsulation class of YOLOv3 network training.

Append an optimizer to the training network after that the construct

function can be called to create the backward graph.

Args:

network (Cell): The training network. Note that loss function should have been added.

optimizer (Optimizer): Optimizer for updating the weights.

sens (Number): The adjust parameter. Default: 1.0.

"""

def \_\_init\_\_(self, network, optimizer, sens=1.0):

super(TrainingWrapper, self).\_\_init\_\_(auto\_prefix=False)

self.network = network

self.network.set\_grad()

self.weights = ms.ParameterTuple(network.trainable\_params())

self.optimizer = optimizer

self.grad = C.GradOperation(get\_by\_list=True, sens\_param=True)

self.sens = sens

self.reducer\_flag = False

self.grad\_reducer = None

self.parallel\_mode = context.get\_auto\_parallel\_context("parallel\_mode")

if self.parallel\_mode in [ParallelMode.DATA\_PARALLEL, ParallelMode.HYBRID\_PARALLEL]:

self.reducer\_flag = True

if self.reducer\_flag:

mean = context.get\_auto\_parallel\_context("gradients\_mean")

if auto\_parallel\_context().get\_device\_num\_is\_set():

degree = context.get\_auto\_parallel\_context("device\_num")

else:

degree = get\_group\_size()

self.grad\_reducer = nn.DistributedGradReducer(optimizer.parameters, mean, degree)

def construct(self, \*args):

weights = self.weights

loss = self.network(\*args)

sens = P.Fill()(P.DType()(loss), P.Shape()(loss), self.sens)

grads = self.grad(self.network, weights)(\*args, sens)

if self.reducer\_flag:

# apply grad reducer on grads

grads = self.grad\_reducer(grads)

return F.depend(loss, self.optimizer(grads))

class YoloBoxScores(nn.Cell):

"""

Calculate the boxes of the original picture size and the score of each box.

Args:

config (Class): YOLOv3 config.

Returns:

Tensor, the boxes of the original picture size.

Tensor, the score of each box.

"""

def \_\_init\_\_(self, config):

super(YoloBoxScores, self).\_\_init\_\_()

self.input\_shape = Tensor(np.array(config.img\_shape), ms.float32)

self.num\_classes = config.num\_classes

def construct(self, box\_xy, box\_wh, box\_confidence, box\_probs, image\_shape):

batch\_size = F.shape(box\_xy)[0]

x = box\_xy[:, :, :, :, 0:1]

y = box\_xy[:, :, :, :, 1:2]

box\_yx = P.Concat(-1)((y, x))

w = box\_wh[:, :, :, :, 0:1]

h = box\_wh[:, :, :, :, 1:2]

box\_hw = P.Concat(-1)((h, w))

new\_shape = P.Round()(image\_shape \* P.ReduceMin()(self.input\_shape / image\_shape))

offset = (self.input\_shape - new\_shape) / 2.0 / self.input\_shape

scale = self.input\_shape / new\_shape

box\_yx = (box\_yx - offset) \* scale

box\_hw = box\_hw \* scale

box\_min = box\_yx - box\_hw / 2.0

box\_max = box\_yx + box\_hw / 2.0

boxes = P.Concat(-1)((box\_min[:, :, :, :, 0:1],

box\_min[:, :, :, :, 1:2],

box\_max[:, :, :, :, 0:1],

box\_max[:, :, :, :, 1:2]))

image\_scale = P.Tile()(image\_shape, (1, 2))

boxes = boxes \* image\_scale

boxes = F.reshape(boxes, (batch\_size, -1, 4))

boxes\_scores = box\_confidence \* box\_probs

boxes\_scores = F.reshape(boxes\_scores, (batch\_size, -1, self.num\_classes))

return boxes, boxes\_scores

class YoloWithEval(nn.Cell):

"""

Encapsulation class of YOLOv3 evaluation.

Args:

network (Cell): The training network. Note that loss function and optimizer must not be added.

config (Class): YOLOv3 config.

Returns:

Tensor, the boxes of the original picture size.

Tensor, the score of each box.

Tensor, the original picture size.

"""

def \_\_init\_\_(self, network, config):

super(YoloWithEval, self).\_\_init\_\_()

self.yolo\_network = network

self.box\_score\_0 = YoloBoxScores(config)

self.box\_score\_1 = YoloBoxScores(config)

self.box\_score\_2 = YoloBoxScores(config)

def construct(self, x, image\_shape):

yolo\_output = self.yolo\_network(x)

boxes\_0, boxes\_scores\_0 = self.box\_score\_0(\*yolo\_output[0], image\_shape)

boxes\_1, boxes\_scores\_1 = self.box\_score\_1(\*yolo\_output[1], image\_shape)

boxes\_2, boxes\_scores\_2 = self.box\_score\_2(\*yolo\_output[2], image\_shape)

boxes = P.Concat(1)((boxes\_0, boxes\_1, boxes\_2))

boxes\_scores = P.Concat(1)((boxes\_scores\_0, boxes\_scores\_1, boxes\_scores\_2))

return boxes, boxes\_scores, image\_shape

定义评价指标

以下代码定义了评价模型结果的相关指标，如：IOU等

def calc\_iou(bbox\_pred, bbox\_ground):

"""Calculate iou of predicted bbox and ground truth."""

x1 = bbox\_pred[0]

y1 = bbox\_pred[1]

width1 = bbox\_pred[2] - bbox\_pred[0]

height1 = bbox\_pred[3] - bbox\_pred[1]

x2 = bbox\_ground[0]

y2 = bbox\_ground[1]

width2 = bbox\_ground[2] - bbox\_ground[0]

height2 = bbox\_ground[3] - bbox\_ground[1]

endx = max(x1 + width1, x2 + width2)

startx = min(x1, x2)

width = width1 + width2 - (endx - startx)

endy = max(y1 + height1, y2 + height2)

starty = min(y1, y2)

height = height1 + height2 - (endy - starty)

if width <= 0 or height <= 0:

iou = 0

else:

area = width \* height

area1 = width1 \* height1

area2 = width2 \* height2

iou = area \* 1. / (area1 + area2 - area)

return iou

def apply\_nms(all\_boxes, all\_scores, thres, max\_boxes):

"""Apply NMS to bboxes."""

x1 = all\_boxes[:, 0]

y1 = all\_boxes[:, 1]

x2 = all\_boxes[:, 2]

y2 = all\_boxes[:, 3]

areas = (x2 - x1 + 1) \* (y2 - y1 + 1)

order = all\_scores.argsort()[::-1]

keep = []

while order.size > 0:

i = order[0]

keep.append(i)

if len(keep) >= max\_boxes:

break

xx1 = np.maximum(x1[i], x1[order[1:]])

yy1 = np.maximum(y1[i], y1[order[1:]])

xx2 = np.minimum(x2[i], x2[order[1:]])

yy2 = np.minimum(y2[i], y2[order[1:]])

w = np.maximum(0.0, xx2 - xx1 + 1)

h = np.maximum(0.0, yy2 - yy1 + 1)

inter = w \* h

ovr = inter / (areas[i] + areas[order[1:]] - inter)

inds = np.where(ovr <= thres)[0]

order = order[inds + 1]

return keep

def metrics(pred\_data):

"""Calculate precision and recall of predicted bboxes."""

config = ConfigYOLOV3ResNet18()

num\_classes = config.num\_classes

count\_corrects = [1e-6 for \_ in range(num\_classes)]

count\_grounds = [1e-6 for \_ in range(num\_classes)]

count\_preds = [1e-6 for \_ in range(num\_classes)]

for i, sample in enumerate(pred\_data):

gt\_anno = sample["annotation"]

box\_scores = sample['box\_scores']

boxes = sample['boxes']

mask = box\_scores >= config.obj\_threshold

boxes\_ = []

scores\_ = []

classes\_ = []

max\_boxes = config.nms\_max\_num

for c in range(num\_classes):

class\_boxes = np.reshape(boxes, [-1, 4])[np.reshape(mask[:, c], [-1])]

class\_box\_scores = np.reshape(box\_scores[:, c], [-1])[np.reshape(mask[:, c], [-1])]

nms\_index = apply\_nms(class\_boxes, class\_box\_scores, config.nms\_threshold, max\_boxes)

class\_boxes = class\_boxes[nms\_index]

class\_box\_scores = class\_box\_scores[nms\_index]

classes = np.ones\_like(class\_box\_scores, 'int32') \* c

boxes\_.append(class\_boxes)

scores\_.append(class\_box\_scores)

classes\_.append(classes)

boxes = np.concatenate(boxes\_, axis=0)

classes = np.concatenate(classes\_, axis=0)

# metric

count\_correct = [1e-6 for \_ in range(num\_classes)]

count\_ground = [1e-6 for \_ in range(num\_classes)]

count\_pred = [1e-6 for \_ in range(num\_classes)]

for anno in gt\_anno:

count\_ground[anno[4]] += 1

for box\_index, box in enumerate(boxes):

bbox\_pred = [box[1], box[0], box[3], box[2]]

count\_pred[classes[box\_index]] += 1

for anno in gt\_anno:

class\_ground = anno[4]

if classes[box\_index] == class\_ground:

iou = calc\_iou(bbox\_pred, anno)

if iou >= 0.5:

count\_correct[class\_ground] += 1

break

count\_corrects = [count\_corrects[i] + count\_correct[i] for i in range(num\_classes)]

count\_preds = [count\_preds[i] + count\_pred[i] for i in range(num\_classes)]

count\_grounds = [count\_grounds[i] + count\_ground[i] for i in range(num\_classes)]

precision = np.array([count\_corrects[ix] / count\_preds[ix] for ix in range(num\_classes)])

recall = np.array([count\_corrects[ix] / count\_grounds[ix] for ix in range(num\_classes)])

return precision, recall

定义相关超参数

这里通过定义一个类来定义所有超参数。

"""Config parameters for YOLOv3 models."""

class ConfigYOLOV3ResNet18:

"""

Config parameters for YOLOv3.

Examples:

ConfigYoloV3ResNet18.

"""

img\_shape = [352, 640]

feature\_shape = [32, 3, 352, 640]

num\_classes = 3

nms\_max\_num = 50

\_NUM\_BOXES = 50

backbone\_input\_shape = [64, 64, 128, 256]

backbone\_shape = [64, 128, 256, 512]

backbone\_layers = [2, 2, 2, 2]

backbone\_stride = [1, 2, 2, 2]

ignore\_threshold = 0.5

obj\_threshold = 0.3

nms\_threshold = 0.4

anchor\_scales = [(5,3),(10, 13), (16, 30),(33, 23),(30, 61),(62, 45),(59, 119),(116, 90),(156, 198)]

out\_channel = int(len(anchor\_scales) / 3 \* (num\_classes + 5))

定义训练网络的函数

######################## train YOLOv3 example ########################

import os

import argparse

import ast

from easydict import EasyDict as edict

import shutil

import numpy as np

import mindspore.nn as nn

from mindspore import context, Tensor

from mindspore.communication.management import init

from mindspore.train.callback import CheckpointConfig, ModelCheckpoint, LossMonitor, TimeMonitor

from mindspore.train import Model

from mindspore.context import ParallelMode

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.common.initializer import initializer

from mindspore.common import set\_seed

import sys

sys.path.insert(0,'./code/') #yours code path

#sys.path.insert(0,'./yolov3/code/')

from src.yolov3 import yolov3\_resnet18, YoloWithLossCell, TrainingWrapper

from src.dataset import create\_yolo\_dataset, data\_to\_mindrecord\_byte\_image

from src.config import ConfigYOLOV3ResNet18

import moxing as mox

set\_seed(1)

def get\_lr(learning\_rate, start\_step, global\_step, decay\_step, decay\_rate, steps=False):

"""Set learning rate."""

lr\_each\_step = []

for i in range(global\_step):

if steps:

lr\_each\_step.append(learning\_rate \* (decay\_rate \*\* (i // decay\_step)))

else:

lr\_each\_step.append(learning\_rate \* (decay\_rate \*\* (i / decay\_step)))

lr\_each\_step = np.array(lr\_each\_step).astype(np.float32)

lr\_each\_step = lr\_each\_step[start\_step:]

return lr\_each\_step

def init\_net\_param(network, init\_value='ones'):

"""Init the parameters in network."""

params = network.trainable\_params()

for p in params:

if isinstance(p.data, Tensor) and 'beta' not in p.name and 'gamma' not in p.name and 'bias' not in p.name:

p.set\_data(initializer(init\_value, p.data.shape, p.data.dtype))

def main(args\_opt):

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", device\_id=args\_opt.device\_id)

if args\_opt.distribute:

device\_num = args\_opt.device\_num

context.reset\_auto\_parallel\_context()

context.set\_auto\_parallel\_context(parallel\_mode=ParallelMode.DATA\_PARALLEL, gradients\_mean=True,

device\_num=device\_num)

init()

rank = args\_opt.device\_id % device\_num

else:

rank = 0

device\_num = 1

loss\_scale = float(args\_opt.loss\_scale)

# When create MindDataset, using the fitst mindrecord file, such as yolo.mindrecord0.

dataset = create\_yolo\_dataset(args\_opt.mindrecord\_file,

batch\_size=args\_opt.batch\_size, device\_num=device\_num, rank=rank)

dataset\_size = dataset.get\_dataset\_size()

print('The epoch size: ', dataset\_size)

print("Create dataset done!")

net = yolov3\_resnet18(ConfigYOLOV3ResNet18())

net = YoloWithLossCell(net, ConfigYOLOV3ResNet18())

init\_net\_param(net, "XavierUniform")

# checkpoint

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=dataset\_size \* args\_opt.save\_checkpoint\_epochs,

keep\_checkpoint\_max=args\_opt.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix="yolov3", directory=cfg.ckpt\_dir, config=ckpt\_config)

if args\_opt.pre\_trained:

if args\_opt.pre\_trained\_epoch\_size <= 0:

raise KeyError("pre\_trained\_epoch\_size must be greater than 0.")

param\_dict = load\_checkpoint(args\_opt.pre\_trained)

load\_param\_into\_net(net, param\_dict)

total\_epoch\_size = 60

if args\_opt.distribute:

total\_epoch\_size = 160

lr = Tensor(get\_lr(learning\_rate=args\_opt.lr, start\_step=args\_opt.pre\_trained\_epoch\_size \* dataset\_size,

global\_step=total\_epoch\_size \* dataset\_size,

decay\_step=1000, decay\_rate=0.95, steps=True))

opt = nn.Adam(filter(lambda x: x.requires\_grad, net.get\_parameters()), lr, loss\_scale=loss\_scale)

net = TrainingWrapper(net, opt, loss\_scale)

callback = [LossMonitor(10\*dataset\_size), ckpoint\_cb]

model = Model(net)

dataset\_sink\_mode = cfg.dataset\_sink\_mode

print("Start train YOLOv3, the first epoch will be slower because of the graph compilation.")

model.train(args\_opt.epoch\_size, dataset, callbacks=callback, dataset\_sink\_mode=dataset\_sink\_mode)

开始训练

在创建好OBS桶后，创建”/yolo/output”两级文件夹用于存放训练后的模型文件。如图所示，进入OBS控制台后，依次点击“对象”、“新建文件夹”

图形用户界面, 应用程序

描述已自动生成

OBS文件夹创建示例

注意：代码中"train\_url"参数需要将“unet4mindspore/unet/yolo/output”改成自己创建的桶内文件夹的路径，用于保存输出的模型。

# ------------yolov3 train -----------------------------

cfg = edict({

"distribute": False,

"device\_id": 0,

"device\_num": 1,

"dataset\_sink\_mode": True,

"lr": 0.001,

"epoch\_size": 60,

"batch\_size": 32,

"loss\_scale" : 1024,

"pre\_trained": None,

"pre\_trained\_epoch\_size":0,

"ckpt\_dir": "./ckpt",

"save\_checkpoint\_epochs" :1,

'keep\_checkpoint\_max': 1,

"train\_url": 'obs://unet4mindspore/unet/yolo/output', #将unet4mindspore修改为自己的桶名称

})

if os.path.exists(cfg.ckpt\_dir):

shutil.rmtree(cfg.ckpt\_dir)

data\_path = './data/'

# if not os.path.exists(data\_path):

# mox.file.copy\_parallel(src\_url=cfg.data\_url, dst\_url=data\_path)

mindrecord\_dir\_train = os.path.join(data\_path,'mindrecord/train')

print("Start create dataset!")

# It will generate mindrecord file in args\_opt.mindrecord\_dir,and the file name is yolo.mindrecord.

prefix = "yolo.mindrecord"

cfg.mindrecord\_file = os.path.join(mindrecord\_dir\_train, prefix)

if os.path.exists(mindrecord\_dir\_train):

shutil.rmtree(mindrecord\_dir\_train)

image\_dir = os.path.join(data\_path, "train")

if os.path.exists(mindrecord\_dir\_train) and os.listdir(mindrecord\_dir\_train):

print('The mindrecord file had exists!')

else:

image\_dir = os.path.join(data\_path, "train")

if not os.path.exists(mindrecord\_dir\_train):

os.makedirs(mindrecord\_dir\_train)

print("Create Mindrecord.")

data\_to\_mindrecord\_byte\_image(image\_dir, mindrecord\_dir\_train, prefix, 1)

print("Create Mindrecord Done, at {}".format(mindrecord\_dir\_train))

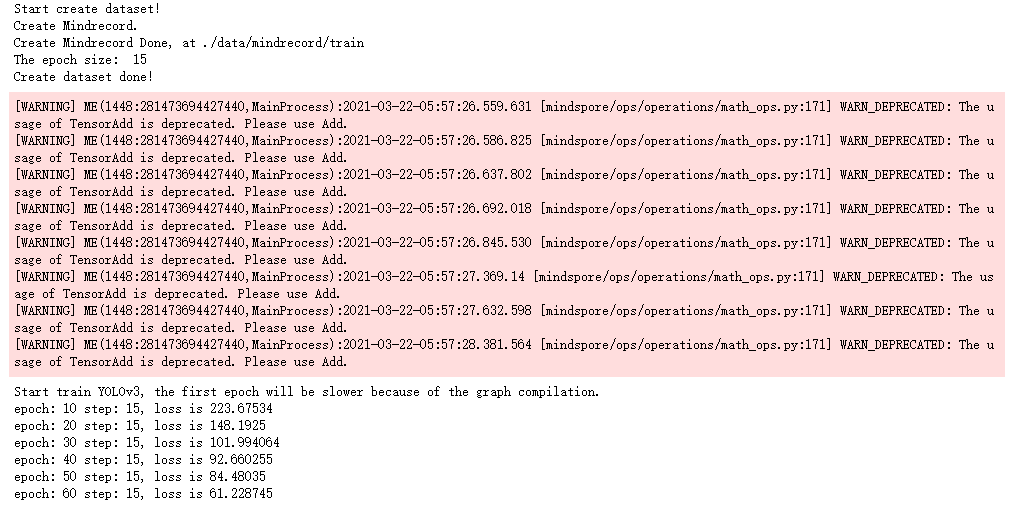
# if you need use mindrecord file next time, you can save them to yours obs.

#mox.file.copy\_parallel(src\_url=args\_opt.mindrecord\_dir\_train, dst\_url=os.path.join(cfg.data\_url,'mindspore/train')

#执行训练

main(cfg)

输出（WARNING不用管）：



同步保存模型文件到自己OBS桶内output路径下：

mox.file.copy\_parallel(src\_url=cfg.ckpt\_dir, dst\_url=cfg.train\_url)



测试网络模型

"""Test for yolov3-resnet18"""

import os

import argparse

import time

from easydict import EasyDict as edict

import matplotlib.pyplot as plt

from PIL import Image

import PIL

import numpy as np

import sys

#sys.path.insert(0,'./yolov3/code/')

sys.path.insert(0,'./code/') # yours code path

import moxing as mox

from mindspore import context, Tensor

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from src.yolov3 import yolov3\_resnet18, YoloWithEval

from src.dataset import create\_yolo\_dataset, data\_to\_mindrecord\_byte\_image

from src.config import ConfigYOLOV3ResNet18

from src.utils import metrics

def apply\_nms(all\_boxes, all\_scores, thres, max\_boxes):

"""Apply NMS to bboxes."""

x1 = all\_boxes[:, 0]

y1 = all\_boxes[:, 1]

x2 = all\_boxes[:, 2]

y2 = all\_boxes[:, 3]

areas = (x2 - x1 + 1) \* (y2 - y1 + 1)

order = all\_scores.argsort()[::-1]

keep = []

while order.size > 0:

i = order[0]

keep.append(i)

if len(keep) >= max\_boxes:

break

xx1 = np.maximum(x1[i], x1[order[1:]])

yy1 = np.maximum(y1[i], y1[order[1:]])

xx2 = np.minimum(x2[i], x2[order[1:]])

yy2 = np.minimum(y2[i], y2[order[1:]])

w = np.maximum(0.0, xx2 - xx1 + 1)

h = np.maximum(0.0, yy2 - yy1 + 1)

inter = w \* h

ovr = inter / (areas[i] + areas[order[1:]] - inter)

inds = np.where(ovr <= thres)[0]

order = order[inds + 1]

return keep

def tobox(boxes, box\_scores):

"""Calculate precision and recall of predicted bboxes."""

config = ConfigYOLOV3ResNet18()

num\_classes = config.num\_classes

mask = box\_scores >= config.obj\_threshold

boxes\_ = []

scores\_ = []

classes\_ = []

max\_boxes = config.nms\_max\_num

for c in range(num\_classes):

class\_boxes = np.reshape(boxes, [-1, 4])[np.reshape(mask[:, c], [-1])]

class\_box\_scores = np.reshape(box\_scores[:, c], [-1])[np.reshape(mask[:, c], [-1])]

nms\_index = apply\_nms(class\_boxes, class\_box\_scores, config.nms\_threshold, max\_boxes)

#nms\_index = apply\_nms(class\_boxes, class\_box\_scores, 0.5, max\_boxes)

class\_boxes = class\_boxes[nms\_index]

class\_box\_scores = class\_box\_scores[nms\_index]

classes = np.ones\_like(class\_box\_scores, 'int32') \* c

boxes\_.append(class\_boxes)

scores\_.append(class\_box\_scores)

classes\_.append(classes)

boxes = np.concatenate(boxes\_, axis=0)

classes = np.concatenate(classes\_, axis=0)

scores = np.concatenate(scores\_, axis=0)

return boxes, classes, scores

def yolo\_eval(cfg):

"""Yolov3 evaluation."""

ds = create\_yolo\_dataset(cfg.mindrecord\_file, batch\_size=1, is\_training=False)

config = ConfigYOLOV3ResNet18()

net = yolov3\_resnet18(config)

eval\_net = YoloWithEval(net, config)

print("Load Checkpoint!")

param\_dict = load\_checkpoint(cfg.ckpt\_path)

load\_param\_into\_net(net, param\_dict)

eval\_net.set\_train(False)

i = 1.

total = ds.get\_dataset\_size()

start = time.time()

pred\_data = []

print("\n========================================\n")

print("total images num: ", total)

print("Processing, please wait a moment.")

num\_class={0:'person', 1: 'face', 2:'mask'}

for data in ds.create\_dict\_iterator(output\_numpy=True):

img\_np = data['image']

image\_shape = data['image\_shape']

#print(image\_shape)

annotation = data['annotation']

image\_file = data['file']

image\_file = image\_file.tostring().decode('ascii')

eval\_net.set\_train(False)

output = eval\_net(Tensor(img\_np), Tensor(image\_shape))

for batch\_idx in range(img\_np.shape[0]):

boxes = output[0].asnumpy()[batch\_idx]

box\_scores = output[1].asnumpy()[batch\_idx]

image = img\_np[batch\_idx,...]

boxes, classes, scores =tobox(boxes, box\_scores)

#print(classes)

#print(scores)

fig = plt.figure() #相当于创建画板

ax = fig.add\_subplot(1,1,1) #创建子图，相当于在画板中添加一个画纸，当然可创建多个画纸，具体由其中参数而定

image\_path = os.path.join(cfg.image\_dir, image\_file)

f = Image.open(image\_path)

img\_np = np.asarray(f ,dtype=np.float32) #H，W，C格式

ax.imshow(img\_np.astype(np.uint8)) #当前画纸中画一个图片

for box\_index in range(boxes.shape[0]):

ymin=boxes[box\_index][0]

xmin=boxes[box\_index][1]

ymax=boxes[box\_index][2]

xmax=boxes[box\_index][3]

#print(xmin,ymin,xmax,ymax)

#添加方框，(xmin,ymin)表示左顶点坐标，(xmax-xmin),(ymax-ymin)表示方框长宽

ax.add\_patch(plt.Rectangle((xmin,ymin),(xmax-xmin),(ymax-ymin),fill=False,edgecolor='red', linewidth=2))

#给方框加标注，xmin,ymin表示x,y坐标，其它相当于画笔属性

ax.text(xmin,ymin,s = str(num\_class[classes[box\_index]])+str(scores[box\_index]),

style='italic',bbox={'facecolor': 'blue', 'alpha': 0.5, 'pad': 0})

plt.show()

注意：这里的"ckpt\_url"为步骤6中的"train\_url"地址，即OBS桶内保存模型文件的地址；

而"test\_url"地址需在OBS中新创建文件夹，用于保存测试输出。本例中新创建路径为”yolo/testoutput”:

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

创建测试输出

# ---------------yolov3 test-------------------------

cfg = edict({

"ckpt\_url": 'obs://unet4mindspore/unet/yolo/output', #将unet4mindspore修改为自己的桶名称

"test\_url": 'obs://unet4mindspore/unet/yolo/testoutput' #将unet4mindspore修改为自己的桶名称

})

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

ckpt\_path = './ckpt/'

if not os.path.exists(ckpt\_path):

mox.file.copy\_parallel(src\_url=args\_opt.ckpt\_url, dst\_url=ckpt\_path)

cfg.ckpt\_path = os.path.join(ckpt\_path, "yolov3-60\_15.ckpt")

data\_path = './data/'

if not os.path.exists(data\_path):

mox.file.copy\_parallel(src\_url=data\_url, dst\_url=data\_path)

mindrecord\_dir\_test = os.path.join(data\_path,'mindrecord/test')

prefix = "yolo.mindrecord"

cfg.mindrecord\_file = os.path.join(mindrecord\_dir\_test, prefix)

cfg.image\_dir = os.path.join(data\_path, "test")

if os.path.exists(mindrecord\_dir\_test) and os.listdir(mindrecord\_dir\_test):

print('The mindrecord file had exists!')

else:

if not os.path.isdir(mindrecord\_dir\_test):

os.makedirs(mindrecord\_dir\_test)

prefix = "yolo.mindrecord"

cfg.mindrecord\_file = os.path.join(mindrecord\_dir\_test, prefix)

print("Create Mindrecord.")

data\_to\_mindrecord\_byte\_image(cfg.image\_dir, mindrecord\_dir\_test, prefix, 1)

print("Create Mindrecord Done, at {}".format(mindrecord\_dir\_test))

# if you need use mindrecord file next time, you can save them to yours obs.

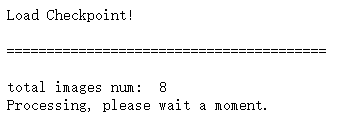
#mox.file.copy\_parallel(src\_url=args\_opt.mindrecord\_dir\_test, dst\_url=os.path.join(cfg.data\_url,'mindspore/test')

print("Start Eval!")

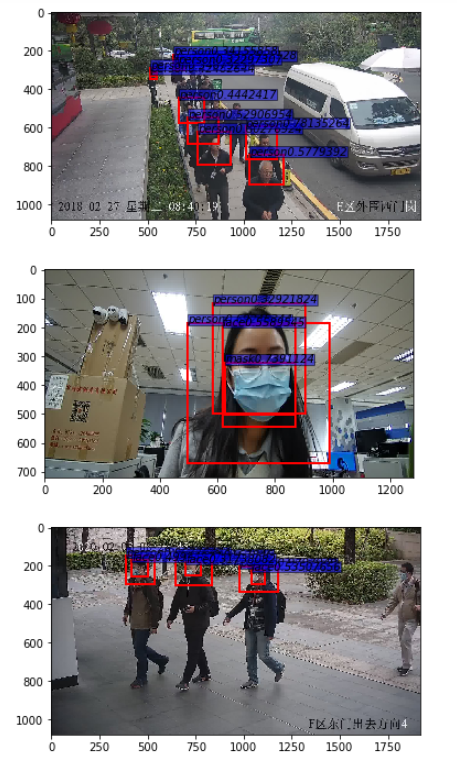
yolo\_eval(cfg)

输出（WARNING不用管）：





输出截图



测试图片效果

## 实验总结

本实验主要介绍如何使用MindSpore在利用yolov3网络模型实现目标检测任务。通过本实验学员将了解如何使用MindSpore深度学习框架实现yolov3目标检测网络模型的开发过程，通过基于该框架的训练和推理过程，进一步增加实践能力。

# 附录：ModelArts开发环境搭建

在华为云ModelArts平台上创建AI框架为Mindspore-1.7，硬件环境为Ascend 910+ARM的开发环境。

进入华为云ModelArts控制台

在[华为云ModelArts主页](https://www.huaweicloud.com/product/modelarts.html)，点击“管理控制台”进入ModelArts的管理页面。

图形用户界面, 文本, 应用程序

描述已自动生成

华为云ModelArts主页

创建Notebook训练作业

控制台区域选择“华北-北京四”，在左侧菜单栏中选择“开发环境”的“Notebook”，点击进入Notebook创建页面。



ModelArts控制台

点击“创建”按钮，创建一个新的Notebook，其配置如下：

名称：自定义。

自动停止：建议1小时。

镜像：Ascend+ARM算法开发和训练基础镜像。

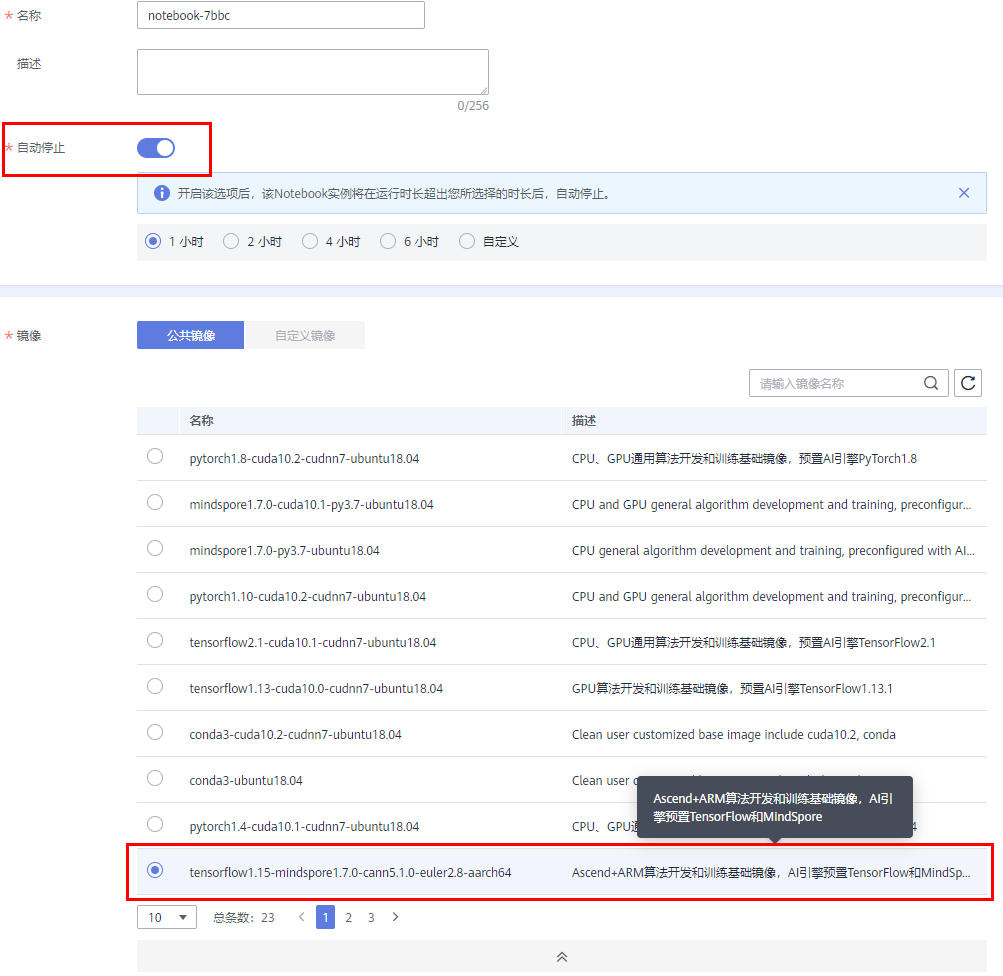
资源池：公共资源池。

类型：ASCEND。

规格：Ascend: 1\*Ascend910|CPU: 24核 96GB。

存储配置：默认存储（50GB），亦可选择EVS，支持自定义存储规格且为专属资源。

如图所示：





Notebook创建配置

配置完成之后“立即创建”，规格确认无误之后“提交”。

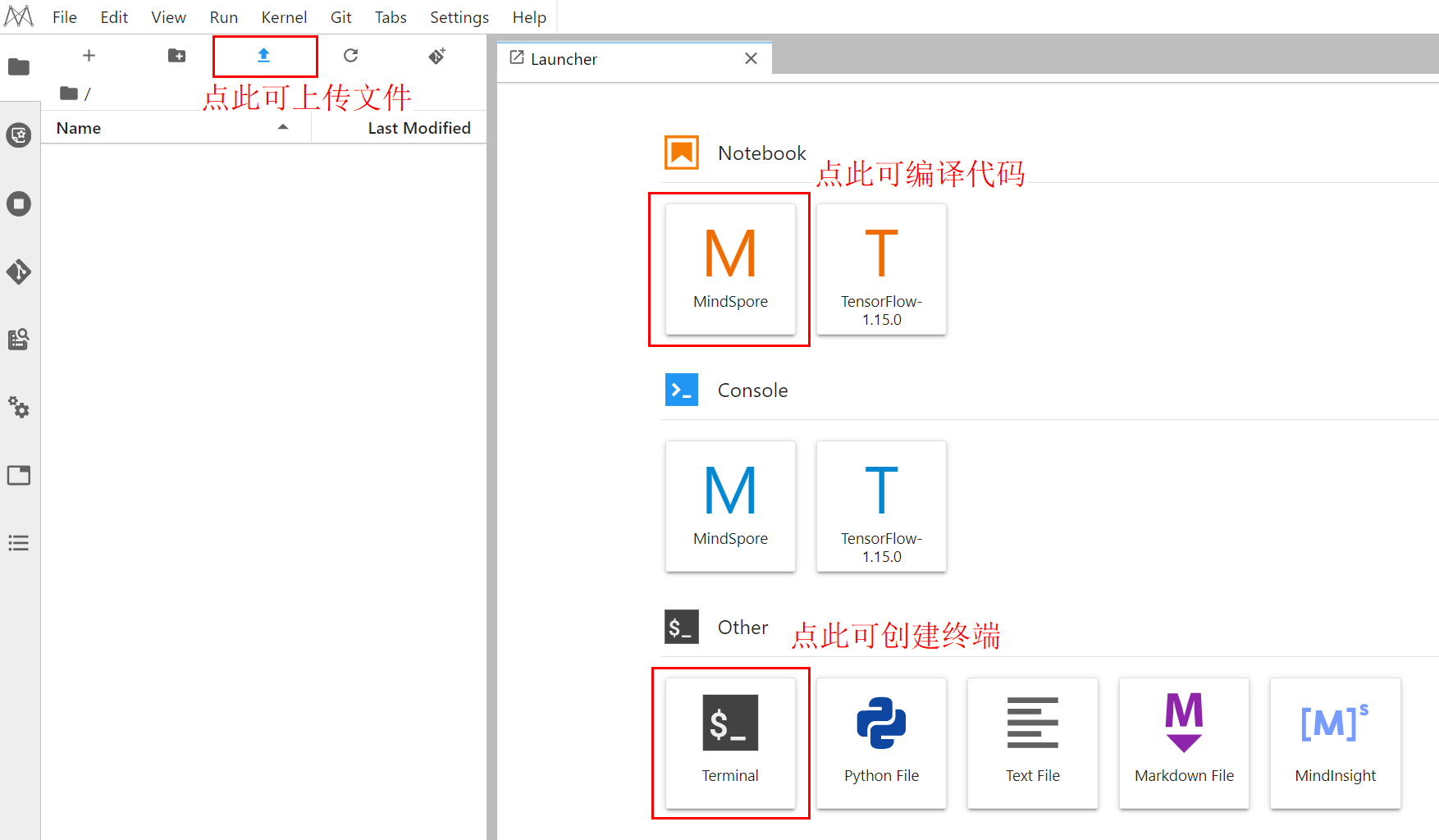
启动Notebook进入开发环境

当上一步创建好Notebook状态显示为“运行中”时，在右侧“操作”栏中“打开”，即可进入在线编程页面。



打开Notebook环境

可以在此页面创建或编辑MindSpore的项目，如图所示：



Notebook开发页面

\*注意：Notebook环境内上传、创建和编辑的文件均在/home/ma-user/work目录下。

停止Notebook训练作业

实验完成之后，请及时关闭Notebook训练作业，避免产生不必要的资源浪费。

登录[华为云ModelArts控制台](https://console.huaweicloud.com/modelarts/?region=cn-north-4" \l "/dev-container)，在“操作”栏选择“停止”操作。

如下图所示：



及时停止Notebook

至此训练用的线上Notebook环境搭建完成。