**《人工智能》实验安排（2023）**

## 1.简介

本学期人工智能实验采用ModelArts，MindSpore开发框架来进行开发，Mindspore实验过程参考附录，实验的主要内容关注机器学习、神经网络、深度学习。

## 2.实验与评价形式

实验以分组形式完成，每组不超过6人，每个人均需完成所有实验内容，但可取最好的实验结果作为本组的最后成绩。实验完成后将组织集体汇报，以组为单位汇报实验内容与结果，同时统计每组最终成绩，公布排行榜。每位同学在每个实验部分的得分按照其个人在组内排名和整组排名综合计算。

## 3.提交材料

实验结束后需提交材料包括：

（1）以组为单位提交汇报PPT。PPT内容包括但不限于：

* 实验过程，介绍如何使用ModelArts，MindSpore建立实验环境，完成实验内容；
* 数据集简介，包括数据内容、特征数量、样本总量、训练集测试集大小、特征分析、数据缺失情况（如果有）、数据异常情况（如果有）、数据清洗过程（如果有）等；
* 算法、模型介绍，使用什么算法或模型，有无改进，如何改等；
* 实验结果，给出测试方案对于回归问题给出R2、均方根误差（RMSE）；对于分类问题给出混淆矩阵、召回率、准确率等（本组最好结果）；给出模型训练时长和模型文件大小
* 进一步改进的设想

（2）每人提交实验报告，基本内容要求与PPT一致，实验结果为自己的实验最终结果。

## 4.实验内容

（1）回归分析

i.分析新冠肺炎全国和地区病例数据，利用ModelArts平台上开发环境中的Notebook自编代码，建立合适的回归模型进行回归分析，建立疾病发展情况的趋势模型，并对指定日期之后的数据进行预测。

ii. 前列腺癌预测，利用ModelArts平台上开发环境中的Notebook自编代码，对前列腺癌数据进行回归分析，该数据集已分出训练和测试样例，请按统一的测试集进行测试。

（2）手写数字识别

i.利用ModelArts上的Minist手写数字集，采用AI开发框架Mindspore实现训练和测试手写数字识别。对比平方差损失和交叉熵损失，Mini-batch和no batch，Relu和Sigmoid，有dropout和无dropout的实验结果（详见课件要求），同样请统一训练集与测试集。

ii. 将上述模型稍加修改，迁移到美国邮政编码手写数字集的数字识别上，检测识别率并进行改进。

iii. 试试用前期遗传算法生成的手写数字图像，看看效果如何。

（3）乳腺癌识别

该问题评分按A，B，C三个级别进行，C为基础版，B为进阶版，A为超级版，C的最高分为90，B的最高分为100，A的最高分可超出100达到120。每个小组按任务级别不同分别比较不同评价指标。三个级别要求如下：

C. 完成图像级别的分类任务，用ModelArts的标签工具对图像进行分类标注，训练模型完成良性、恶性和正常B超影像的分类识别任务。评价指标为每类数据的F1值；

B. 完成肿瘤检测任务，用ModelArts的标签工具对图像进行检测框标注，不仅要完成良性、恶性等的识别，还要求用检测框定位出肿瘤位置。评价指标为分类的F1和检测框的mIOU（均交并比，良性、恶性两类检测框的IOU均值）；

A. 完成肿瘤分割任务，利用数据集提供的MASK图像提取分割标签，不仅要完成良性、恶性等的识别，还需要分割出肿瘤轮廓。评价指标为分类的F1和MASK的mIOU；

**注意：完成以上实验后，每组可自选一个实验部署到华为鲲鹏云主机上运行，附录2中有华为鲲鹏云上搭建python环境的参考介绍。**

## 5.华为云使用注意事项

实验在华为云上进行，华为公司为同学们提供代金券以支付服务器等的租赁费用，相关注意事项包括：

1. 使用前需注册华为云账号，并进行实名认证，签署开发者协议，领取代金券，具体操作见附录1；
2. 代金券可用于服务器租赁等的支出，注意租赁服务器时选择按需计费的方式，不要包月；
3. 由于云端资源只要占用就会收费，不管是计算还是存储，所以大家在使用云服务器进行模型训练，完成后应该主动停止，存储资源在不用了后也应主动释放，不要以为自己关机退出了就是停止计费了。存储资源收费较低，可以在确实不用了后再释放。计算资源根据自己的需要选择服务器，不要一味追求高性能，否则费用可能不够，训练完成后就主动关闭；
4. 一旦出现账户欠费情况，请及时与老师联系，交由后台处理，若自己充值了则无法退还；

附录1：

基于Mindspore的手写数字识别实验指南

## 实验介绍

Mindspore是最佳匹配昇腾AI处理器算力的全场景深度学习框架，为数据科学家和算法工程师提供设计友好、运行高效的开发体验，推动人工智能软硬件应用生态繁荣发展。

本章将通过手写数字识别的案例讲述华为AI开发框架Mindspore的基本使用。

## 环境准备

本实验在华为云ModelArts上进行，开发环境快速搭建可参考文末附录：ModelArts开发环境搭建。

## 实验过程

实验流程：

1、处理需要的数据集，这里使用了MNIST数据集。

2、定义一个网络，这里我们使用LeNet网络。

3、定义损失函数和优化器。

4、加载数据集并进行训练，训练完成后，查看结果及保存模型文件。

5、加载保存的模型，进行推理。

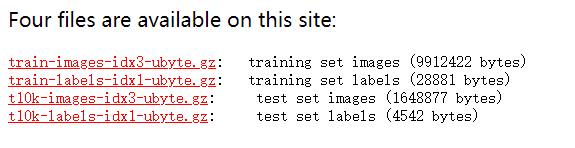
6、验证模型，加载测试数据集和训练后的模型，验证结果精度。

## 手写数字识别

获取数据集

实验数据链接：<http://yann.lecun.com/exdb/mnist/>

打开网页后将训练数据集和测试数据集下载，如下图所示。



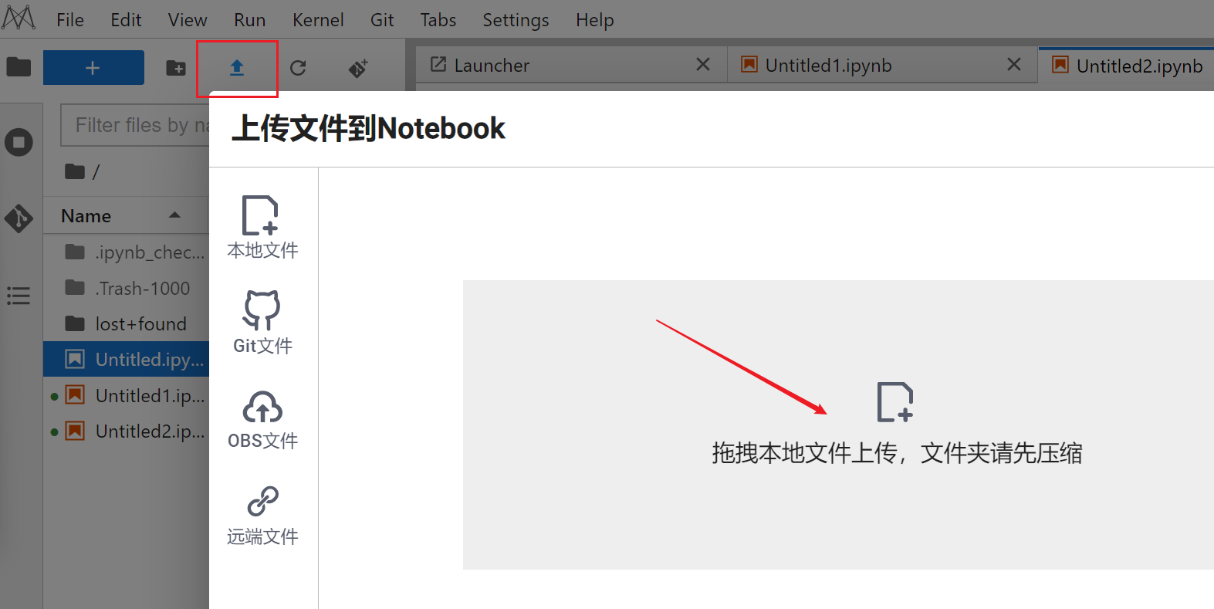
创建MNIST\_Data文件夹，MNIST\_Data文件夹下设置train和test文件夹，将下载的训练集和测试集数据**解压后**分别置于train和test文件夹下：

* 训练集train：train-images-idx3-ubyte，train-labels-idx1-ubyte
* 测试集test：t10k-images-idx3-ubyte，t10k-labels-idx1-ubyte

图形用户界面, 应用程序

描述已自动生成

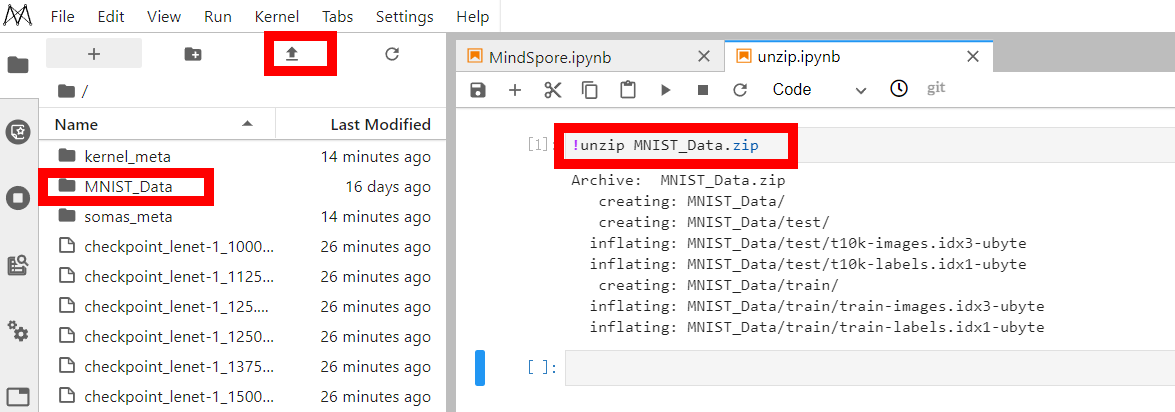
将MNIST\_Data文件夹压缩，上传至ModelArts实验平台：



MNIST\_Data压缩文件夹上传后使用unzip命令解压，具体命令：

!unzip MNIST\_Data.zip

如下图所示：



处理MNIST数据集

由于我们后面会采用LeNet这样的卷积神经网络对数据集进行训练，而采用LeNet在训练数据时，对数据格式是有所要求的，所以接下来的工作需要我们先查看数据集内的数据是什么样的，这样才能构造一个针对性的数据转换函数，将数据集数据转换成符合训练要求的数据形式。

查看原始数据集数据：

from mindspore import context

import matplotlib.pyplot as plt

import matplotlib

import numpy as np

import mindspore.dataset as ds

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend") # Windows version, set to use CPU for graph calculation

train\_data\_path = "./MNIST\_Data/train"

test\_data\_path = "./MNIST\_Data/test"

mnist\_ds = ds.MnistDataset(train\_data\_path) # Load training dataset

print('The type of mnist\_ds:', type(mnist\_ds))

print("Number of pictures contained in the mnist\_ds：",mnist\_ds.get\_dataset\_size()) # 60000 pictures in total

dic\_ds = mnist\_ds.create\_dict\_iterator() # Convert dataset to dictionary type

item = dic\_ds.\_\_next\_\_()

img = item["image"].asnumpy()

label = item["label"].asnumpy()

print("The item of mnist\_ds:", item.keys()) # Take a single data to view the data structure, including two keys, image and label

print("Tensor of image in item:", img.shape) # View the tensor of image (28,28,1)

print("The label of item:", label)

plt.imshow(np.squeeze(img))

plt.title("number:%s"% item["label"])

plt.show()

输出：

图形用户界面, 应用程序

描述已自动生成

从上面的运行情况我们可以看到,训练数据集train-images-idx3-ubyte和train-labels-idx1-ubyte对应的是6万张图片和6万个数字下标；载入数据后经过create\_dict\_iterator()转换字典型的数据集，取其中的一个数据查看，这是一个key为image和label的字典，其中image的张量形状为高度28、宽度28、通道1，label为对应图片的数字。

数据处理

定义数据集及数据操作

我们定义一个函数create\_dataset()来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

1、定义数据集。

2、定义进行数据增强和处理所需要的一些参数。

3、根据参数，生成对应的数据增强操作。

4、使用map()映射函数，将数据操作应用到数据集。

5、对生成的数据集进行处理。

# Data processing module

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

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

from mindspore.dataset.vision import Inter

from mindspore.common import dtype as mstype

def create\_dataset(data\_path, batch\_size=32, repeat\_size=1,

num\_parallel\_workers=1):

""" create dataset for train or test

Args:

data\_path: Data path

batch\_size: The number of data records in each group

repeat\_size: The number of replicated data records

num\_parallel\_workers: The number of parallel workers

"""

# define dataset

mnist\_ds = ds.MnistDataset(data\_path)

# Define some parameters needed for data enhancement and rough justification

resize\_height, resize\_width = 32, 32

rescale = 1.0 / 255.0

shift = 0.0

rescale\_nml = 1 / 0.3081

shift\_nml = -1 \* 0.1307 / 0.3081

# According to the parameters, generate the corresponding data enhancement method

resize\_op = CV.Resize((resize\_height, resize\_width), interpolation=Inter.LINEAR) # Resize images to (32, 32) by bilinear interpolation

rescale\_nml\_op = CV.Rescale(rescale\_nml, shift\_nml) # normalize images

rescale\_op = CV.Rescale(rescale, shift) # rescale images

hwc2chw\_op = CV.HWC2CHW() # change shape from (height, width, channel) to (channel, height, width) to fit network.

type\_cast\_op = C.TypeCast(mstype.int32) # change data type of label to int32 to fit network

# Using map () to apply operations to a dataset

mnist\_ds = mnist\_ds.map(input\_columns="label", operations=type\_cast\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=resize\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=rescale\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=rescale\_nml\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=hwc2chw\_op, num\_parallel\_workers=num\_parallel\_workers)

# Process the generated dataset

buffer\_size = 10000

mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size) # 10000 as in LeNet train script

mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=True)

mnist\_ds = mnist\_ds.repeat(repeat\_size)

return mnist\_ds

* batch\_size：每组包含的数据个数，现设置每组包含32个数据。
* repeat\_size：数据集复制的数量。

先进行shuffle、batch操作，再进行repeat操作。

查看数据集内包含多少组数据。

datas = create\_dataset(train\_data\_path) # Process the train dataset

print('Number of groups in the dataset:', datas.get\_dataset\_size()) # Number of query dataset groups

输出：

Number of groups in the dataset: 1875

取出其中一组数据，查看包含的key，图片数据的张量，以及下标labels的值。

data = datas.create\_dict\_iterator().\_\_next\_\_() # Take a set of datasets

print(data.keys())

images = data["image"].asnumpy() # Take out the image data in this dataset

labels = data["label"].asnumpy() # Take out the label (subscript) of this data set

print('Tensor of image:', images.shape) # Query the tensor of images in each dataset (32,1,32,32)

print('labels:', labels)

输出：

dict\_keys(['label', 'image'])

Tensor of image: (32, 1, 32, 32)

labels: [8 2 8 5 1 0 0 2 6 0 7 6 7 9 3 3 8 9 5 6 6 5 5 5 9 5 1 8 9 7 2 7]

查看image的图像和下标对应的值。

count = 1

for i in images:

plt.subplot(4, 8, count)

plt.imshow(np.squeeze(i))

plt.title('num:%s'%labels[count-1])

plt.xticks([])

count += 1

plt.axis("off")

plt.show() # Print a total of 32 pictures in the group

输出：

图片包含 文本

描述已自动生成

构造神经网络

在对手写字体识别上，通常采用卷积神经网络架构（CNN）进行学习预测，最经典的属1998年由Yann LeCun创建的LeNet5架构，  
其中分为：  
1、输入层；  
2、卷积层C1；  
3、池化层S2；  
4、卷积层C3；  
5、池化层S4；  
6、全连接F6；  
7、全连接；  
8、全连接OUTPUT。

在构建LeNet5前，我们需要对全连接层以及卷积层进行初始化。

TruncatedNormal：参数初始化方法，MindSpore支持TruncatedNormal、Normal、Uniform等多种参数初始化方法，具体可以参考MindSpore API的mindspore.common.initializer（<https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/mindspore.common.initializer.html>）模块说明。

import mindspore.nn as nn

from mindspore.common.initializer import TruncatedNormal

# Initialize 2D convolution function

def conv(in\_channels, out\_channels, kernel\_size, stride=1, padding=0):

"""Conv layer weight initial."""

weight = weight\_variable()

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

kernel\_size=kernel\_size, stride=stride, padding=padding,

weight\_init=weight, has\_bias=False, pad\_mode="valid")

# Initialize full connection layer

def fc\_with\_initialize(input\_channels, out\_channels):

"""Fc layer weight initial."""

weight = weight\_variable()

bias = weight\_variable()

return nn.Dense(input\_channels, out\_channels, weight, bias)

# Set truncated normal distribution

def weight\_variable():

"""Weight initial."""

return TruncatedNormal(0.02)

使用MindSpore定义神经网络需要继承mindspore.nn.cell.Cell，Cell是所有神经网络（Conv2d等）的基类。

神经网络的各层需要预先在\_\_init\_\_()方法中定义，然后通过定义construct()方法来完成神经网络的前向构造，按照LeNet5的网络结构，定义网络各层如下：

class LeNet5(nn.Cell):

"""Lenet network structure."""

# define the operator required

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

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

self.batch\_size = 32 # 32 pictures in each group

self.conv1 = conv(1, 6, 5) # Convolution layer 1, 1 channel input (1 Figure), 6 channel output (6 figures), convolution core 5 \* 5

self.conv2 = conv(6, 16, 5) # Convolution layer 2,6-channel input, 16 channel output, convolution kernel 5 \* 5

self.fc1 = fc\_with\_initialize(16 \* 5 \* 5, 120)

self.fc2 = fc\_with\_initialize(120, 84)

self.fc3 = fc\_with\_initialize(84, 10)

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

# use the preceding operators to construct networks

def construct(self, x):

x = self.conv1(x) # 1\*32\*32-->6\*28\*28

x = self.relu(x) # 6\*28\*28-->6\*14\*14

x = self.max\_pool2d(x) # Pool layer

x = self.conv2(x) # Convolution layer

x = self.relu(x) # Function excitation layer

x = self.max\_pool2d(x) # Pool layer

x = self.flatten(x) # Dimensionality reduction

x = self.fc1(x) # Full connection

x = self.relu(x) # Function excitation layer

x = self.fc2(x) # Full connection

x = self.relu(x) # Function excitation layer

x = self.fc3(x) # Full connection

return x

构建完成后，我们将LeNet5的整体参数打印出来查看一下。

network = LeNet5()

print(network)

输出：

LeNet5<

(conv1): Conv2d<input\_channels=1, output\_channels=6, kernel\_size=(5, 5), stride=(1, 1), pad\_mode=valid, padding=0, dilation=(1, 1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.TruncatedNormal object at 0xffff3c079ed0>, bias\_init=zeros, format=NCHW>

(conv2): Conv2d<input\_channels=6, output\_channels=16, kernel\_size=(5, 5), stride=(1, 1), pad\_mode=valid, padding=0, dilation=(1, 1), group=1, has\_bias=False, weight\_init=<mindspore.common.initializer.TruncatedNormal object at 0xffff2c0a61d0>, bias\_init=zeros, format=NCHW>

(fc1): Dense<input\_channels=400, output\_channels=120, has\_bias=True>

(fc2): Dense<input\_channels=120, output\_channels=84, has\_bias=True>

(fc3): Dense<input\_channels=84, output\_channels=10, has\_bias=True>

(relu): ReLU<>

(max\_pool2d): MaxPool2d<kernel\_size=2, stride=2, pad\_mode=VALID>

(flatten): Flatten<>

>

返回可训练参数的列表：

param = network.trainable\_params()

param

输出：

[Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires\_grad=True),

Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32, requires\_grad=True),

Parameter (name=fc1.weight, shape=(120, 400), dtype=Float32, requires\_grad=True),

Parameter (name=fc1.bias, shape=(120,), dtype=Float32, requires\_grad=True),

Parameter (name=fc2.weight, shape=(84, 120), dtype=Float32, requires\_grad=True),

Parameter (name=fc2.bias, shape=(84,), dtype=Float32, requires\_grad=True),

Parameter (name=fc3.weight, shape=(10, 84), dtype=Float32, requires\_grad=True),

Parameter (name=fc3.bias, shape=(10,), dtype=Float32, requires\_grad=True)]

搭建训练网络并进行训练

构建完成神经网络后，就可以着手进行训练网络的构建，模型训练函数为Model.train(),参数主要包含:

1、圈数epoch size（每圈需要遍历完成1875组图片）;

2、数据集ds\_train;

3、回调函数callbacks包含ModelCheckpoint,LossMonitor、Callback模型检测参数;

4、底层数据通道dataset\_sink\_mode，此参数默认True；因此功能不支持CPU模式，在CPU环境下需设置成False。

# Training and testing related modules

import argparse

from mindspore import Tensor

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

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

from mindspore.train import Model

from mindspore.nn.metrics import Accuracy

from mindspore.nn.loss import SoftmaxCrossEntropyWithLogits

def train\_net(model, epoch\_size, mnist\_path, repeat\_size, ckpoint\_cb, step\_loss\_info):

"""Define the training method."""

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

# load training dataset

ds\_train = create\_dataset(os.path.join(mnist\_path, "train"), 32, repeat\_size)

model.train(epoch\_size, ds\_train, callbacks=[ckpoint\_cb, LossMonitor(), step\_loss\_info], dataset\_sink\_mode=True)

自定义一个存储每一步训练的step和对应loss值的类Step\_loss\_info()，并继承了Callback类，可以自定义训练过程中的处理措施，非常方便，等训练完成后，可将数据绘图查看loss的变化情况。

# Custom callback function

class Step\_loss\_info(Callback):

def step\_end(self, run\_context):

cb\_params = run\_context.original\_args()

# step\_ Loss dictionary for saving loss value and step number information

step\_loss["loss\_value"].append(str(cb\_params.net\_outputs))

step\_loss["step"].append(str(cb\_params.cur\_step\_num))

定义损失函数及优化器

MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。这里使用SoftmaxCrossEntropyWithLogits损失函数。

import os

if os.name == "nt":

os.system('del/f/s/q \*.ckpt \*.meta')# Clean up old run files before in Windows

else:

os.system('rm -f \*.ckpt \*.meta \*.pb')# Clean up old run files before in Linux

lr = 0.01 # learning rate

momentum = 0.9 #

# create the network

network = LeNet5()

# define the optimizer

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

# define the loss function

net\_loss = SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# define the model

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()} )

epoch\_size = 10

mnist\_path = "./MNIST\_Data"

config\_ck = CheckpointConfig(save\_checkpoint\_steps=125, keep\_checkpoint\_max=16)

# save the network model and parameters for subsequence fine-tuning

ckpoint\_cb = ModelCheckpoint(prefix="checkpoint\_lenet", config=config\_ck)

# group layers into an object with training and evaluation features

step\_loss = {"step": [], "loss\_value": []}

# step\_ Loss dictionary for saving loss value and step number information

step\_loss\_info = Step\_loss\_info()

# save the steps and loss value

repeat\_size = 1

train\_net(model, epoch\_size, mnist\_path, repeat\_size, ckpoint\_cb, step\_loss\_info)

输出：

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

epoch: 1 step: 1875, loss is 0.01626664400100708

epoch: 2 step: 1875, loss is 0.07023413479328156

epoch: 3 step: 1875, loss is 0.009500676766037941

epoch: 4 step: 1875, loss is 0.0036996034905314445

epoch: 5 step: 1875, loss is 0.023579785600304604

epoch: 6 step: 1875, loss is 0.04619445651769638

epoch: 7 step: 1875, loss is 0.09923187643289566

epoch: 8 step: 1875, loss is 0.00039681425550952554

epoch: 9 step: 1875, loss is 0.0004879096522927284

epoch: 10 step: 1875, loss is 0.011665047146379948

训练完成后，能在Jupyter的工作路径上生成多个模型文件，名称具体含义checkpoint\_{网络名称}-{第几个epoch}\_{第几个step}.ckpt。

查看损失函数随着训练步数的变化情况

steps = step\_loss["step"]

loss\_value = step\_loss["loss\_value"]

steps = list(map(int, steps))

loss\_value = list(map(float, loss\_value))

plt.plot(steps, loss\_value, color="red")

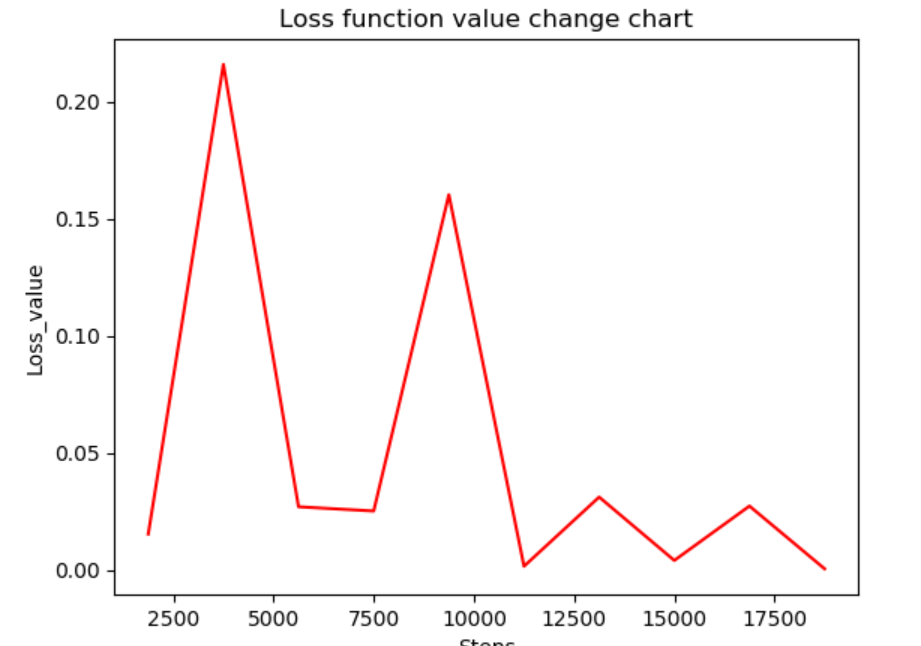
plt.xlabel("Steps")

plt.ylabel("Loss\_value")

plt.title("Loss function value change chart")

plt.show()

输出：



数据测试验证模型精度

搭建测试网络的过程主要为：

1、载入模型.cptk文件中的参数param；

2、将参数param载入到神经网络LeNet5中；

3、载入测试数据集；

4、调用函数model.eval()传入参数测试数据集ds\_eval，就生成模型checkpoint\_lenet-10\_1875.ckpt的精度值。

def test\_net(network, model, mnist\_path):

"""Define the evaluation method."""

print("============== Starting Testing ==============")

# load the saved model for evaluation

param\_dict = load\_checkpoint("checkpoint\_lenet-10\_1875.ckpt")

# load parameter to the network

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

# load testing dataset

ds\_eval = create\_dataset(os.path.join(mnist\_path, "test"))

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

print("============== Accuracy:{} ==============".format(acc))

test\_net(network, model, mnist\_path)

输出：

============== Starting Testing ==============

============== Accuracy:{'Accuracy': 0.969551282051282} ==============

经过1875\*epoch步训练后生成的模型精度超过95%，模型优良。我们可以看一下模型随着训练步数变化，精度随之变化的情况。

acc\_model\_info()函数是将每125步的保存的模型，调用model.eval()函数将测试出的精度返回到步数列表和精度列表，如下：

def acc\_model\_info(network, model, mnist\_path, model\_numbers, epoch\_size):

"""Define the plot info method"""

step\_list = []

acc\_list = []

for i in range(1, epoch\_size +1):

# load the saved model for evaluation

#加载同一个模型得到的模型训练步数变化，精度随之变化

param\_dict = load\_checkpoint("checkpoint\_lenet-1\_1875.ckpt")

#加载不同一个模型得到的模型训练步数变化，精度随之变化

#param\_dict = load\_checkpoint("checkpoint\_lenet-{}\_1875.ckpt".format(str(i)))

# load parameter to the network

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

# load testing dataset

for i in range(1, model\_numbers +1):

ds\_eval = create\_dataset(os.path.join(mnist\_path, "test"))

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

acc\_list.append(acc['Accuracy'])

step\_list.append(i\*125)

return step\_list,acc\_list

# Draw line chart according to training steps and model accuracy

l1,l2 = acc\_model\_info(network, model, mnist\_path, 15, 10)

plt.xlabel("Model of Steps")

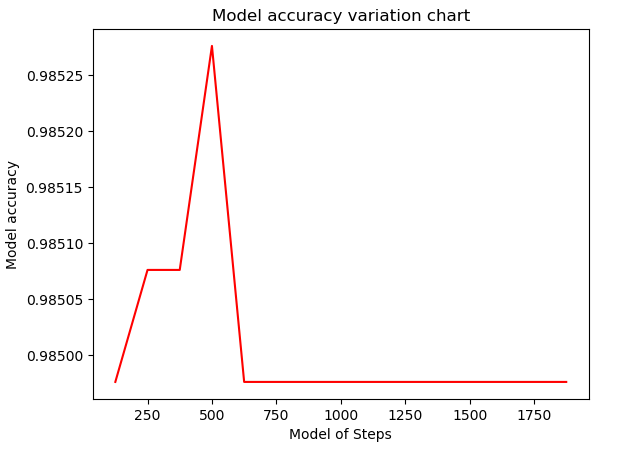
plt.ylabel("Model accuracy")

plt.title("Model accuracy variation chart")

plt.plot(l1, l2, 'red')

plt.show()

输出：



模型预测应用

使用生成的模型应用到分类预测单个或者单组图片数据上，具体步骤如下：

1、需要将要测试的数据转换成适应LeNet5的数据类型。

2、提取出image的数据。

3、使用函数model.predict()预测image对应的数字。需要说明的是predict返回的是image对应0-9的概率值。

4、调用plot\_pie()将预测的各数字的概率显示出来。负概率的数字会被去掉。

ds\_test = create\_dataset(test\_data\_path).create\_dict\_iterator()

data = ds\_test.\_\_next\_\_()

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

labels = data["label"].asnumpy() # The subscript of data picture is the standard for us to judge whether it is correct or not

output =model.predict(Tensor(data['image']))

# The predict function returns the probability of 0-9 numbers corresponding to each picture

prb = output.asnumpy()

pred = np.argmax(output.asnumpy(), axis=1)

err\_num = []

index = 1

for i in range(len(labels)):

plt.subplot(4, 8, i+1)

color = 'blue' if pred[i] == labels[i] else 'red'

plt.title("pre:{}".format(pred[i]), color=color)

plt.imshow(np.squeeze(images[i]))

plt.axis("off")

if color == 'red':

index = 0

# Print out the wrong data identified by the current group

print("Row {}, column {} is incorrectly identified as {}, the correct value should be {}".format(int(i/8)+1, i%8+1, pred[i], labels[i]), '\n')

if index:

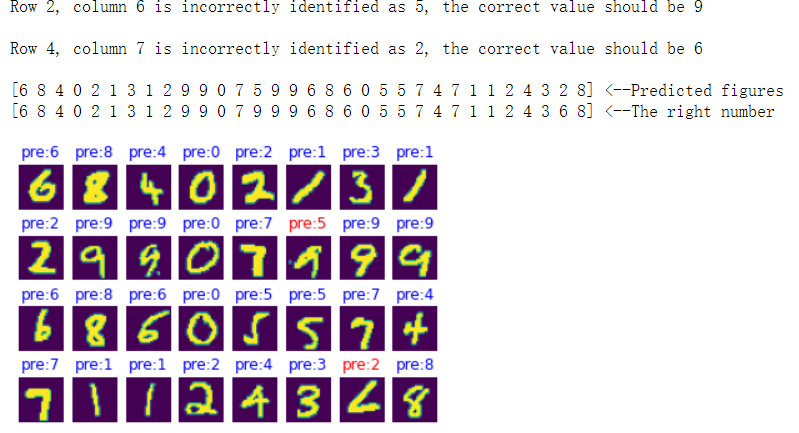
print("All the figures in this group are predicted correctly！")

print(pred, "<--Predicted figures") # Print the numbers recognized by each group of pictures

print(labels, "<--The right number") # Print the subscript corresponding to each group of pictures

plt.show()

输出：



构建一个概率分析的饼图函数。

# define the pie drawing function of probability analysis

def plot\_pie(prbs):

dict1 = {}

# Remove the negative number and build the dictionary dict1. The key is the number and the value is the probability value

for i in range(10):

if prbs[i] > 0:

dict1[str(i)] = prbs[i]

label\_list = dict1.keys() # Label of each part

size = dict1.values() # Size of each part

colors = ["red", "green", "pink", "blue", "purple", "orange", "gray"] # Building a round cake pigment Library

color = colors[: len(size)]# Color of each part

plt.pie(size, colors=color, labels=label\_list, labeldistance=1.1, autopct="%1.1f%%", shadow=False, startangle=90, pctdistance=0.6)

plt.axis("equal") # Set the scale size of x-axis and y-axis to be equal

plt.legend()

plt.title("Image classification")

plt.show()

for i in range(2):

print("Figure {} probability of corresponding numbers [0-9]:\n".format(i+1), prb[i])

plot\_pie(prb[i])

输出：

图表, 饼图

描述已自动生成

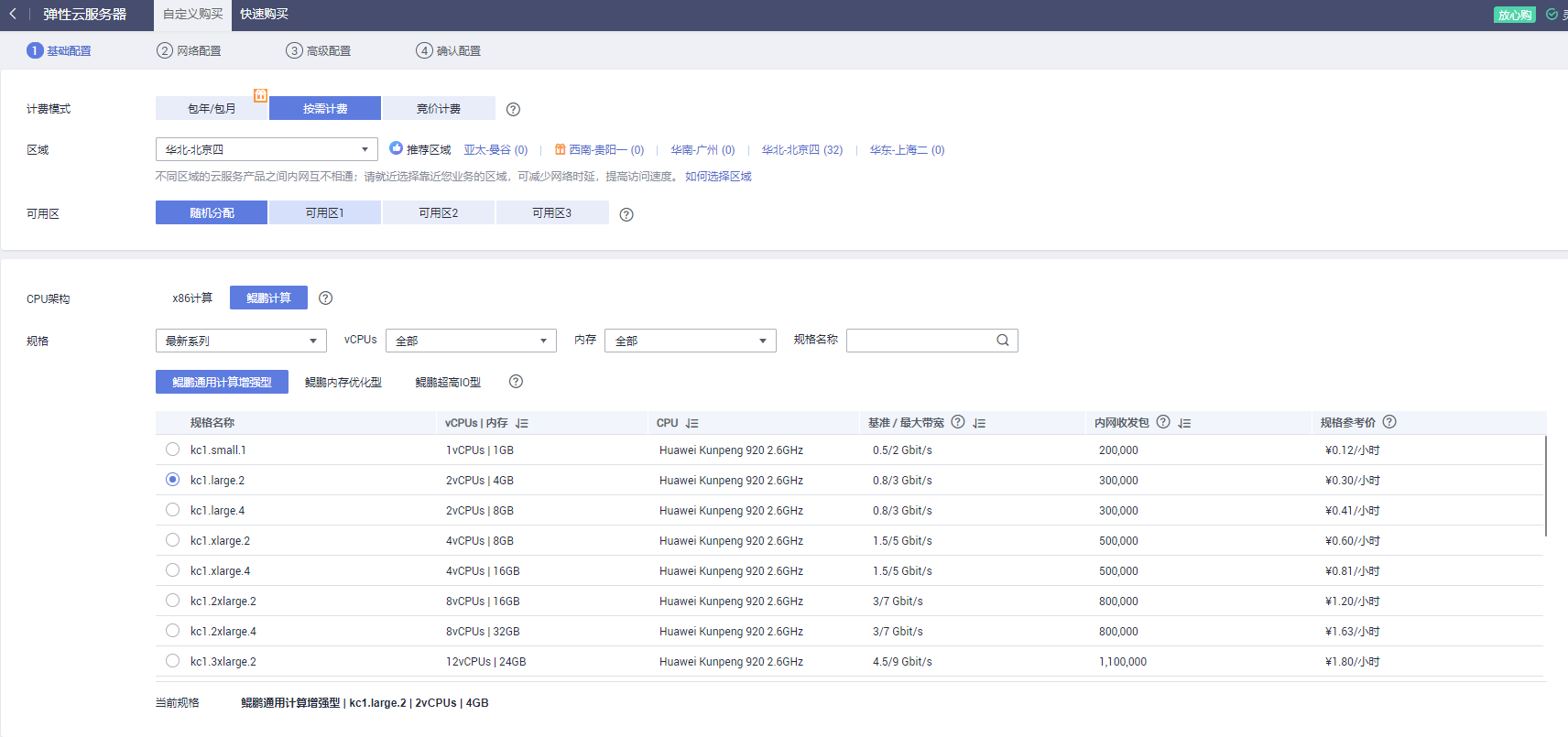
附录2：

# 鲲鹏云搭建python3.7.4

**前提条件：**

（本步骤可省略，沙箱自动提供一个可供用户使用并登陆的ARM（鲲鹏）Linux服务器）

1. 购买并创建华为云ARM（鲲鹏）Linux弹性云服务器。
2. 弹性云服务器需要绑定弹性公网IP
3. 本地已经安装用于连接 Linux 弹性云服务器的工具





# 部署开始

### 1、预制条件

Python可以运行在几乎所有操作系统上，本次安装基于centos7.5环境，由于系统自带python2.7.5环境，本次主要验证python3的安装。

安装依赖：

首先安装gcc编译器，gcc有些系统版本已经默认安装，通过  gcc --version  查看，没安装的先安装gcc

# yum -y install gcc

安装其它依赖包，（注：不要缺少，否则有可能安装python出错，python3.7.4以下的版本可不装 libffi-devel ）

# yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gcc make libffi-devel xz-devel gdbm-devel

### 2、下载python3安装包

在[https://www.python.org/downloads/source/](javascript:;)中选择自己需要的python源码包，本次安装版本为3.7.4

下载

#wget https://www.python.org/ftp/python/3.7.4/Python-3.7.4.tgz

### 3、安装python3

(1) 解压安装包

#tar –zxvf Python-3.7.4.tgz

(2) 建立一个空文件夹，用于存放python3程序

#mkdir /usr/local/python3

(3) 编译源码

#cd Python-3.7.4

#./configure --prefix=/usr/local/python3

#make

#make install

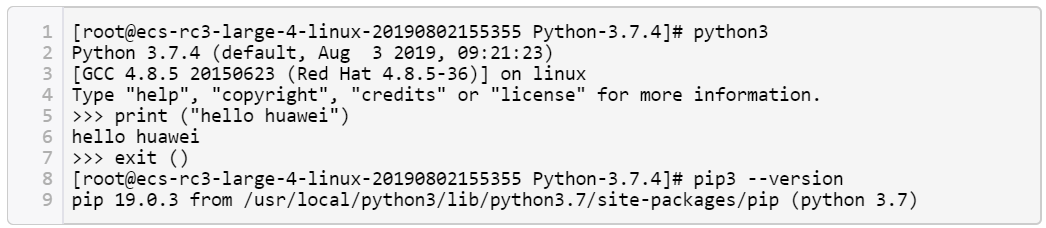
如果没有报错就可以认为安装成功。

(4) 建立软链接

#ln -s /usr/local/python3/bin/python3.7 /usr/bin/python3

#ln –s /usr/local/python3/bin/pip3.7 /usr/bin/pip3

(6) 验证



### 5、常见问题

Python 3.7安装时需要注意安装依赖，依赖缺失会导致安装失败

### 6、 环境信息



Python 2.7安装见<https://bbs.huaweicloud.com/forum/thread-27946-1-1.html>

Python 3.8安装见<https://bbs.huaweicloud.com/blogs/135990>

其他numpy, matlibplot等库的安装请参考centOS下对应的安装方法