**实验（实习）报告**

1. 实验目的

掌握卷积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。

1. 实验任务

1）初级实验：①花卉图像分类卷积网络训练验证实验；

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

3）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战；⑥前沿网络案例Yolov3/ Deeplabv3。

1. 实验步骤
2. 花卉图像分类实验
   1. 导入实验环境

步骤 1 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")

步骤 2 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

})

* 1. 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。

步骤 1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

步骤 2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)

步骤 3划分训练集与测试集

#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。

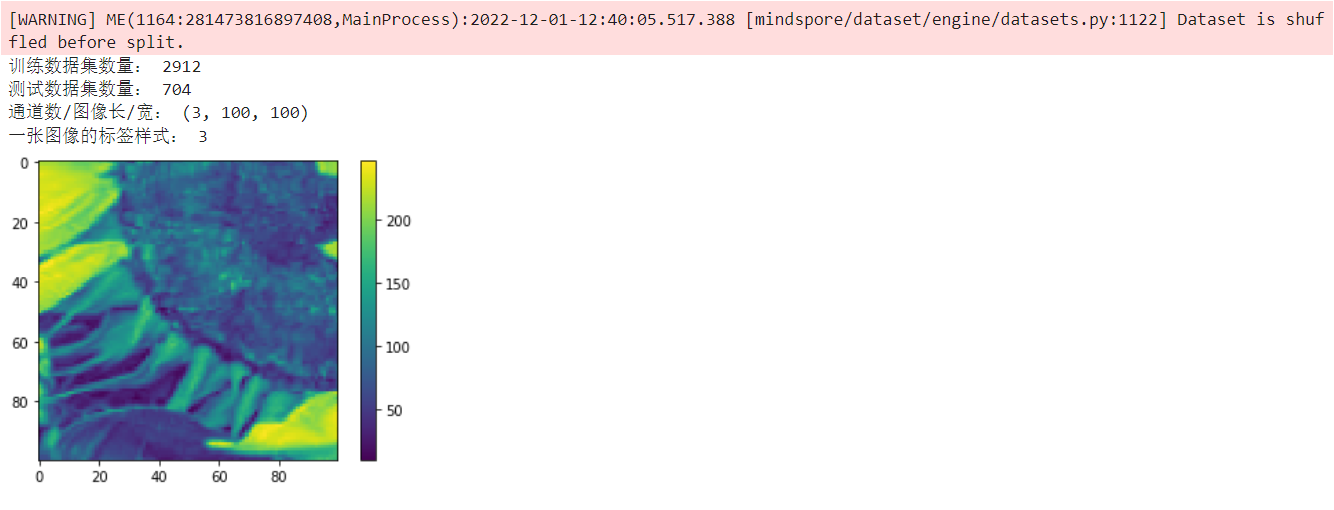
plt.figure()

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

plt.colorbar()

plt.grid(False)

plt.show()



* 1. 构建CNN图像识别模型

步骤 1 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

步骤 2 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)

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

epoch: 10 step: 91, loss is 1.1753120422363281

epoch: 20 step: 91, loss is 1.0859466791152954

epoch: 30 step: 91, loss is 0.5677516460418701

epoch: 40 step: 91, loss is 0.9325251579284668

epoch: 50 step: 91, loss is 0.8652545213699341

epoch: 60 step: 91, loss is 0.6390314102172852

epoch: 70 step: 91, loss is 0.7013850212097168

epoch: 80 step: 91, loss is 0.6687897443771362

epoch: 90 step: 91, loss is 0.603812575340271

epoch: 100 step: 91, loss is 0.6513016223907471

epoch: 110 step: 91, loss is 0.33892685174942017

epoch: 120 step: 91, loss is 0.5725375413894653

epoch: 130 step: 91, loss is 0.3894873857498169

epoch: 140 step: 91, loss is 0.3057480454444885

epoch: 150 step: 91, loss is 0.6159017086029053

epoch: 160 step: 91, loss is 0.3429974913597107

epoch: 170 step: 91, loss is 0.4333230257034302

epoch: 180 step: 91, loss is 0.546169638633728

epoch: 190 step: 91, loss is 0.3887840509414673

epoch: 200 step: 91, loss is 0.36199477314949036

epoch: 210 step: 91, loss is 0.4250497817993164

epoch: 220 step: 91, loss is 0.37851858139038086

epoch: 230 step: 91, loss is 0.3388608694076538

epoch: 240 step: 91, loss is 0.6281015276908875

epoch: 250 step: 91, loss is 0.5210052728652954

epoch: 260 step: 91, loss is 0.3364684283733368

epoch: 270 step: 91, loss is 0.2803938388824463

epoch: 280 step: 91, loss is 0.3913695216178894

epoch: 290 step: 91, loss is 0.8158993124961853

epoch: 300 step: 91, loss is 0.22055236995220184

epoch: 310 step: 91, loss is 0.3294251561164856

epoch: 320 step: 91, loss is 0.48163872957229614

epoch: 330 step: 91, loss is 0.1272602677345276

epoch: 340 step: 91, loss is 0.229139044880867

epoch: 350 step: 91, loss is 0.2894950211048126

epoch: 360 step: 91, loss is 0.19915755093097687

epoch: 370 step: 91, loss is 0.19907592236995697

epoch: 380 step: 91, loss is 0.42353999614715576

epoch: 390 step: 91, loss is 0.14113196730613708

epoch: 400 step: 91, loss is 0.22388432919979095

{'acc': 0.9176136363636364}

* 1. 图像分类模型验证

步骤 1 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)

步骤 2 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()

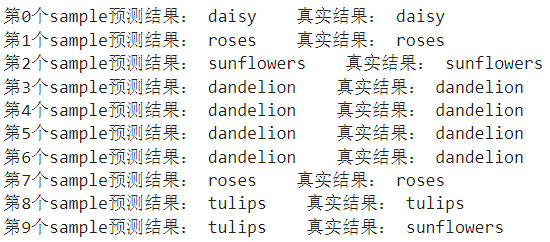
#显示预测结果

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]])



1. 正则化实验

2.1 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')

2.2 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

2.3 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

2.4 建立网络

2.4.1 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

2.4.2 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)

2.5 启动训练

2.5.1 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)

2.5.2 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\_fc = fc\_predict.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'})

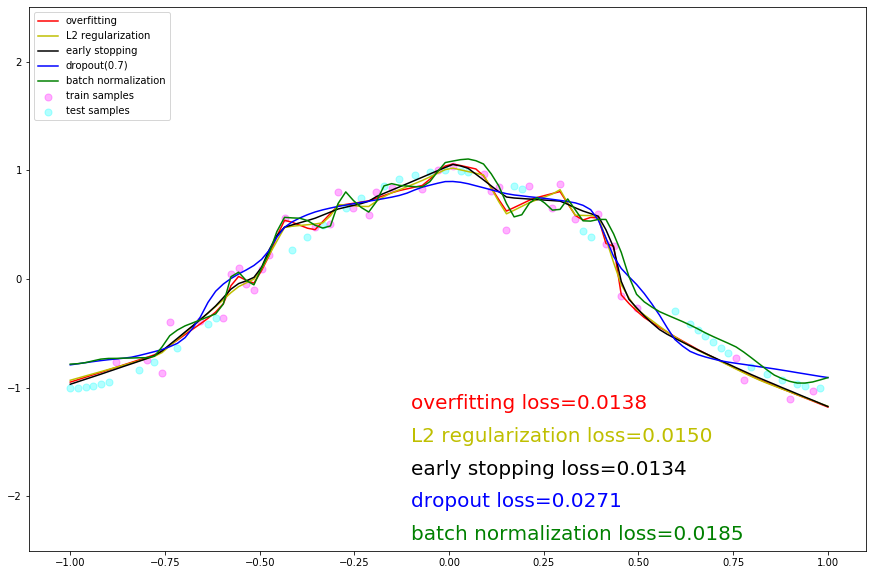
        plt.text(-0.1, -2.4, 'batch normalization loss=%.4f' % loss\_bn.asnumpy(), fontdict={'size': 20, 'color': 'green'})

        plt.legend(loc='upper left');

        plt.ylim((-2.5, 2.5));

        clear\_output(wait=True)

        plt.show()



1. 基于Lenet的手写数字识别

3.1 导入实验环境

import os

import numpy as np

import mindspore as ms

import mindspore.context as context

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

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

from mindspore import nn

from mindspore.train import Model

from mindspore.train.callback import LossMonitor

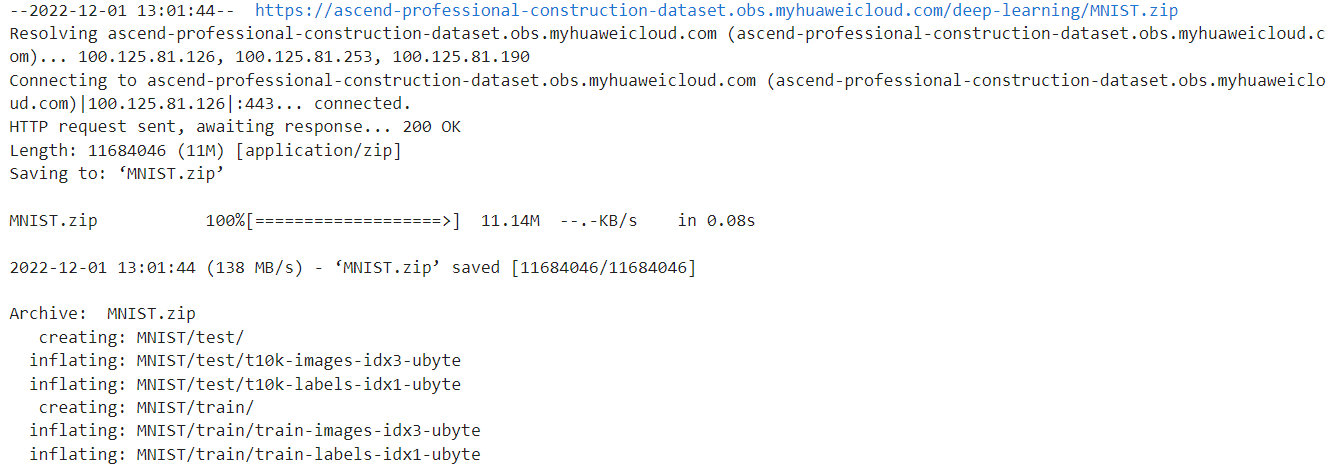
context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend') # Ascend, CPU, GPU

3.2 数据处理

步骤 1获取数据集

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

!unzip MNIST.zip



步骤 2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)

    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

步骤 3 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()

for i in range(1, 5):

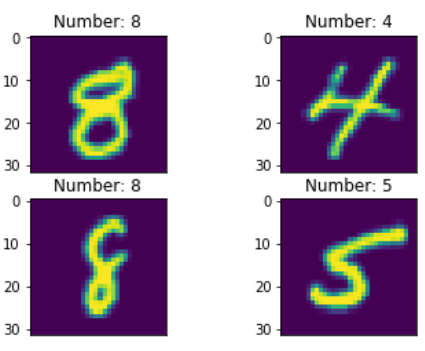
    plt.subplot(2, 2, i)

    plt.imshow(images[i][0])

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

    plt.xticks([])

plt.show()



3.3 定义模型

class LeNet5(nn.Cell):

    def \_\_init\_\_(self):

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

        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)

    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

3.4 训练

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()

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

    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'})

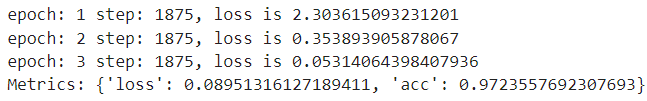
    # dataset\_sink\_mode can be True when using Ascend

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

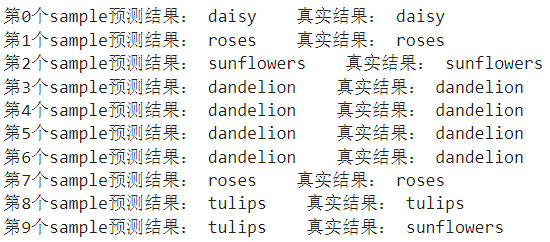
    metrics = model.eval(ds\_eval, dataset\_sink\_mode=True)

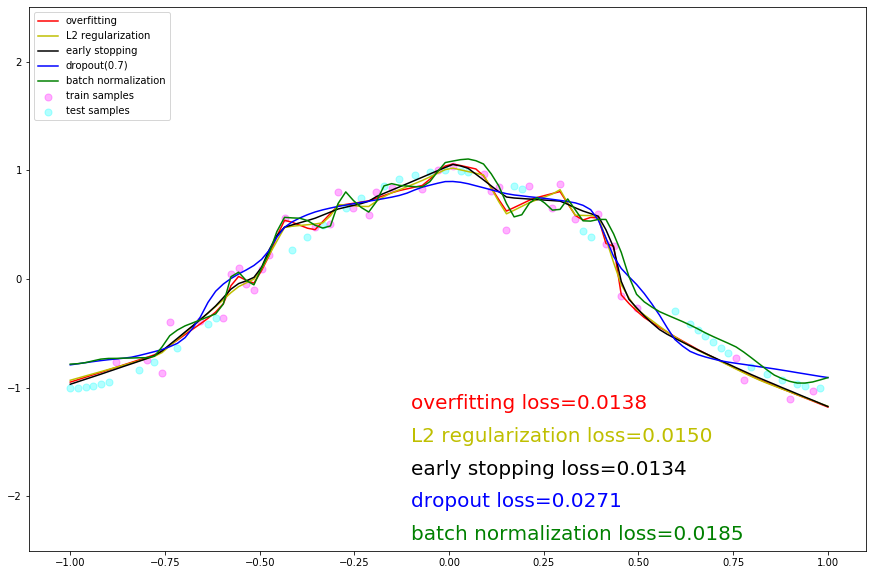
print('Metrics:', metrics)

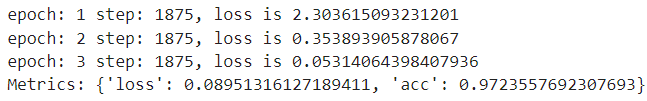
train('MNIST')



1. 实验结果







1. 实验总结

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。

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：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；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；带有Early Stop的模型处于折中的状态。

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。