**北京科技大学实验报告**

成绩：

学院：计算机与通信工程学院 专业：物联网工程 班级：

姓名： 学号： 实验日期：2024 年 6 月 7 日

**实验简介（200字以内）：**

**在本实验中，我使用了华为提供的Mindspore框架，完成了四项课下实验任务，包括课程学习与环境搭建，重写黑白棋AI类函数，用不同模型重新进行手写数字识别任务，以及线性判别式分析与图片分离等，并较为顺利地完成了相应算法、python程序的设计，并输出较理想的结果，从而对各种人工智能算法有了更加深入的理解。**

# 实验一 华为深度学习平台

## 1.1 实验目的

本章实验的主要目是掌握Python程序的编写方法，具备Python基础编程能力，以及解决实际应用问题的能力；掌握深度学习所需要的预备知识，包含神经网络、数据集、监督学习、无监督学习和凸函数等基本概念；熟悉华为深度学习平台及其使用方法，学会安装本地Mindspore开发框架，了解其设计思路，掌握Mindspore简单应用及开发。

## 1.2 实验环境

操作前提：注册、登录华为云账号

软件环境：Python3.7及以上

开发环境：MindSpore 2.2

MindSpore官网安装页面：https://www.mindspore.cn/install/

MindSpore详细资料查询(**[https://www.mindspore.cn/api/zh-CN/0.2.0-alpha/api/python/mindspore/mindspore.nn.html#mindspore.nn.Dense](https://www.mindspore.cn/api/zh-CN/0.2.0-alpha/api/python/mindspore/mindspore.nn.html" \l "mindspore.nn.Dense)**)

## 1.3 实验简介

MindSpore是华为开发的全场景深度学习框架，是其人工智能解决方案的重要组成部分，与TensorFlow、PyTorch、PaddlePaddle 等流行深度学习框架对标，它提供统一的模型训练、推理和导出等接口，支持端、边、云等不同场景下的灵活部署。MindSpore目前提供了Python编程范式，从后端的硬件支持到前端 API，中间会涉及多种优化与特性。例如不采用计算图的自动微分、自动并行与优化计算过程等等，是和华为旗下昇腾芯片深度优化融合的AI框架。

## 1.4 实验内容

**（1）安装Python3.7**

**（2）打开下面网址**

**<https://edu.huaweicloud.com/courses>**

**推荐报名学习以下课程：**

**《Python入门篇》课程**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXX121+Self-paced&courseType=1>

**• 《人工智能概览》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXE151+Self-paced&courseType=1>

**• 《华为云入门》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXX003+2018.5&courseType=1>

**• 《Python进阶篇》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXX122+Self-paced&courseType=1>

**• 《AI全栈成长计划-AI进阶篇》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXE145+Self-paced&courseType=1>

**• 《华为AI开发框架MindSpore》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXE155+Self-paced&courseType=1>

**• 《神经网络基础》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXE174+Self-paced&courseType=1>

**• 《深度学习概览》**

<https://connect.huaweicloud.com/courses/learn/Learning/sp:cloudEdu_?courseNo=course-v1:HuaweiX+CBUCNXE153+Self-paced&courseType=1>

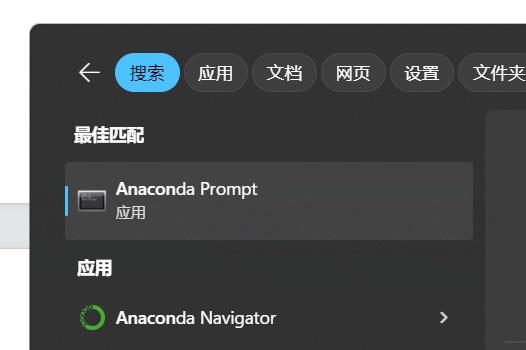
**[《机器学习》&《模式识别》课程方案](https://edu.hicomputing.huawei.com/learn/courses-list/detail/1540025532606275586?l=L3_CUSTOMER)**

**https://edu.hicomputing.huawei.com/learn/courses-list/detail/1540025532606275586?l=L3\_CUSTOMER**

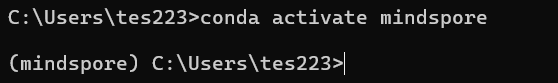
**（3）按照附件《MindSpore环境搭建实验手册》将Mindspore环境安装在本地环境，并展示安装成功的测试截图（最好是找一段Mindspore框架下的应用代码运行测试），在‘1.5 实验步骤’中简要阐述实验步骤，并附上截图.**

## 1.5 步骤

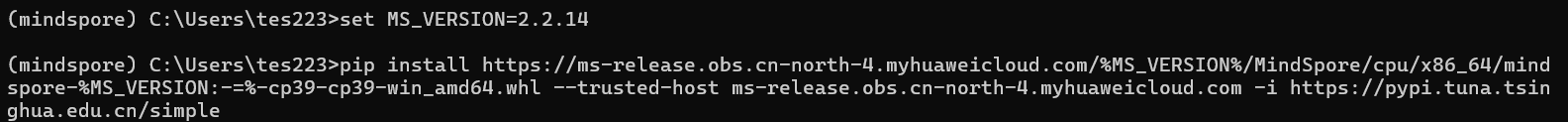
按照《实验手册》中的步骤，首先应安装Miniconda。由于以前进行过相关基础学习，本地环境中已经存在Anaconda，所以不再重复进行安装。（如图所示）



然后根据实验要求创建并激活虚拟环境，如下图所示，该环境已被成功创建并激活。



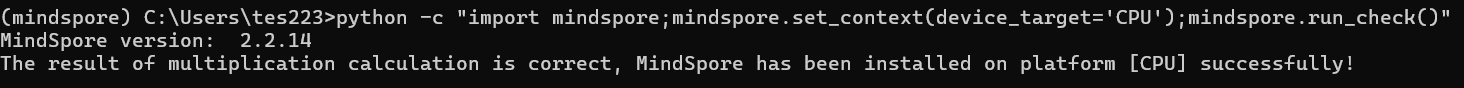
随后，根据指导书要求进行Mindspore的安装。由于本地环境是Windows版本，故只能选择CPU版本进行安装。此处在安装时，指定了Mindspore的版本为2.2.14，安装命令如下图所示。



安装提示完成后，进行验证安装。Mindspore的官网给出了验证安装的方式，如下图所示。



在本地按照上面的步骤进行验证：



提示安装成功。

在实际操作中发现了一个问题：如果按照安装说明中选择较为旧版的2.0.0版本进行安装，由于在验证安装的时候会使用到pillow库的ANTIALIAS方法，而该方法已在pillow库的较新版本中被删除，所以会报错。解决方法是将pillow库降级到旧版，或选择最新的Mindspore版本进行安装。

**参考：**

昇腾开发者社区：https://hiascend.com

昇腾论坛：https://bbs.huaweicloud.com/forum/forum-726-1.html

MindSpore开源社区： https://www.mindspore.cn/

ModelArts社区： https://bbs.huaweicloud.com/forum/forum-718-1.html

Gitee仓库：https://gitee.com/mindspore

# 实验二、黑白棋

演绎推理是由一般到特殊的推理方法，与归纳法相对。推论前提与结论之间的联系是必然的，是一种确实性推理。运用此法研究问题，首先要正确掌握作为指导思想或依据的一般原理、原则；其次要全面了解所要研究的课题、问题的实际情况和特殊性；然后才能推导出一般原理用于特定事物的结论。演绎推理的形式有三段论、假言推理和选言推理等。

## 2.1 实验目的

本章实验的主要目的是掌握逻辑与推理相关基础知识点，了解算法推理相关基础知识，掌握归纳法和演绎推理的主要步骤，熟悉python编程，使用python语言实现黑白棋下棋过程。

## 2.2 实验环境

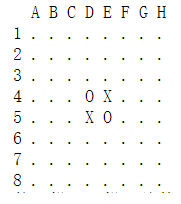
软件环境：Python3.7

开发环境：PC 64bit

## 2.3 实验简介

黑白棋(reversi),也叫苹果棋，翻转棋，是一个经典的策略性游戏。一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，变为己方棋子，故又称“翻转棋”。棋子双面为红、绿色的成为“苹果棋”。它使用8\*8的棋盘，由两人执黑子和白子轮流下棋，最后子多方为胜。

**游戏规则：**棋局开始时黑棋位于E4和D5，白棋位于D4和E5，如下图所示



（1）黑方先行，双方交替下棋。

（2）一步合法的棋步包含：在一个空格新落下一个棋子，并且反转对手一个或多个棋子。

（3）新落下的棋子与棋盘上已有的同色棋子间，对方被夹住的所有棋子都要反转过来。可以横着夹，竖着夹，斜着夹。夹住的位置上必须全部都是对手的棋子，不能有空格。

（4）一步棋可以在数个(横向、纵向、对角线)方向上翻棋，任何被夹住的棋子都必须被翻转过来，棋手无权选择不去翻某个棋子。

（5）除非至少翻转了对手的一个棋子，否则就不能落子。如果一方没有合法棋步，也就是说不管他下到哪里，都不能至少翻转对手的一个棋子，那他这一轮 只能弃权，而由他的对手继续落子直到他有合法棋步可下。

（6）如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。

（7）棋局持续下去，直到棋盘填满或者双方都无合法棋步可下。

## 2.4 实验内容

**（1）掌握并成功运行给定的使用MCTS算法实现miniAlphaGo for Reversi参考代码；**

**（2）在其基础上进行修改，使用《人工智能原理》课程学到的相关算法实现miniAlphaGo for Reversi，在‘2.6 实验步骤’中详细描述所用算法流程、设计实现步骤等；**

**（3）通过实验对人工智能能力在搜索领域的影响有更深入的认识。**

## 2.5 实验详细设计与参考代码

### 2.5.1 下载工具包

!pip install func\_timeout

### 2.5.2 导入实验环境

# 从func\_timeout超时框架导入func\_timeout方法和FunctionTimedOut方法,用于下棋过程中时间的显示

# 如果是第一次安装此模块，请使用命令：pip install func\_timeout

from func\_timeout import func\_timeout, FunctionTimedOut

import datetime # datetime模块用于操作时间

import random # random模块用于生成随机数

from math import log, sqrt # math数学计算库导入对数函数log，开方函数sqrt

from time import time # 时间模块用于时间的设置

from copy import deepcopy # 从copy模块导入深度拷贝方法

**思考题：1、 请描述下random模块的使用方法？**

### 2.5.3 棋盘类

class ReversiBoard(object):

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

"""

初始化棋盘，棋盘大小为8\*8，黑棋用 X 表示，白棋用 O 表示，未落子时用 . 表示

"""

self.board\_init()

def board\_init(self):

"""

重置棋盘

"""

self.empty = '.' # 未落子状态

self.\_board = [[self.empty for \_ in range(8)] for \_ in range(8)] # 规格：8\*8

self.\_board[3][4], self.\_board[4][3] = 'X', 'X' # 黑棋棋子初始状态

self.\_board[3][3], self.\_board[4][4] = 'O', 'O' # 白棋棋子初始状态

def display(self, step\_time=None, total\_time=None):

"""

打印棋盘

:param step\_time: 每一步的耗时, 比如:{"X":1,"O":0},默认值是None

:param total\_time: 总耗时, 比如:{"X":1,"O":0},默认值是None

:return:

"""

board = self.\_board

print(' ', ' '.join(list('ABCDEFGH'))) # 打印列名

for i in range(8): # 打印行名和棋盘

# print(board)

print(str(i + 1), ' '.join(board[i]))

''' Display time. '''

if (not step\_time) or (not total\_time):

# 棋盘初始化时展示的时间

step\_time = {"X": 0, "O": 0}

total\_time = {"X": 0, "O": 0}

print("统计棋局: 棋子总数 / 每一步耗时 / 总时间 ")

print("黑 棋: " + str(self.count('X')) + ' / ' + str(step\_time['X']) + ' / ' + str(

total\_time['X']))

print("白 棋: " + str(self.count('O')) + ' / ' + str(step\_time['O']) + ' / ' + str(

total\_time['O']) + '\n')

else:

# 比赛时展示时间

print("统计棋局: 棋子总数 / 每一步耗时 / 总时间 ")

print("黑 棋: " + str(self.count('X')) + ' / ' + str(step\_time['X']) + ' / ' + str(

total\_time['X']))

print("白 棋: " + str(self.count('O')) + ' / ' + str(step\_time['O']) + ' / ' + str(

total\_time['O']) + '\n')

def count(self, color):

"""

统计 color 一方棋子的数量。(O:白棋, X:黑棋, .:未落子状态)

:param color: [O,X,.] 表示棋盘上不同的棋子

:return: 返回 color 棋子在棋盘上的总数

"""

count = 0

for y in range(8):

for x in range(8):

if self.\_board[x][y] == color:

count += 1

return count

def get\_winner(self):

"""

判断黑棋和白旗的输赢，通过棋子的个数进行判断

:return: 0-黑棋赢，1-白旗赢，2-表示平局，黑棋个数和白旗个数相等

"""

# 定义黑白棋子初始的个数

black\_count, white\_count = 0, 0

for i in range(8):

for j in range(8):

# 统计黑棋棋子的个数

if self.\_board[i][j] == 'X':

black\_count += 1

# 统计白旗棋子的个数

if self.\_board[i][j] == 'O':

white\_count += 1

if black\_count > white\_count:

# 黑棋胜

return 0, black\_count - white\_count

elif black\_count < white\_count:

# 白棋胜

return 1, white\_count - black\_count

elif black\_count == white\_count:

# 表示平局，黑棋个数和白旗个数相等

return 2, 0

def \_move(self, action, color):

"""

落子并获取反转棋子的坐标

:param action: 落子的坐标 可以是 D3 也可以是(2,3)

:param color: [O,X,.] 表示棋盘上不同的棋子

:return: 返回反转棋子的坐标列表，落子失败则返回False

"""

# 判断action 是不是字符串，如果是则转化为数字坐标

if isinstance(action, str):

action = self.board\_num(action)

fliped = self.\_can\_fliped(action, color)

if fliped:

# 有就反转对方棋子坐标

for flip in fliped:

x, y = self.board\_num(flip)

self.\_board[x][y] = color

# 落子坐标

x, y = action

# 更改棋盘上 action 坐标处的状态，修改之后该位置属于 color[X,O,.]等三状态

self.\_board[x][y] = color

return fliped

else:

# 没有反转子则落子失败

return False

def backpropagation(self, action, flipped\_pos, color):

"""

回溯

:param action: 落子点的坐标

:param flipped\_pos: 反转棋子坐标列表

:param color: 棋子的属性，[X,0,.]三种情况

:return:

"""

# 判断action 是不是字符串，如果是则转化为数字坐标

if isinstance(action, str):

action = self.board\_num(action)

#还原棋盘

self.\_board[action[0]][action[1]] = self.empty

# 如果 color == 'X'，则 op\_color = 'O';否则 op\_color = 'X'

op\_color = "O" if color == "X" else "X"

for p in flipped\_pos:

# 判断action 是不是字符串，如果是则转化为数字坐标

if isinstance(p, str):

p = self.board\_num(p)

self.\_board[p[0]][p[1]] = op\_color

def is\_on\_board(self, x, y):

"""

判断坐标是否出界

:param x: row 行坐标

:param y: col 列坐标

:return: True or False

"""

return x >= 0 and x <= 7 and y >= 0 and y <= 7

def \_can\_fliped(self, action, color):

"""

检测落子是否合法,如果不合法，返回 False，否则返回反转子的坐标列表

:param action: 下子位置

:param color: [X,0,.] 棋子状态

:return: False or 反转对方棋子的坐标列表

"""

# 判断action 是不是字符串，如果是则转化为数字坐标

if isinstance(action, str):

action = self.board\_num(action)

xstart, ystart = action

# 如果该位置已经有棋子或者出界，返回 False

if not self.is\_on\_board(xstart, ystart) or self.\_board[xstart][ystart] != self.empty:

return False

# 临时将color放到指定位置

self.\_board[xstart][ystart] = color

# 棋手

op\_color = "O" if color == "X" else "X"

# 要被翻转的棋子

flipped\_pos = [] #数字坐标

flipped\_pos\_board = [] #棋盘坐标

for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0],

[-1, 1]]:

x, y = xstart, ystart

x += xdirection

y += ydirection

# 如果(x,y)在棋盘上，而且为对方棋子,则在这个方向上继续前进，否则循环下一个角度。

if self.is\_on\_board(x, y) and self.\_board[x][y] == op\_color:

x += xdirection

y += ydirection

# 进一步判断点(x,y)是否在棋盘上，如果不在棋盘上，继续循环下一个角度,如果在棋盘上，则进行while循环。

if not self.is\_on\_board(x, y):

continue

# 一直走到出界或不是对方棋子的位置

while self.\_board[x][y] == op\_color:

# 如果一直是对方的棋子，则点（x,y）一直循环，直至点（x,y)出界或者不是对方的棋子。

x += xdirection

y += ydirection

# 点(x,y)出界了和不是对方棋子

if not self.is\_on\_board(x, y):

break

# 出界了，则没有棋子要翻转OXXXXX

if not self.is\_on\_board(x, y):

continue

# 是自己的棋子OXXXXXXO

if self.\_board[x][y] == color:

while True:

x -= xdirection

y -= ydirection

# 回到了起点则结束

if x == xstart and y == ystart:

break

# 需要翻转的棋子

flipped\_pos.append([x, y])

# 将前面临时放上的棋子去掉，即还原棋盘

self.\_board[xstart][ystart] = self.empty # restore the empty space

# 没有要被翻转的棋子，则走法非法。返回 False

if len(flipped\_pos) == 0:

return False

for fp in flipped\_pos:

flipped\_pos\_board.append(self.num\_board(fp))

# 走法正常，返回翻转棋子的棋盘坐标

return flipped\_pos\_board

def get\_legal\_actions(self, color):

"""

按照黑白棋的规则获取棋子的合法走法

:param color: 不同颜色的棋子，X-黑棋，O-白棋

:return: 生成合法的落子坐标，用list()方法可以获取所有的合法坐标

"""

# 表示棋盘坐标点的8个不同方向坐标，比如方向坐标[0][1]则表示坐标点的正上方。

direction = [(-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1)]

op\_color = "O" if color == "X" else "X"

# 统计 op\_color 一方邻近的未落子状态的位置

op\_color\_near\_points = []

board = self.\_board

for i in range(8):

# i 是行数，从0开始，j是列数，也是从0开始

for j in range(8):

# 判断棋盘[i][j]位子棋子的属性，如果是op\_color，则继续进行下一步操作，

# 否则继续循环获取下一个坐标棋子的属性

if board[i][j] == op\_color:

# dx，dy 分别表示[i][j]坐标在行、列方向上的步长，direction 表示方向坐标

for dx, dy in direction:

x, y = i + dx, j + dy

# 表示x、y坐标值在合理范围，棋盘坐标点board[x][y]为未落子状态，

# 而且（x,y）不在op\_color\_near\_points 中，统计对方未落子状态位置的列表才可以添加该坐标点

if 0 <= x <= 7 and 0 <= y <= 7 and board[x][y] == self.empty and (

x, y) not in op\_color\_near\_points:

op\_color\_near\_points.append((x, y))

l = [0, 1, 2, 3, 4, 5, 6, 7]

for p in op\_color\_near\_points:

# 判断落位是否合法，合法则进行下一步

if self.\_can\_fliped(p, color):

# 判断p是不是数字坐标，如果是则返回棋盘坐标

# p = self.board\_num(p)

if p[0] in l and p[1] in l:

p = self.num\_board(p)

yield p

def board\_num(self, action):

"""

棋盘坐标转化为数字坐标

:param action:棋盘坐标，比如A1

:return:数字坐标，比如 A1 --->(0,0)

"""

row, col = str(action[1]).upper(), str(action[0]).upper()

if row in '12345678' and col in 'ABCDEFGH':

# 坐标正确

x, y = '12345678'.index(row), 'ABCDEFGH'.index(col) # 转化为对应的索引

return x, y

def num\_board(self, action):

"""

数字坐标转化为棋盘坐标

:param action:数字坐标 ,比如(0,0)

:return:棋盘坐标，比如 （0,0）---> A1

"""

row, col = action

l = [0, 1, 2, 3, 4, 5, 6, 7]

if col in l and row in l:

return chr(ord('A') + col) + str(row + 1) #ord()函数：返回字符对应的十进制整数；chr()：返回当前整数对应的 ASCII 字符

### 2.5.4游戏类

class Game(object):

def \_\_init\_\_(self, black\_player, white\_player):

self.game\_init()

def game\_init(self):

self.board = ReversiBoard() # 棋盘

self.current\_player = None

self.black\_player = black\_player # 黑棋一方

self.white\_player = white\_player # 白棋一方

self.black\_player.color = "X"

self.white\_player.color = "O"

def switch\_player(self, black\_player, white\_player):

"""

游戏过程中切换玩家

:param black\_player: 黑棋

:param white\_player: 白棋

:return: 当前玩家

"""

# 如果当前玩家是 None 或者 白棋一方 white\_player，则返回 黑棋一方 black\_player;

if self.current\_player is None:

return black\_player

else:

# 如果当前玩家是黑棋一方 black\_player 则返回 白棋一方 white\_player

if self.current\_player == self.black\_player:

return white\_player

else:

return black\_player

def print\_winner(self, winner):

"""

打印赢家

:param winner: [0,1,2] 分别代表黑棋获胜、白棋获胜、平局3种可能。

:return:

"""

print(['黑棋获胜!', '白棋获胜!', '平局'][winner])

def force\_loss(self, is\_timeout=False, is\_board=False, is\_legal=False):

"""

落子3个不合符规则和超时则结束游戏,修改棋盘也是输

:param is\_timeout: 时间是否超时，默认不超时

:param is\_board: 是否修改棋盘

:param is\_legal: 落子是否合法

:return: 赢家（0,1）,棋子差 0

"""

if self.current\_player == self.black\_player:

win\_color = '白棋 - O'

loss\_color = '黑棋 - X'

winner = 1

else:

win\_color = '黑棋 - X'

loss\_color = '白棋 - O'

winner = 0

if is\_timeout:

print('\n{} 思考超过 60s, {} 胜'.format(loss\_color, win\_color))

if is\_legal:

print('\n{} 落子 3 次不符合规则,故 {} 胜'.format(loss\_color, win\_color))

if is\_board:

print('\n{} 擅自改动棋盘判输,故 {} 胜'.format(loss\_color, win\_color))

diff = 0

return winner, diff

def run(self):

"""

运行游戏

:return:

"""

# 定义统计双方下棋时间

total\_time = {"X": 0, "O": 0}

# 定义双方每一步下棋时间

step\_time = {"X": 0, "O": 0}

# 初始化胜负结果和棋子差

winner = None

diff = -1

# 游戏开始

print('\n=====开始游戏!=====\n')

# 棋盘初始化

self.board.display(step\_time, total\_time)

while True:

# 切换当前玩家,如果当前玩家是 None 或者白棋 white\_player，则返回黑棋 black\_player;

# 否则返回 white\_player。

self.current\_player = self.switch\_player(self.black\_player, self.white\_player)

start\_time = datetime.datetime.now()

# 当前玩家对棋盘进行思考后，得到落子位置

# 判断当前下棋方

color = "X" if self.current\_player == self.black\_player else "O"

# 获取当前下棋方合法落子位置

legal\_actions = list(self.board.get\_legal\_actions(color))

# print("%s合法落子坐标列表："%color,legal\_actions)

if len(legal\_actions) == 0:

# 判断游戏是否结束

if self.game\_over():

# 游戏结束，双方都没有合法位置

winner, diff = self.board.get\_winner() # 得到赢家 0,1,2

break

else:

# 另一方有合法位置,切换下棋方

continue

board = deepcopy(self.board.\_board)

# legal\_actions 不等于 0 则表示当前下棋方有合法落子位置

try:

for i in range(0, 3):

# 获取落子位置

action = func\_timeout(60, self.current\_player.get\_move,

kwargs={'board': self.board})

# 如果 action 是 Q 则说明人类想结束比赛

if action == "Q":

# 说明人类想结束游戏，即根据棋子个数定输赢。

break

if action not in legal\_actions:

# 判断当前下棋方落子是否符合合法落子,如果不合法,则需要对方重新输入

print("你落子不符合规则,请重新落子！")

continue

else:

# 落子合法则直接 break

break

else:

# 落子3次不合法，结束游戏！

winner, diff = self.force\_loss(is\_legal=True)

break

except FunctionTimedOut:

# 落子超时，结束游戏

winner, diff = self.force\_loss(is\_timeout=True)

break

# 结束时间

end\_time = datetime.datetime.now()

if board != self.board.\_board:

# 修改棋盘，结束游戏！

winner, diff = self.force\_loss(is\_board=True)

break

if action == "Q":

# 说明人类想结束游戏，即根据棋子个数定输赢。

winner, diff = self.board.get\_winner() # 得到赢家 0,1,2

break

if action is None:

continue

else:

# 统计一步所用的时间

es\_time = (end\_time - start\_time).seconds

if es\_time > 60:

# 该步超过60秒则结束比赛。

print('\n{} 思考超过 60s'.format(self.current\_player))

winner, diff = self.force\_loss(is\_timeout=True)

break

# 当前玩家颜色，更新棋局

self.board.\_move(action, color)

# 统计每种棋子下棋所用总时间

if self.current\_player == self.black\_player:

# 当前选手是黑棋一方

step\_time["X"] = es\_time

total\_time["X"] += es\_time

else:

step\_time["O"] = es\_time

total\_time["O"] += es\_time

# 显示当前棋盘

self.board.display(step\_time, total\_time)

# 判断游戏是否结束

if self.game\_over():

# 游戏结束

winner, diff = self.board.get\_winner() # 得到赢家 0,1,2

break

print('\n=====游戏结束!=====\n')

self.board.display(step\_time, total\_time)

self.print\_winner(winner)

# 返回'black\_win','white\_win','draw',棋子数差

if winner is not None and diff > -1:

result = {0: 'black\_win', 1: 'white\_win', 2: 'draw'}[winner]

# return result,diff

def game\_over(self):

"""

判断游戏是否结束

:return: True/False 游戏结束/游戏没有结束

"""

# 根据当前棋盘，判断棋局是否终止

# 如果当前选手没有合法下棋的位子，则切换选手；如果另外一个选手也没有合法的下棋位置，则比赛停止。

b\_list = list(self.board.get\_legal\_actions('X'))

w\_list = list(self.board.get\_legal\_actions('O'))

is\_over = len(b\_list) == 0 and len(w\_list) == 0 # 返回值 True/False

return is\_over

### 2.5.5 玩家类

class HumanPlayer:

"""

人类玩家

"""

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

"""

玩家初始化

:param color: 下棋方，'X' - 黑棋，'O' - 白棋

"""

self.color = color

def get\_move(self, board):

"""

根据当前棋盘输入人类合法落子位置

:param board: 棋盘

:return: 人类下棋落子位置

"""

# 如果 self.color 是黑棋 "X",则 player 是 "黑棋"，否则是 "白棋"

if self.color == "X":

player = "黑棋"

else:

player = "白棋"

# 人类玩家输入落子位置，如果输入 'Q', 则返回 'Q'并结束比赛。

# 如果人类玩家输入棋盘位置，e.g. 'A1'，

# 首先判断输入是否正确，然后再判断是否符合黑白棋规则的落子位置

while True:

action = input(

"请'{}-{}'方输入一个合法的坐标(e.g. 'D3'，若不想进行，请务必输入'Q'结束游戏。): ".format(player,

self.color))

# 如果人类玩家输入 Q 则表示想结束比赛

if action == "Q" or action == 'q':

return "Q"

else:

row, col = action[1].upper(), action[0].upper()

# 检查人类输入是否正确

if row in '12345678' and col in 'ABCDEFGH':

# 检查人类输入是否为符合规则的可落子位置

if action in board.get\_legal\_actions(self.color):

return action

else:

print("你的输入不合法，请重新输入!")

### 2.5.6 基本函数

def oppo(color):

"""

交换棋手

:return: 切换下一步落子棋手

"""

if color == 'X':

return 'O'

return 'X'

class TreeNode():

"""

蒙特卡洛树节点

"""

def \_\_init\_\_(self, parent, color):

self.parent = parent

self.w = 0

self.n = 0

self.color = color

self.child = dict()

### 2.5.7 AI类

class SilentGame(object):

''' 重构游戏类，模拟下棋过程中，不实时打印棋盘 '''

def \_\_init\_\_(self, black\_player, white\_player, board=ReversiBoard(), current\_player=None):

self.board = deepcopy(board) # 棋盘

# 定义棋盘上当前下棋棋手，先默认是 None

self.current\_player = current\_player

self.black\_player = black\_player # 黑棋一方

self.white\_player = white\_player # 白棋一方

self.black\_player.color = "X"

self.white\_player.color = "O"

def switch\_player(self, black\_player, white\_player):

"""

游戏过程中切换玩家

:param black\_player: 黑棋

:param white\_player: 白棋

:return: 当前玩家

"""

# 如果当前玩家是 None 或者 白棋一方 white\_player，则返回 黑棋一方 black\_player;

if self.current\_player is None:

return black\_player

else:

# 如果当前玩家是黑棋一方 black\_player 则返回 白棋一方 white\_player

if self.current\_player == self.black\_player:

return white\_player

else:

return black\_player

def print\_winner(self, winner):

"""

打印赢家

:param winner: [0,1,2] 分别代表黑棋获胜、白棋获胜、平局3种可能。

:return:

"""

print(['黑棋获胜!', '白棋获胜!', '平局'][winner])

def force\_loss(self, is\_timeout=False, is\_board=False, is\_legal=False):

"""

落子3个不合符规则和超时则结束游戏,修改棋盘也是输

:param is\_timeout: 时间是否超时，默认不超时

:param is\_board: 是否修改棋盘

:param is\_legal: 落子是否合法

:return: 赢家（0,1）,棋子差 0

"""

if self.current\_player == self.black\_player:

win\_color = '白棋 - O'

loss\_color = '黑棋 - X'

winner = 1

else:

win\_color = '黑棋 - X'

loss\_color = '白棋 - O'

winner = 0

if is\_timeout:

print('\n{} 思考超过 60s, {} 胜'.format(loss\_color, win\_color))

if is\_legal:

print('\n{} 落子 3 次不符合规则,故 {} 胜'.format(loss\_color, win\_color))

if is\_board:

print('\n{} 擅自改动棋盘判输,故 {} 胜'.format(loss\_color, win\_color))

diff = 0

return winner, diff

def run(self):

"""

运行游戏

:return:

"""

# 定义统计双方下棋时间

total\_time = {"X": 0, "O": 0}

# 定义双方每一步下棋时间

step\_time = {"X": 0, "O": 0}

# 初始化胜负结果和棋子差

winner = None

diff = -1

# 游戏开始

while True:

# 切换当前玩家,如果当前玩家是 None 或者白棋 white\_player，则返回黑棋 black\_player;

# 否则返回 white\_player。

self.current\_player = self.switch\_player(self.black\_player, self.white\_player)

start\_time = datetime.datetime.now()

# 当前玩家对棋盘进行思考后，得到落子位置

# 判断当前下棋方

color = "X" if self.current\_player == self.black\_player else "O"

# 获取当前下棋方合法落子位置

legal\_actions = list(self.board.get\_legal\_actions(color))

# print("%s合法落子坐标列表："%color,legal\_actions)

if len(legal\_actions) == 0:

# 判断游戏是否结束

if self.game\_over():

# 游戏结束，双方都没有合法位置

winner, diff = self.board.get\_winner() # 得到赢家 0,1,2

break

else:

# 另一方有合法位置,切换下棋方

continue

action = self.current\_player.get\_move(self.board)

if action is None:

continue

else:

self.board.\_move(action, color)

if self.game\_over():

winner, diff = self.board.get\_winner() # 得到赢家 0,1,2

break

return winner, diff

def game\_over(self):

"""

判断游戏是否结束

:return: True/False 游戏结束/游戏没有结束

"""

# 根据当前棋盘，判断棋局是否终止

# 如果当前选手没有合法下棋的位子，则切换选手；如果另外一个选手也没有合法的下棋位置，则比赛停止。

b\_list = list(self.board.get\_legal\_actions('X'))

w\_list = list(self.board.get\_legal\_actions('O'))

is\_over = len(b\_list) == 0 and len(w\_list) == 0 # 返回值 True/False

return is\_over

class RoxannePlayer(object):

''' Roxanne 策略 详见 《Analysis of Monte Carlo Techniques in Othello》 '''

''' 提出者：Canosa, R. Roxanne canosa homepage. https://www.cs.rit.edu/~rlc/ '''

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

"""

Roxanne策略初始化

:param roxanne\_table: 从上到下依次按落子优先级排序

:param color: 执棋方

"""

self.roxanne\_table = [

['A1', 'H1', 'A8', 'H8'],

['C3', 'F3', 'C6', 'F6'],

['C4', 'F4', 'C5', 'F5', 'D3', 'E3', 'D6', 'E6'],

['A3', 'H3', 'A6', 'H6', 'C1', 'F1', 'C8', 'F8'],

['A4', 'H4', 'A5', 'H5', 'D1', 'E1', 'D8', 'E8'],

['B3', 'G3', 'B6', 'G6', 'C2', 'F2', 'C7', 'F7'],

['B4', 'G4', 'B5', 'G5', 'D2', 'E2', 'D7', 'E7'],

['B2', 'G2', 'B7', 'G7'],

['A2', 'H2', 'A7', 'H7', 'B1', 'G1', 'B8', 'G8']

]

self.color = color

def roxanne\_select(self, board):

"""

采用Roxanne 策略选择落子策略

:return: 落子策略

"""

action\_list = list(board.get\_legal\_actions(self.color))

if len(action\_list) == 0:

return None

else:

for move\_list in self.roxanne\_table:

random.shuffle(move\_list)

for move in move\_list:

if move in action\_list:

return move

def get\_move(self, board):

"""

采用Roxanne 策略进行搜索

:return: 落子

"""

if self.color == 'X':

player\_name = '黑棋'

else:

player\_name = '白棋'

# print("请等一会，对方 {}-{} 正在思考中...".format(player\_name, self.color))

action = self.roxanne\_select(board)

return action

class AIPlayer(object):

''' 蒙特卡罗树搜索智能算法 '''

def \_\_init\_\_(self, color, time\_limit=2):

"""

蒙特卡洛树搜索策略初始化

:param color: 执棋方

:param time\_limit: 蒙特卡洛树搜索每步的搜索时间步长

:param tick:记录开始搜索的时间

:param sim\_black, sim\_white: 采用Roxanne策略代替随机策略搜索

"""

self.time\_limit = time\_limit

self.tick = 0

self.sim\_black = RoxannePlayer('X')

self.sim\_white = RoxannePlayer('O')

self.color = color

def mcts(self, board):

"""

蒙特卡洛树搜索，在时间限制范围内，拓展节点搜索结果

:return: 选择最佳拓展

"""

root = TreeNode(None, self.color)

while time() - self.tick < self.time\_limit - 1:

sim\_board = deepcopy(board)

choice = self.select(root, sim\_board)

self.expand(choice, sim\_board)

winner, diff = self.simulate(choice, sim\_board)

back\_score = [1, 0, 0.5][winner]

if choice.color == 'X':

back\_score = 1 - back\_score

self.back\_prop(choice, back\_score)

best\_n = -1

best\_move = None

for k in root.child.keys():

if root.child[k].n > best\_n:

best\_n = root.child[k].n

best\_move = k

return best\_move

def select(self, node, board):

"""

蒙特卡洛树搜索，节点选择

:return: 搜索树向下递归选择子节点

"""

if len(node.child) == 0:

return node

else:

best\_score = -1

best\_move = None

for k in node.child.keys():

if node.child[k].n == 0:

best\_move = k

break

else:

N = node.n

n = node.child[k].n

w = node.child[k].w

score = w / n + sqrt(2 \* log(N) / n)

if score > best\_score:

best\_score = score

best\_move = k

board.\_move(best\_move, node.color)

return self.select(node.child[best\_move], board)

def expand(self, node, board):

"""

蒙特卡洛树搜索，节点扩展

"""

for move in board.get\_legal\_actions(node.color):

node.child[move] = TreeNode(node, oppo(node.color))

def simulate(self, node, board):

"""

蒙特卡洛树搜索，采用Roxanne策略代替随机策略搜索，模拟扩展搜索树

"""

if node.color == 'O':

current\_player = self.sim\_black

else:

current\_player = self.sim\_white

sim\_game = SilentGame(self.sim\_black, self.sim\_white, board, current\_player)

return sim\_game.run()

def back\_prop(self, node, score):

"""

蒙特卡洛树搜索，反向传播，回溯更新模拟路径中的节点奖励

"""

node.n += 1

node.w += score

if node.parent is not None:

self.back\_prop(node.parent, 1 - score)

def get\_move(self, board):

"""

蒙特卡洛树搜索

:return: 采取最佳拓展落子策略

"""

self.tick = time()

if self.color == 'X':

player\_name = '黑棋'

else:

player\_name = '白棋'

# print("请等一会，对方 {}-{} 正在思考中...".format(player\_name, self.color))

action = self.mcts(deepcopy(board))

return action

### 2.5.8 测试

# 人类玩家黑棋初始化

black\_player = HumanPlayer("X")

# AI 玩家 白棋初始化

white\_player = AIPlayer("O")

# 游戏初始化，第一个玩家是黑棋，第二个玩家是白棋

game = Game(black\_player, white\_player)

# 开始下棋

game.run()

输出结果：

=====开始游戏!=====

A B C D E F G H

1 . . . . . . . .

2 . . . . . . . .

3 . . . . . . . .

4 . . . O X . . .

5 . . . X O . . .

6 . . . . . . . .

7 . . . . . . . .

8 . . . . . . . .

统计棋局: 棋子总数 / 每一步耗时 / 总时间

黑 棋: 2 / 0 / 0

白 棋: 2 / 0 / 0

黑棋 - X 思考超过 60s, 白棋 - O 胜

=====游戏结束!=====

A B C D E F G H

1 . . . . . . . .

2 . . . . . . . .

3 . . . . . . . .

4 . . . O X . . .

5 . . . X O . . .

6 . . . . . . . .

7 . . . . . . . .

8 . . . . . . . .

统计棋局: 棋子总数 / 每一步耗时 / 总时间

黑 棋: 2 / 0 / 0

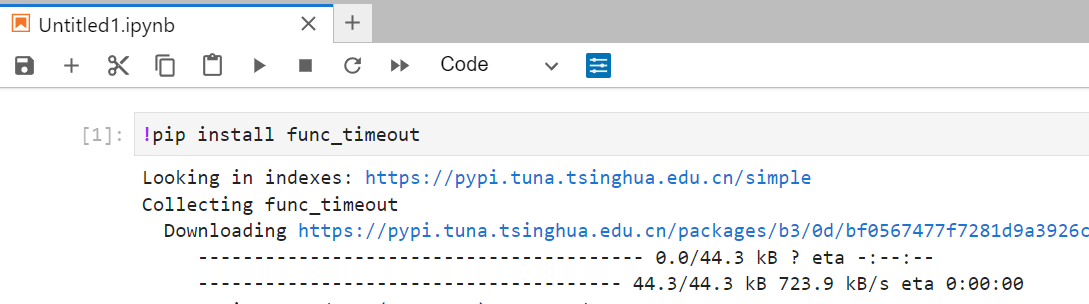
白 棋: 2 / 0 / 0

白棋获胜!

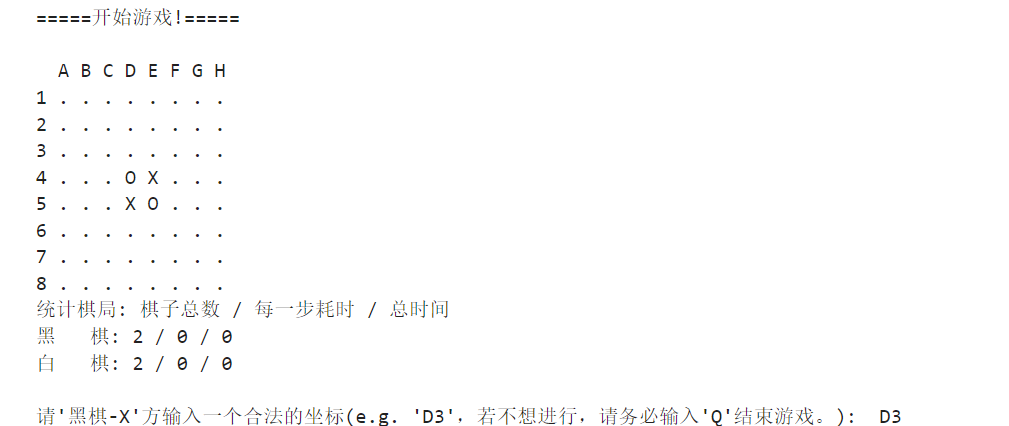
## 2.6 实验步骤

首先进行示例代码的测试。本次测试在本机使用上一步配置好的anaconda环境+jupyter lab工具运行。

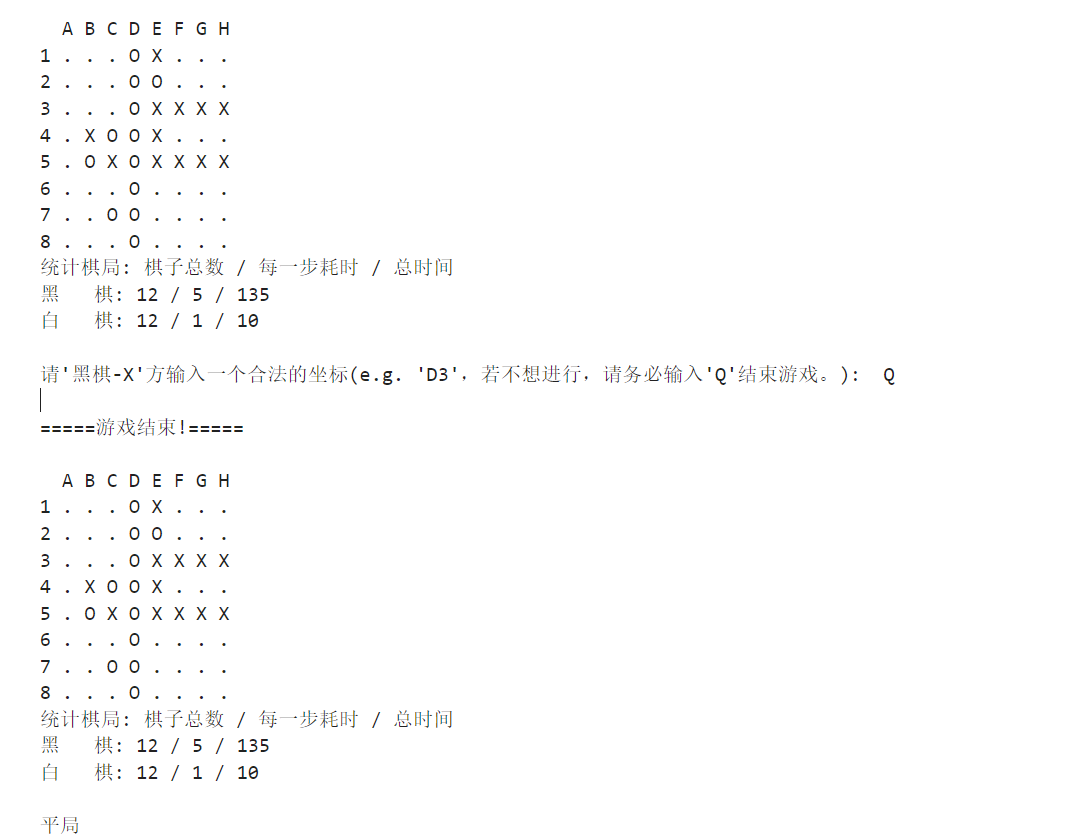
将示例代码在jupyter lab中复现，按照示例代码的要求安装相关库，如下：



库安装完成后，将代码逐步运行，在运行game.run()函数后，即可进行黑白棋下棋的测试，此程序默认是黑方先手，那么输入一个合法的坐标，即可开始下棋。



如上图所示，程序在每步进行下棋后都会输出当前的棋盘情况，并显示当前黑白双方的统计信息，提示用户输入下一步的下棋坐标。



待不想继续下棋后，可按Q键退出，棋盘下满或一方已无处可下棋时，游戏也会自动结束。此时，系统也会输出当前的棋盘情况，并根据双方所拥有棋子的数量情况输出胜负结果。

下面进行算法的修改，只需要对原始代码中的AI类进行相应算法修改即可，代码的其他部分和算法具体实现无关。

本次使用的是mininax算法，即一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的方法。而开始的时候总和为0。

在本实验中，该算法包括以下三个主要函数：

minimax(board, depth, maximizing\_player)是递归函数，根据当前局面和深度，以及当前轮到的玩家是最大化还是最小化玩家，来选择最优的落子策略。在此函数中，我们通过递归调用自身来构建博弈树，并根据评估函数的结果计算每个节点的值。

evaluate(board)是评估函数，用于评估当前棋局的得分。在本次实验中，该函数考虑了棋盘上棋子的数量、角落和边缘的控制情况，分别给出判断依据和权重，最后对相对应的落子策略给出一个简单的评估得分。

get\_move(board)是用于获取最佳落子策略的函数，在此函数中，我们通过调用minimax函数来找到最优的落子。

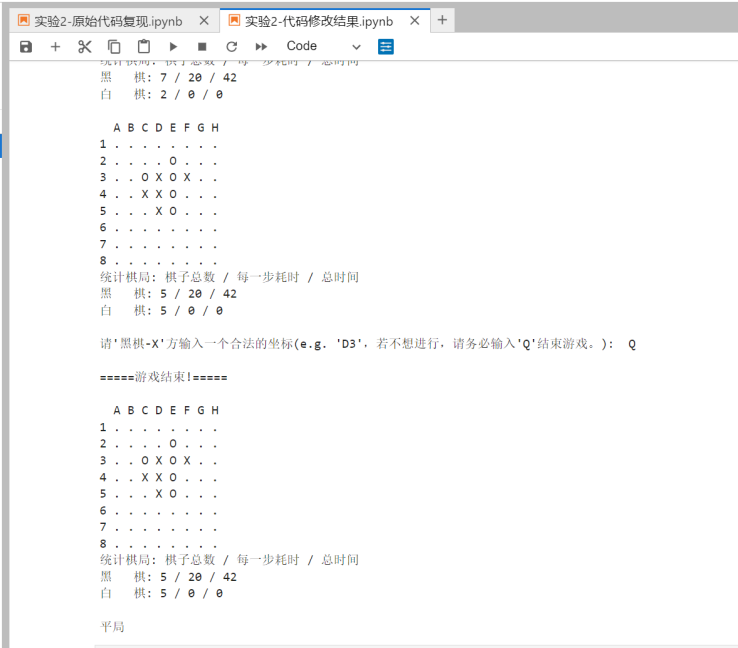
修改部分算法代码的具体实现如下：（在工作文件夹中附带了对应的juputer notebook文件，可以打开文件运行直接进行复现。）







代码运行结果如下：



如图所示，该代码可以正常运行，并给出相应的下棋策略，说明编程结果正确。

# 实验三MNIST手写数字识别

## 3.1 实验目的

本章实验的主要目的是掌握深度学习相关基础知识点，了解深度学习相关基础知识，经典全连接神经网络、卷积神经网络等。掌握不同神经网络架构的设计原理，熟悉使用MindSpore深度学习框架实现深度学习实验的一般流程。

## 3.2 实验环境

软件环境：Python3.7

开发环境：MindSpore

MindSpore官网安装页面：https://www.mindspore.cn/install/

MindSpore详细资料查询(**[https://www.mindspore.cn/api/zh-CN/0.2.0-alpha/api/python/mindspore/mindspore.nn.html#mindspore.nn.Dense](https://www.mindspore.cn/api/zh-CN/0.2.0-alpha/api/python/mindspore/mindspore.nn.html" \l "mindspore.nn.Dense)**)

ModelArts使用：**https://education.huaweicloud.com/courses/course-v1:HuaweiX+CBUCNXE017+Self-paced/about**

## 3.3 实验简介

本例子会实现一个简单的图片分类的功能，整体流程如下：

（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，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。

## 3.4 实验内容

**（1）掌握并成功运行给定的**MNIST手写数字识别程序。

**（2）在其基础上进行修改，使用其他任意数据集替代MNIST数据集或者其他任意深度学习模型替代LeNet进行对比实验，并在‘3.6 实验步骤中’详细阐述：**

**a、所用数据集内容、结构、处理方式；**

**b、所选用深度学习模型结构；**

**c、所选模型与LeNet模型训练结果对比分析。**

## 3.5 实验详细设计与参考代码

### 3.5.1 数据准备

示例中用到的MNIST数据集是由10类28\*28的灰度图片组成，训练数据集包含60000张图片，测试数据集包含10000张图片。

MNIST数据集下载页面：http://yann.lecun.com/exdb/mnist/。页面提供4个数据集下载链接，其中前2个文件是训练数据需要，后2个文件是测试结果需要。

将数据集下载并解压到本地路径下，这里将数据集解压分别存放到工作区的./MNIST\_Data/train、./MNIST\_Data/test路径下。目录结构如下：

└─MNIST\_Data

├─ test

│ t10k-images.idx3-ubyte

│ t10k-labels.idx1-ubyte

│

└─ train

train-images.idx3-ubyte

train-labels.idx1-ubyte

为了方便样例使用，我们在样例脚本中添加了自动下载数据集的功能。

### 3.5.2 导入Python库&模块并配置运行信息

在使用前，导入需要的Python库。目前使用到os库，为方便理解，其他需要的库，我们在具体使用到时再说明。详细的MindSpore的模块说明，可以在MindSpore API页面中搜索查询。可以通过context.set\_context来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。导入context模块，配置运行需要的信息。

import os

import mindspore as ms

import mindspore.context as context

#transforms.c\_transforms用于通用型数据增强，vision.c\_transforms用于图像类数据增强

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

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

#nn模块用于定义网络，model模块用于编译模型，callback模块用于设定监督指标

from mindspore import nn

from mindspore.train import Model

from mindspore.train.callback import LossMonitor

#设定运行模式为图模式，运行硬件为昇腾芯片

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

在样例中我们配置样例运行使用图模式。根据实际情况配置硬件信息，譬如代码运行在Ascend AI处理器上，则device\_target选择Ascend，代码运行在CPU、GPU同理。详细参数说明，请参见context.set\_context接口说明。

### 3.5.3 数据处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。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。

代码如下：

#根据数据集存储地址，生成数据集

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):

#生成训练集和测试集的路径

data\_train = os.path.join(data\_dir, 'train') # train set

data\_test = os.path.join(data\_dir, 'test') # test set

#利用MnistDataset方法读取mnist数据集，如果training是True则读取训练集

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

#map方法是非常有效的方法，可以整体对数据集进行处理，resize改变数据形状，rescale进行归一化，HWC2CHW改变图像通道

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

#利用map方法改变数据集标签的数据类型

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

# shuffle是打乱操作，同时设定了batchsize的大小，并将最后不足一个batch的数据抛弃

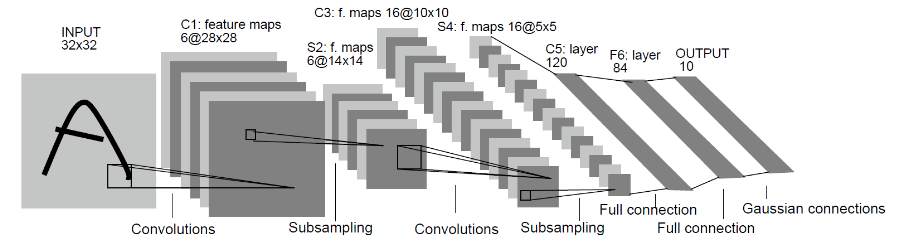
ds = ds.shuffle(buffer\_size=buffer\_size).batch(batch\_size, drop\_remainder=True)

return ds

batch\_size：每组包含的数据个数，现设置每组包含32个数据。先进行修改图片尺寸，归一化，修改图像频道数等工作，再修改标签的数据类型。最后进行shuffle操作，同时设定batch\_size，设置drop\_remainder为True，则数据集中不足最后一个batch的数据会被抛弃。MindSpore也支持进行多种数据处理和增强的操作。

### 3.5.4 定义网络

我们选择相对简单的LeNet网络。LeNet网络不包括输入层的情况下，共有7层：2个卷积层、2个下采样层（池化层）、3个全连接层。每层都包含不同数量的训练参数，如下图所示：



图表 1 **LeNet模型图**

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LeNet的网络结构，定义网络各层如下：

#定义模型结构，MindSpore中的模型时通过construct定义模型结构，在\_\_init\_\_中初始化各层的对象

class LeNet5(nn.Cell):

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

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

#定义卷积层，ReLU激活函数，平坦层和全连接层

#conv2d的输入通道为1维，输出为6维，卷积核尺寸为5\*5，步长为1，不适用padding

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Lenet5架构，x代表网络的输入

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

### 3.5.5 定义损失函数及优化器

在进行定义之前，先简单介绍损失函数及优化器的概念：

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失函数的值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。

**定义损失函数：**

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

MindSpore提供了callback机制，可以在训练过程中执行自定义逻辑，这里使用框架提供的ModelCheckpoint为例。 ModelCheckpoint可以保存网络模型和参数，以便进行后续的fine-tuning（微调）操作。

代码如下：

# 构建训练、验证函数进行模型训练和验证，提供数据路径，设定学习率，epoch数量

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

#调用函数，读取训练集

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

#调用函数，读取验证集

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

#构建网络

net = LeNet5()

#设定loss函数

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

#设定优化器

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

#设定损失监控

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

#编译形成模型

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

# 训练网络，dataset\_sink\_mode为on\_device模式

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

#用验证机评估网络表现

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

#输出相关指标

print('Metrics:', metrics)

### 3.5.6开始训练及验证过程

#main函数负责调用之前定义的函数，完成整个训练验证过程

if \_\_name\_\_ == "\_\_main\_\_":

#argsparse是python的命令行解析的标准模块，可以通过命令行传入参数

import argparse

parser = argparse.ArgumentParser()

#设定训练数据路径

parser.add\_argument('--data\_url', required=False, default='./MNIST\_Data/', help='Location of data.')

parser.add\_argument('--train\_url', required=False, default=None, help='Location of training outputs.')

args, unknown = parser.parse\_known\_args()

#判断路径是否为obs路径，如果是，从obs路径下载数据

if args.data\_url.startswith('s3'):

import moxing

# WAY1: copy dataset from your own OBS bucket to container/cache.

# moxing.file.copy\_parallel(src\_url=args.data\_url, dst\_url='MNIST/')

# WAY2: copy dataset from other's OBS bucket, which has been set public read or public read&write.

moxing.file.copy\_parallel(src\_url="s3://share-course/dataset/MNIST/", dst\_url='MNIST/')

data\_path = 'MNIST/'

else:

data\_path = os.path.abspath(args.data\_url)

#调用train函数，训练并验证模型

train(data\_path)

训练过程中会打印loss值，类似下图。loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。 训练过程中loss打印示例如下：

epoch: 1 step: 1875, loss is 2.2767615

epoch: 2 step: 1875, loss is 0.109801136

epoch: 3 step: 1875, loss is 0.016787775

Metrics: {'acc': 0.9755608974358975, 'loss': 0.07498854234551963}

## 3.6 实验步骤

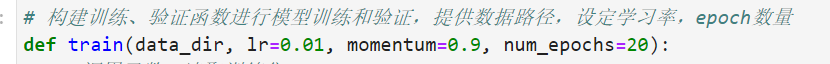
**（按步骤描述自己完成的实验内容）**

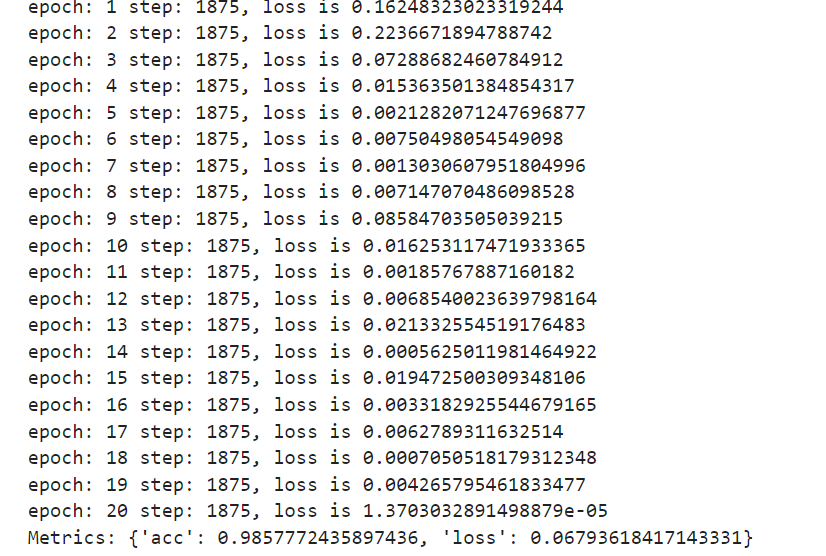
首先进行示例代码的测试，按照上述实验步骤的要求，创建工作文件夹，下载对应数据集。由于本实验所涉及的计算量并不大，所以仍选择在本地环境中利用jupyter lab运行。由于示例代码是用于在华为Ascend环境下运行的，而我们本地安装的是Windows系统下的CPU版本，所以在示例代码中需要将Ascend对应的部分改成CPU，如图所示。

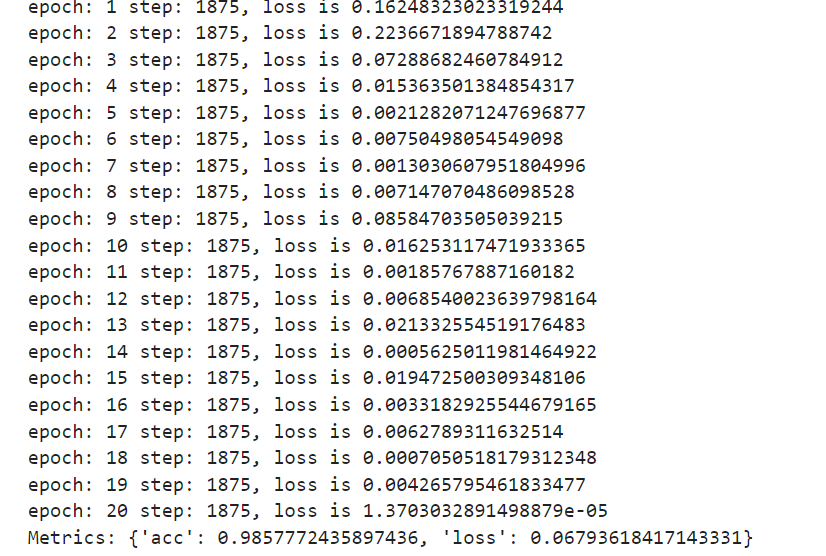


为了使训练效果更好，在本次实验中，我略微增加了训练的轮数，从3修改为了20。

将示例代码逐块在jupyter lab中运行，开始训练后，程序开始逐epoch输出该epoch下的loss结果，如下图所示。







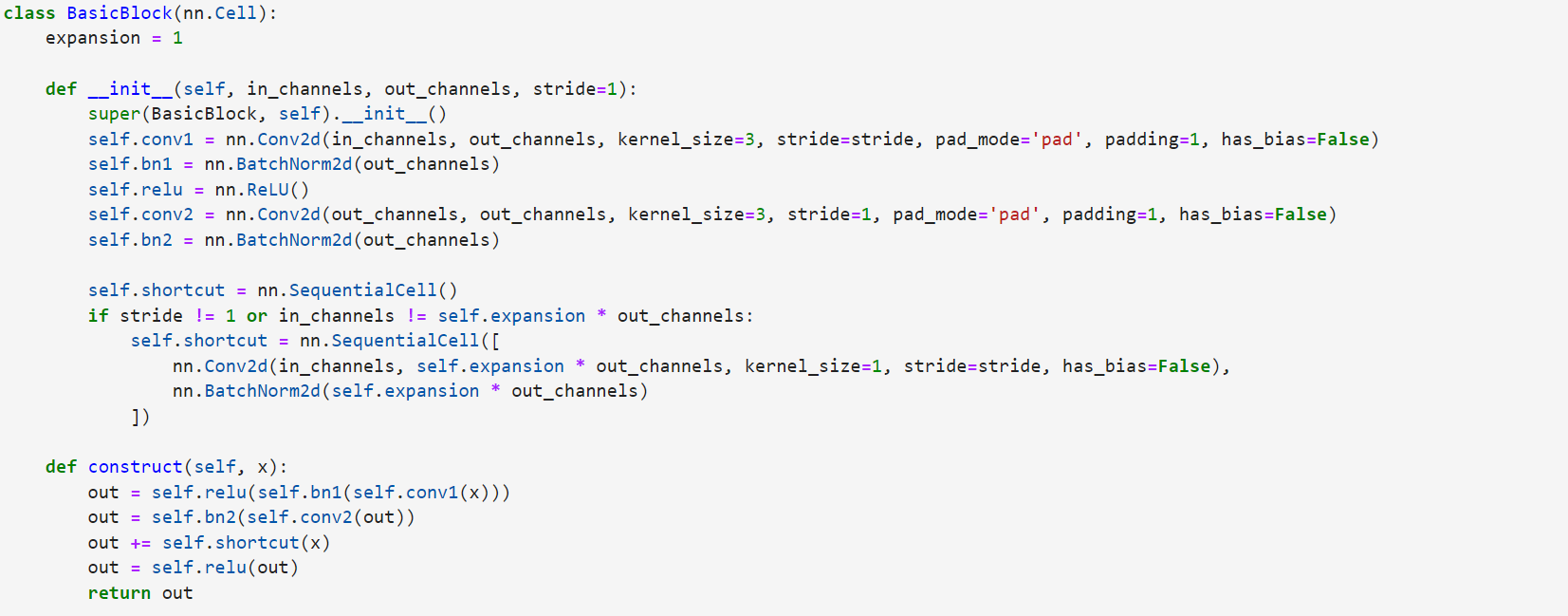
由上图可知，在我们训练完成后，在验证集上进行测试时，数字识别的准确性达到了98.5%，比示例中给出的acc值更高，说明程序运行正确，复现成功。

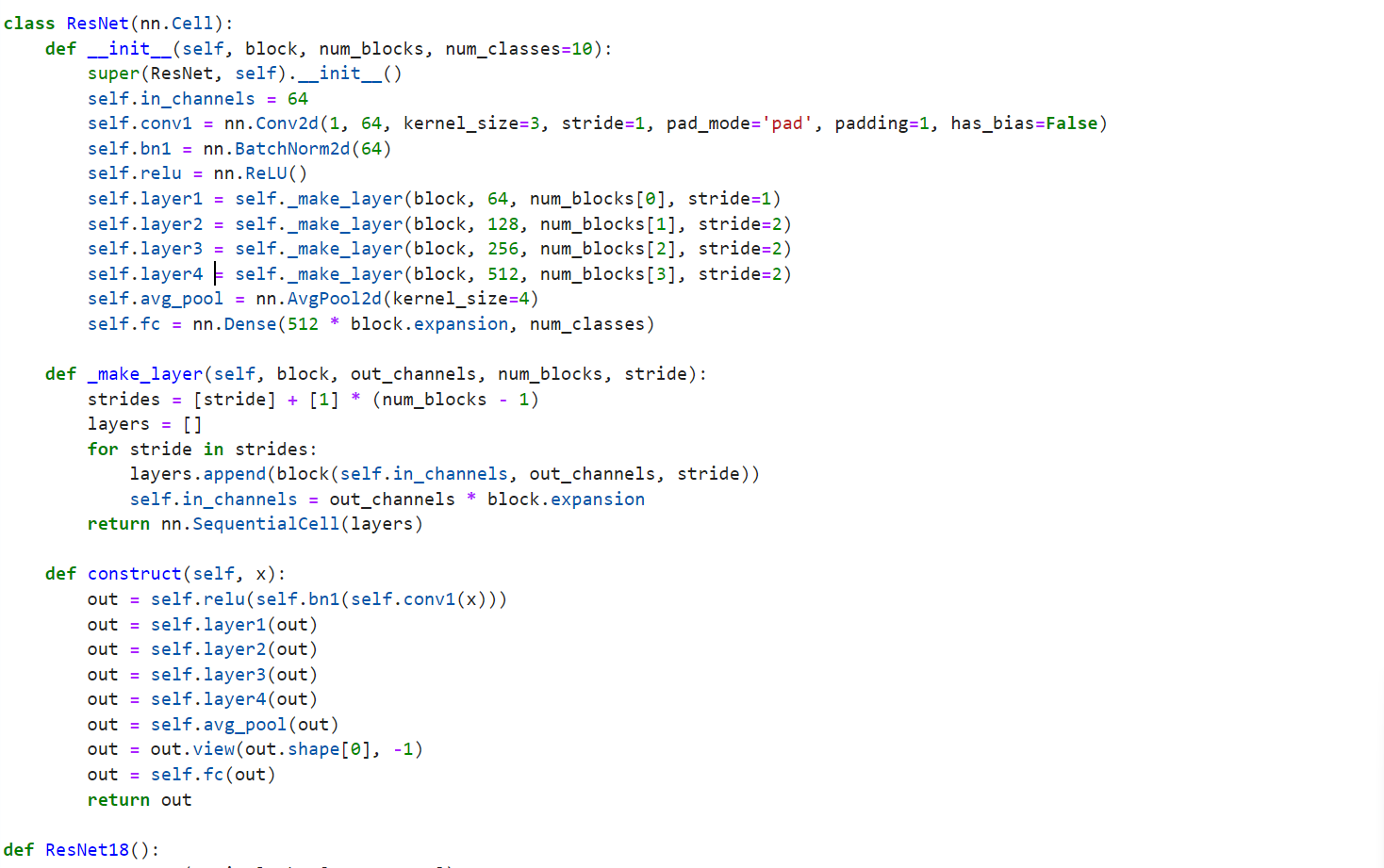
接下来进行程序的修改。在《人工智能与机器学习实验》课程中，我们也做了类似的项目：猫十二分类，当时我使用的是Resnet残差神经网络，达成了相对不错的训练效果，因此在本项目中，我也利用Resnet对实验代码进行修改。

如下图所示，在实验代码中BasicBlock类定义了ResNet中的基本块。它由两个卷积层（conv1和conv2）和两个批归一化层（bn1和bn2）组成。这些层用于提取特征并进行归一化。BasicBlock还包括一个跳跃连接（shortcut），其中会根据需要添加卷积层和批归一化层，以确保跳跃连接的输入和输出维度一致。construct方法用于定义前向传播的计算过程，其中通过ReLU激活函数将输入通过卷积层、批归一化层和跳跃连接相加，然后再次通过ReLU激活函数。

ResNet类定义了整个ResNet模型。它包括一个初始的卷积层（conv1）和批归一化层（bn1），用于对输入进行初始处理。然后，它通过调用\_make\_layer方法构建了四个阶段（layer1，layer2，layer3和layer4），每个阶段包含了多个BasicBlock块。\_make\_layer方法使用block类（和相应的参数创建一个包含多个块的SequentialCell。在每个阶段中，输入通过一系列的块进行处理。最后，输出通过avg\_pool层和fc层进行分类。

construct方法定义了前向传播的计算过程。输入通过初始的卷积层和批归一化层，然后依次通过每个阶段的块。最后，输出通过平均池化层进行降维，并通过全连接层进行分类。

代码经过修改的部分如图所示（在工作文件夹中附带了对应的juputer notebook文件，可以打开文件运行直接进行复现。）  




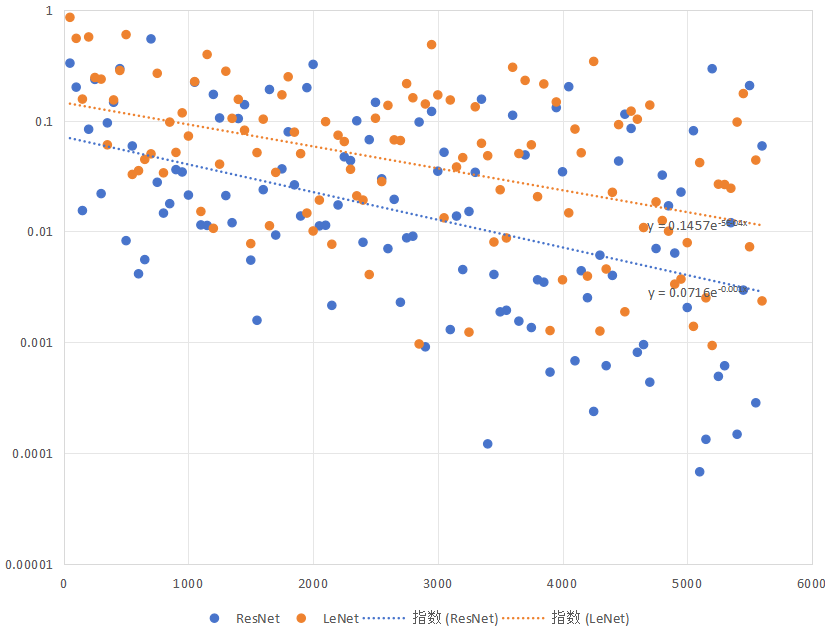
同时将train函数中的net部分修改为ResNet18()。

由于这次是在本地进行的实验，CPU跑ResNet比较慢，因此Resnet的部分只进行了3个epoch的训练，并修改为每进行50个step就输出一次loss值，以便直观地看出step数量和loss值的关系，并将示例代码以同样的方式重跑，将二者的loss、acc值进行对比。

首先关注修改后代码的准确性，如下图所示，该模型在经过了3个epoch后，预测准确性就达到了接近99%，说明模型是有效的，编程是正确的。但是相对LeNet并没有很大的提升。



下面我们关注两个模型的Loss值随时间变化，如下图所示，已经将坐标轴变为了对数坐标轴，并进行了对数拟合。



在经过了对数拟合后，可以明显看出，对应step的loss值，Resnet模型要远低于Lenet模型，说明ResNet在同样的step数量下可能比ResNet更有效。但我们需要注意一个问题，就是算法的效率绝对不能只依靠观察step-loss关系。在这次实际实验中，我使用了笔记本主流CPU：Intel Core i7-12700H进行训练，该CPU具有14核20线程。但是在跑示例代码的LeNet模型时，3个epoch只用了不到一分钟的时间，而换成了ResNet之后训练了将近一个小时才完成了3个epoch，所以造成了较大时间开销，并不能说明在手写数字识别任务上，ResNet比LeNet更好。

# 实验四 线性判别式分析

## 4.1 实验目的

线性判别式分析是非常重要的机器学习概念，LDA的全称是linear discriminant analysis，即线性判别分析，LDA一般用于数据降维。LDA降维的主要原理是寻找一个投影方向，让原先的数据在这个方向上投影后，不同类别之间数据点距离尽可能大，而同类别数据点距离尽可能小。此外，LDA属于监督式学习，本身除了可以用于数据降维外，还可以进行分类预测。

掌握机器学习库sklearn，并能够用python编程实现经典的机器学习算法模型、并对算法进行分析。

## 4.2 实验环境

软件环境：Python3.7

开发环境：PC 64bit

## 4.3 实验简介

（1）复习LDA的基本原理

（2）了解如何使用LDA进行降维和分类

（3）了解python基本的数据清洗和数据分析方法

（4）了解如何使用sklearn中的LDA相关函数

## 4.4 实验内容

实验包含两部分内容：

（1）依据4.5节的实验流程完成实验，并在‘4.6实验步骤中’详细阐述。

（2）对以下图片数据使用LDA进行分类（将图片保存为jpg格式），不需要对3个通道的数据进行降维。

a、依据RGB三个通道的特征，将图片分成黄色、红色、绿色、天蓝色（背景天空）。

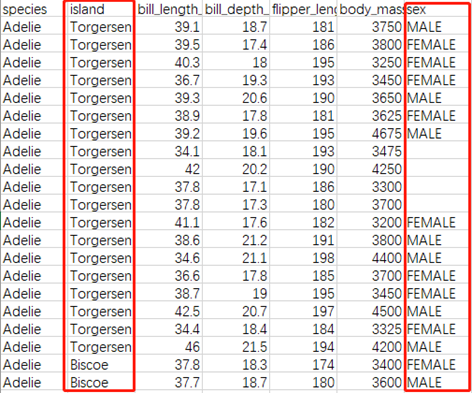
b、在‘4.6实验步骤中’详细阐述实验过程，分类准确率，以及分类超平面的数学表达式。



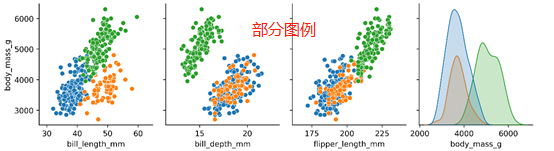
## 4.5 实验流程

本实验将使用palmerpenguins数据集(“peng.csv”)，实验要求如下：

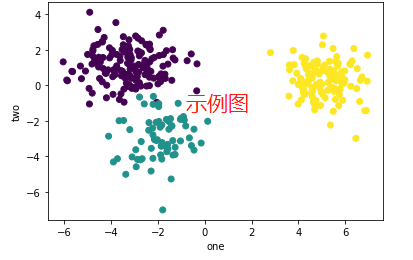
1. 导入csv文件数据，进行数据清洗（去除不完整的特征项和不用的特征项），本次实验要求去掉如下图” island ”和” sex ”特征项



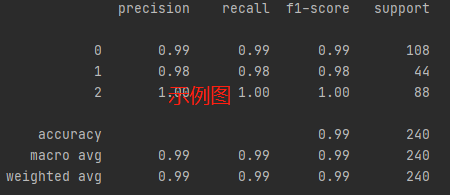
1. 使用seaborn库的*pairplot*函数，展示数据集各属性之间的散点图，并描述此时的目标可分性



1. 分离出目标属性与四个数值型属性
2. 使用sklearn的*LinearDiscriminantAnalysis*将4维数据降维至2维
3. 画出降维后的散点图，并描述此时的目标可分性，绘图结果应如下所示，对比分析降维前后的变化



1. 划分训练集和测试集，利用LDA进行分类预测，分类结果用sklearn.metric中的classification\_report方法显示，如下图，了解各指标含义并分析分类结果。



## 4.6 实验步骤

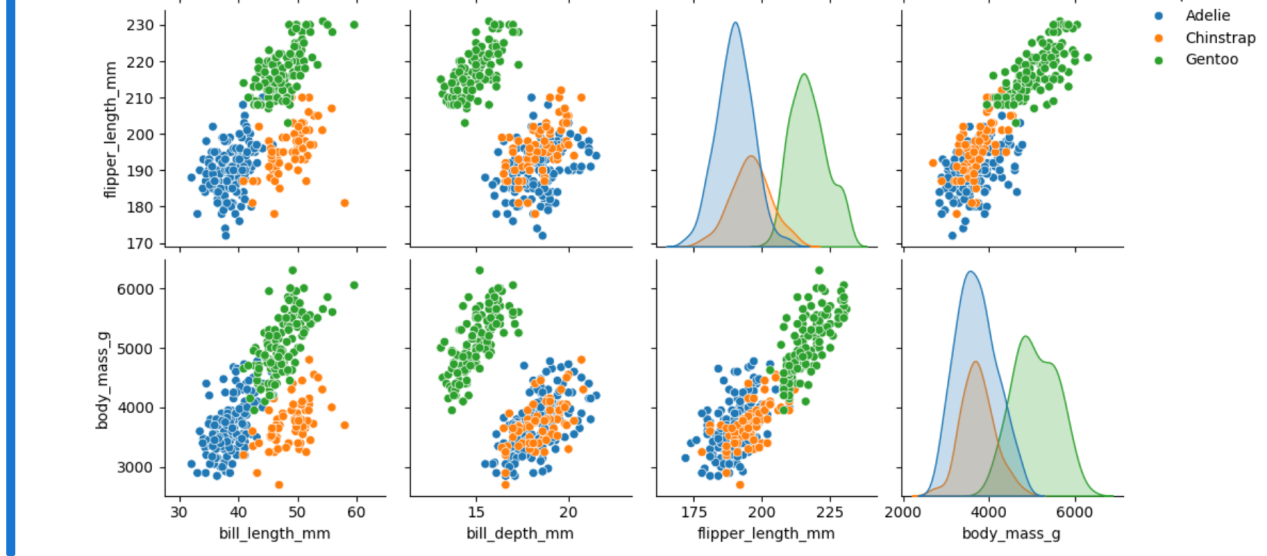
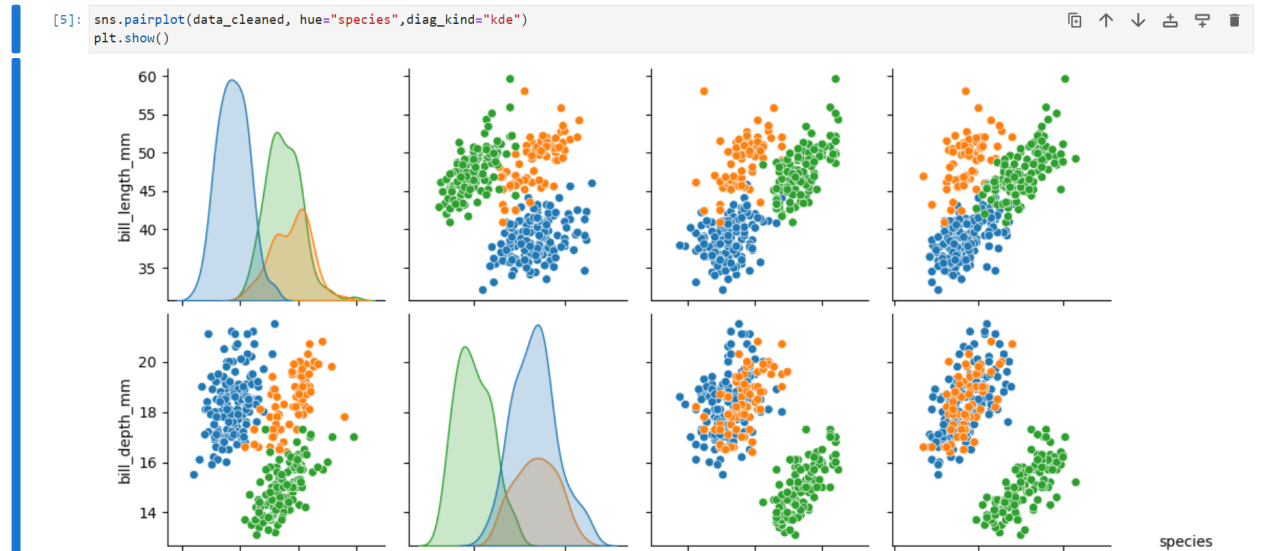
## 本部分的实验代码由于合作版权问题已经删除，此文档中只保留文字说明部分，不包括任何代码部分。

由于本实验所涉及的计算量并不大，所以仍选择在本地环境中利用jupyter lab运行。首先进行相应库的导入，此处手动pip进入环境，安装了pandas, seaborn, matplotlib, scikit-learn, numpy, opencv-python等库，然后导入csv数据。

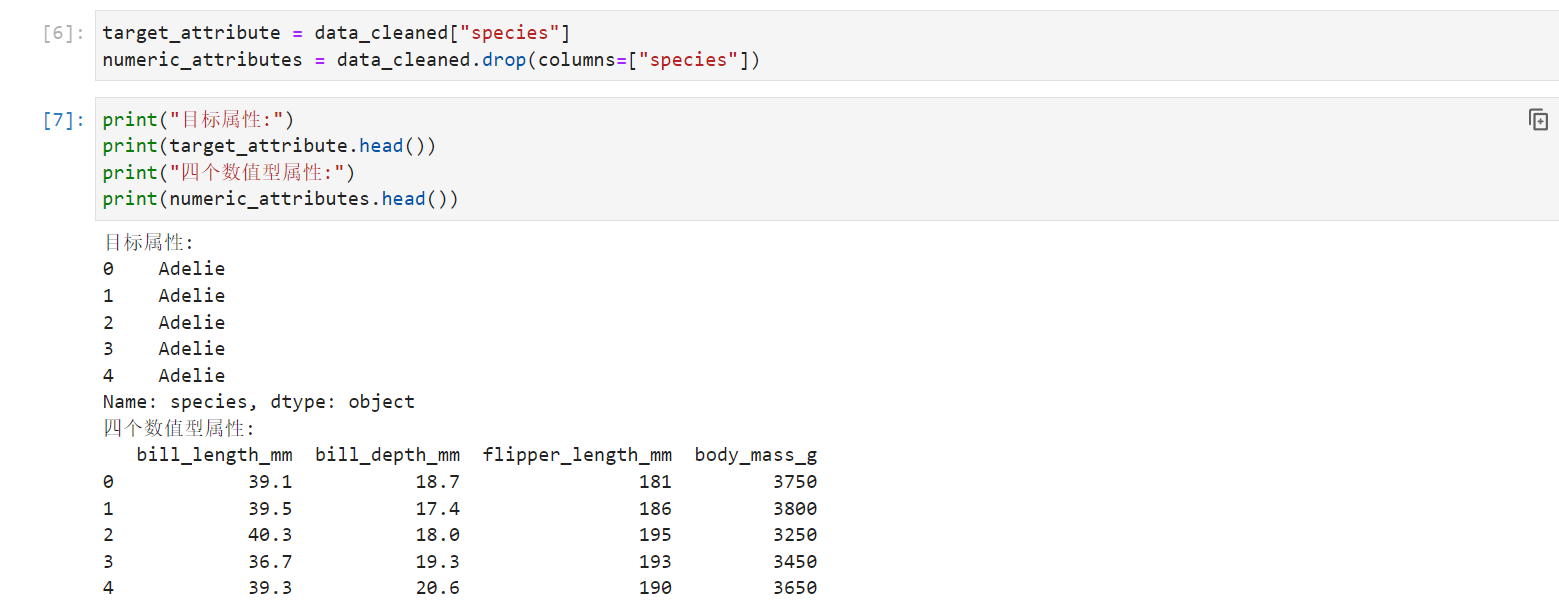
进行数据清洗，删除不用的特征项，并输出清洗结果，确认清理已经完成。

如上图所示，已经达到了实验说明中要求的数据清理结果。

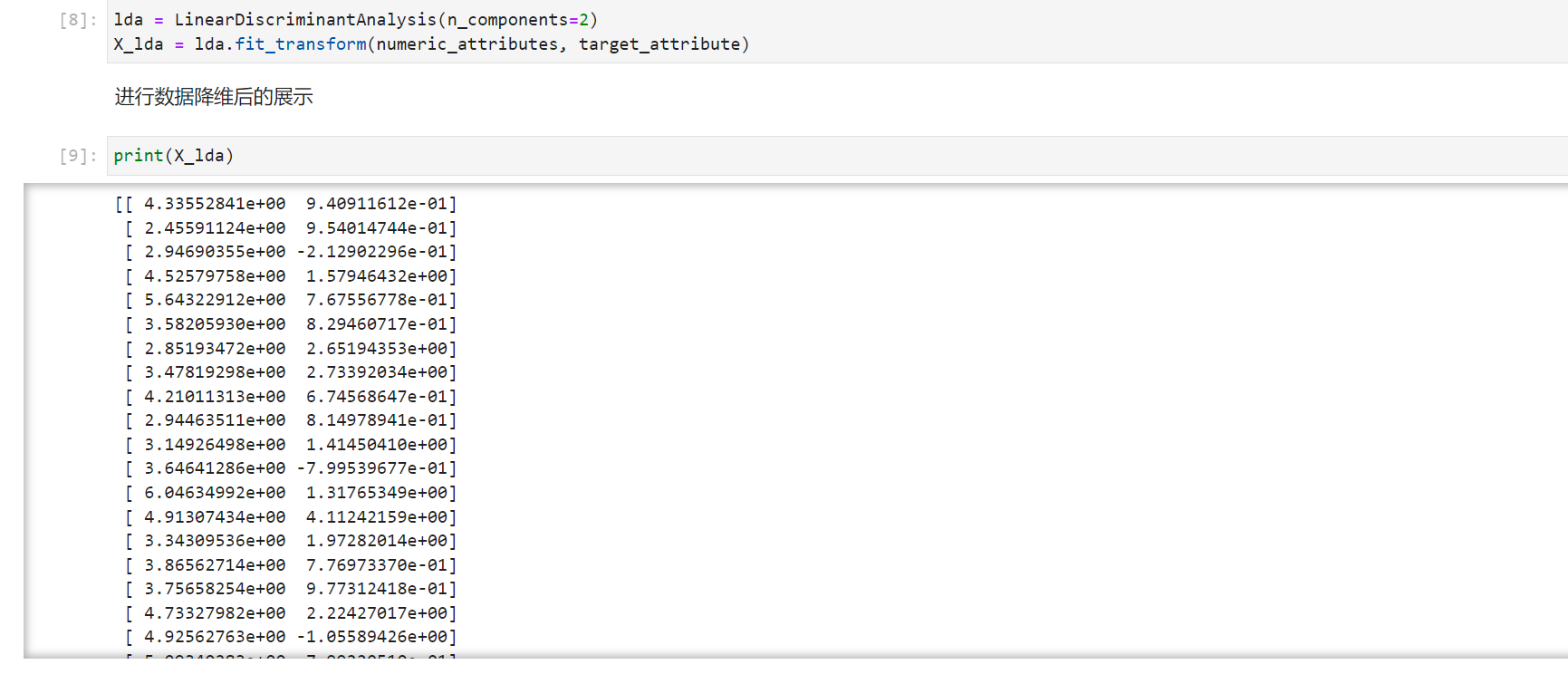
下面利用plt进行图像展示（散点图）。



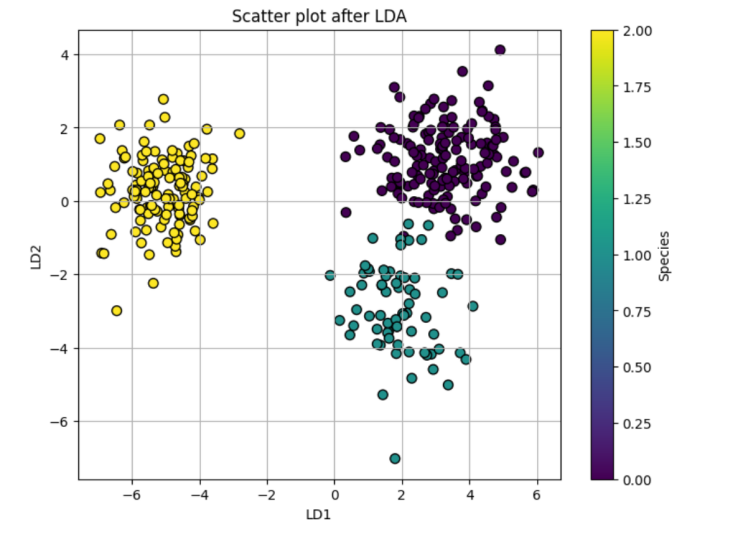
下面进行目标属性分离和降维，先进行属性分离，并将分离后的数据展示，分离目标属性和四个数值型属性，并将属性分离和降维结果进行展示。

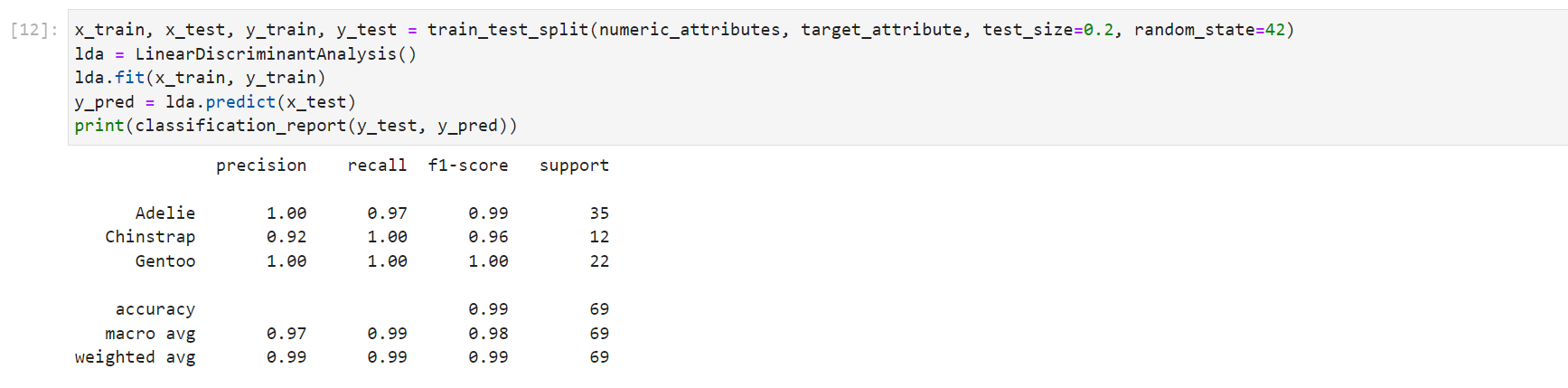


之后进行数据降维，并展示数据降维结果。



将目标属性转换为数字标签，即可进行散点图绘制，输出散点图绘制结果。





上图即为生成的分类报告

最后进行图片分类任务，图片使用的是Word文档中自带的花朵图片，存储在了工作文件夹内部，重命名为test.jpg。

输出的图片展示如下：





至此，图片分离任务顺利完成，分离结果见工作文件夹中的图片。

## 附录：sklearn中的LDA相关函数

LDA算法在sklearn中所在的类：*sklearn.discriminant\_analysis.LinearDiscriminantAnalysis*，它既可以做分类，也可以用来做降维。

由于LDA既能做分类，又能做降维，所以它同时具有分类器和转换器的方法，如分类器的方法：

* *fit(X,y)*：训练模型，需要指定目标变量y
* *predict(X)*：对数据集X进行预测
* *score(X,y)*：评估模型正确率

同时还有转换器的方法：

* *transform(X)*：对数据集X进行降维转换
* *fit\_transform(X,y)*：同时训练和转换，需要指定目标变量y