中 国 地 质 大 学

课程大作业封面

课 程 名 称 人工智能基础

学 生 姓 名 刘瑾瑾

学 生 学 号 20201000128

日 期 2023 年 1 月 1 日

大作业 1：八数码问题的 A*搜索算法实现

一.内容：

设计估价函数（编程语言不限），以八数码为例演示 A*算法的搜索过程，争取做到直观、清晰地演示算法，代码要适当加注释。

八数码问题：在 3×3 方格棋盘上，分别放置了标有数字 1, 2, 3, 4, 5, 6, 7, 8 的八张牌，初始状态 S_0 根据题目要求设定，使用的操作有: 空格上移，空格左移，空格右移，空格下移。试采用 A*算法写程序实现这一搜索过程。

二. 要求：

1. 设置相同的初始状态和目标状态，针对不同的估价函数,求得问题的解，比较它们对搜索算法性能的影响,包括扩展节点数、生成节点数等,填入表 1。
2. 设置与上述 1 相同的初始状态和目标状态,用宽度优先搜索算法（即令估计代价 $h(n)=0$ 的 A*算法）求得问题的解，以及搜索过程中的扩展节点数、生成节点数，填入表 1。

表 1 不同启发函数 $h(n)$求解 8 数码问题的结果比较

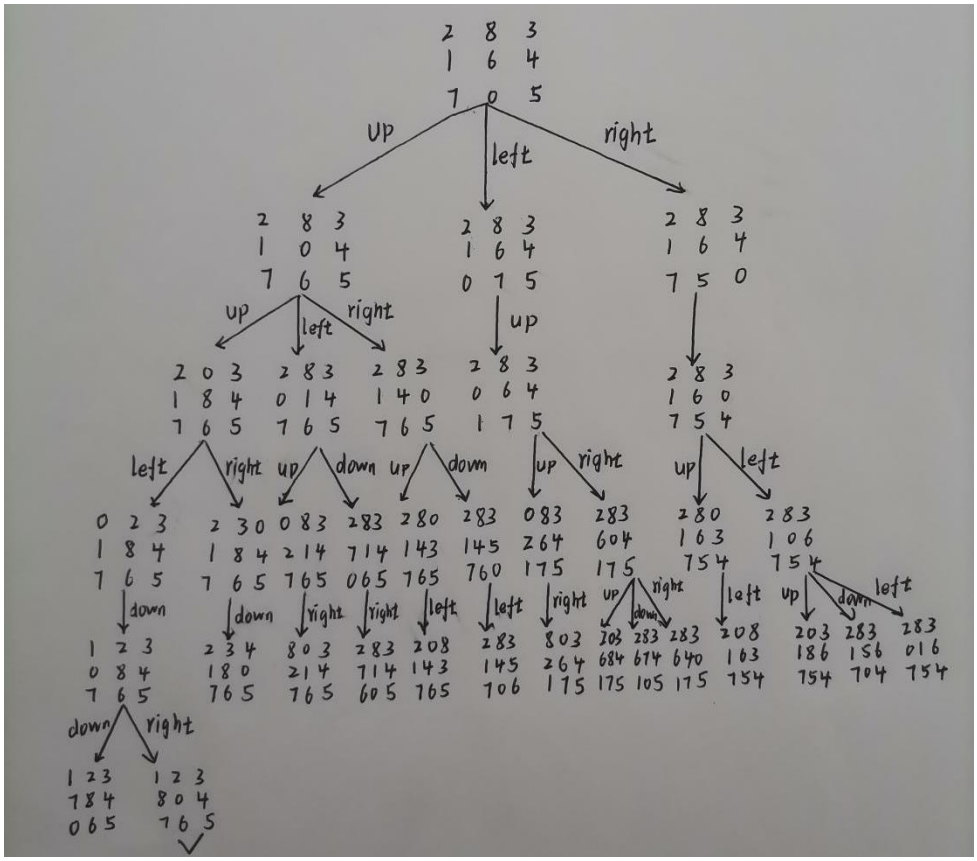
	启发函数 $h(n)$		
	不在位数	哈密顿距离	0
初始状态	1 2 3 8 0 5 4 6 7	1 2 3 8 0 5 4 6 7	1 2 3 8 0 5 4 6 7
目标状态	1 2 3 8 0 7 6 4 5	1 2 3 8 0 7 6 4 5	1 2 3 8 0 7 6 4 5
最优解	最佳路径长度为：16 初始状态： 1 2 3 8 0 5 4 6 7 第 1 层的状态： 1 2 3 8 6 5 4 0 7 第 2 层的状态： 1 2 3 8 6 5 4 7 0 第 3 层的状态： 1 2 3 8 6 0 4 7 5 第 4 层的状态： 1 2 0 8 6 3 4 7 5	最佳路径长度为：16 初始状态： 1 2 3 8 0 5 4 6 7 第 1 层的状态： 1 2 3 8 5 0 4 6 7 第 2 层的状态： 1 2 0 8 5 3 4 6 7 第 3 层的状态： 1 0 2 8 5 3 4 6 7 第 4 层的状态： 0 1 2 8 5 3 4 6 7	最佳路径长度为：16 初始状态： 1 2 3 8 0 5 4 6 7 第 1 层的状态： 1 2 3 8 6 5 4 0 7 第 2 层的状态： 1 2 3 8 6 5 4 7 0 第 3 层的状态： 1 2 3 8 6 0 4 7 5 第 4 层的状态： 1 2 0 8 6 3 4 7 5

	第 5 层的状态： 1 0 2 8 6 3 4 7 5 第 6 层的状态： 0 1 2 8 6 3 4 7 5 第 7 层的状态： 8 1 2 0 6 3 4 7 5 第 8 层的状态： 8 1 2 6 0 3 4 7 5 第 9 层的状态： 8 1 2 6 7 3 4 0 5 第 10 层的状态： 8 1 2 6 7 3 0 4 5 第 11 层的状态： 8 1 2 0 7 3 6 4 5 第 12 层的状态： 0 1 2 8 7 3 6 4 5 第 13 层的状态： 1 0 2 8 7 3 6 4 5 第 14 层的状态： 1 2 0 8 7 3 6 4 5 第 15 层的状态： 1 2 3 8 7 0 6 4 5	第 5 层的状态： 8 1 2 0 5 3 4 6 7 第 6 层的状态： 8 1 2 4 5 3 0 6 7 第 7 层的状态： 8 1 2 4 5 3 6 0 7 第 8 层的状态： 8 1 2 4 0 3 6 5 7 第 9 层的状态： 8 1 2 0 4 3 6 5 7 第 10 层的状态： 0 1 2 8 4 3 6 5 7 第 11 层的状态： 1 0 2 8 4 3 6 5 7 第 12 层的状态： 1 2 0 8 4 3 6 5 7 第 13 层的状态： 1 2 3 8 4 0 6 5 7 第 14 层的状态： 1 2 3 8 4 7 6 5 0 第 15 层的状态： 1 2 3 8 4 7 6 0 5	第 5 层的状态： 1 0 2 8 6 3 4 7 5 第 6 层的状态： 0 1 2 8 6 3 4 7 5 第 7 层的状态： 8 1 2 0 6 3 4 7 5 第 8 层的状态： 8 1 2 6 0 3 4 7 5 第 9 层的状态： 8 1 2 6 7 3 4 0 5 第 10 层的状态： 8 1 2 6 7 3 0 4 5 第 11 层的状态： 8 1 2 0 7 3 6 4 5 第 12 层的状态： 0 1 2 8 7 3 6 4 5 第 13 层的状态： 1 0 2 8 7 3 6 4 5 第 14 层的状态： 1 2 0 8 7 3 6 4 5 第 15 层的状态： 1 2 3 8 7 0 6 4 5

	第 16 层的状态： 1 2 3 8 0 7 6 4 5	第 16 层的状态： 1 2 3 8 0 7 6 4 5	第 16 层的状态： 1 2 3 8 0 7 6 4 5
扩展节点数 （不包括叶子节点）	847	303	9219
生成节点数 （包含叶子节点）	1377	492	14726
运行时间 （迭代次数）	847	303	9219

三. 问题分析

1. 画出[2, 8, 3], [1, 6, 4], [7, 0, 5]推导至[1, 2, 3], [8, 0, 4], [7, 6, 5]的图解。



2. 分析不同的估价函数对 A*算法性能的影响。

A*算法是目前最有影响的启发式图搜索算法，也成为最佳图搜索算法。A*算法的启发函数由估价函数和实际代价函数组成，即 f(n)=g(n)+h(n)，g(n)是从初始结点到 n 结点的实际代价，而 h(n)是从 n 结点到目的结点的最佳路径的估计代价,定义 h*(n)为状态 n 到目的状态的最优路径的代价，则 h(n)≤h*(n)。如果某一问题有解，那么利用 A*搜索算法对该问题进行搜索则一定能搜索到解，并且一定能搜索到最优的解。

在八数码问题中，估价函数有三种：（1）不在位数：现有状态与目标状态不相同的位

数；（2）哈密顿距离：现有状态各数码移动到目标状态所需要移动距离的综合；（3）0：即宽度优先搜索算法，盲目图搜索。启发函数的值越小，优先选择程序越高。通过不在位法和哈密顿距离法与宽度优先搜索比较，采用启发函数优于盲目搜索。当初始状态为 1 2 3 8 0 5 4 6 7，目标状态为 1 2 3 8 0 7 6 4 5 时，搜索深度为 16，此时不在位法扩展节点数位 847，生成节点数位 1377，迭代次数位 847，而哈密顿距离法扩展结点为 303，生成节点数为 492，迭代次数为 303，所以当搜索深度比较高时，相较于不在位法，哈密顿距离法所用时间更短，生成节点数更少，效率更高。

3. 根据宽度优先搜索算法和 A*算法求解 8 数码问题的结果,分析启发式搜索的特点。

宽度优先搜索最大的优点之处在于能够保证找到实际问题的解，但是在搜索的进行过程中必须需要存储大量的状态信息而会导致空间复杂度相当的大，从而容易产生“组合爆炸”的问题。

A*算法是一种启发式的搜索，A* 算法实现的过程中也需要保存一些搜索的状态,但是由于 A* 算法在搜索的过程中每次进行扩展时可以有启发地选择了有最优的节点进行扩展，因此从局部最优推至全局最优，大大的缩小了搜索的时间复杂度和空间复杂度，能够快速找到最优解。不同于 A 算法，A 算法在启发信息给得越多即估价函数值越大,需要搜索处理的状态数就越少,其效率就越高，但也不是估价函数值越大越好,因为估价函数值太大会使 A 算法不一定能搜索到最优解，而使用 A* 算法在求解实际问题过程中一定能够找到最优解。

所以，启发式搜索通过启发信息选取估价函数，大大提高了搜索的效率，降低了时间复杂的和空间复杂度，启发信息给得越多即估价函数值越大,需要搜索处理的状态数就越少,其效率就越高，但估价函数过大容易导致无最优解（A 算法），且获得的启发信息越多，需要耗费计算时间越长，所以可能抵消降低搜索空间复杂度和时间复杂度所带来的益处。

四. 心得体会

通过对宽度优先搜索算法和启发式搜索算法的实际操作和应用，我对搜索算法的流程更加清晰，对于 open 表和 close 表的作用也有了更深刻的理解。启发式搜索相交于宽度优先搜索可以降低搜索过程中的时间和空间复杂度，对于解决计算量巨大、计算周期长的问题具有很大的优势，如围棋、象棋等。另外，通过对比不在位法和哈密顿距离法，我发现，选择合适的启发函数可以进一步提高搜索的效率，启发式的具体选择要根据应用场景具体选择。在本次操作过程中，我深刻的意识到仅仅只学习课本上的知识是远远不够的，我们不仅需要明白底层原理，更需要知道如何通过代码逻辑将其表示出来，即如何实际应用。总而言之，只有将理论知识应用到实践中，才能够真正掌握。

大作业 2：产生式系统设计

一．目的

1. 熟悉知识的表示方法
2. 掌握产生式系统的运行机制
3. 产生式系统推理的基本方法

二．内容

设计并编程实现一个小型产生式系统（如分类、诊断等类型）。

三．要求

1. 具体应用领域自选，具体系统名称自定。
2. 用一阶谓词逻辑和产生式规则作为知识表示，利用产生式系统实验程序，建立知识库，分别运行正、反向推理。

四．系统设计

1. 系统设置，包括系统名称和系统谓词，给出谓词名及定义。

（1）系统名称：动物识别系统

（2）系统功能：对虎、金钱豹、斑马、长颈鹿、企鹅、鸵鸟和信天翁七种动物进行识别

2. 编辑知识库，通过输入规则或修改规则等，建立规则库。

R1：动物有毛发→ 哺乳类

R2：动物产奶 → 哺乳类

R3：动物有羽毛 → 鸟类

R4：动物会飞 ∧会下蛋 → 鸟类

R5：哺乳类∧动物吃肉→ 食肉动物

R6：动物有犬齿 ∧有爪 ∧眼睛盯前方→食肉动物

R7：哺乳类 ∧有蹄 →蹄类

R8：哺乳类 ∧反刍 → 蹄类

R9：哺乳类 ∧ 肉食类∧ 黄褐色 ∧ 有暗斑点→ 金钱豹

R10：哺乳类∧ 肉食类 ∧ 黄褐色 ∧ 有黑色条纹→虎

R11：蹄类 ∧ 长脖 ∧ 长腿 ∧ 有暗斑点→ 长颈鹿

R12：蹄类 ∧ 有黑色条纹→ 斑马

R13：鸟类 ∧长脖 ∧ 长腿 ∧ 不会飞∧黑白二色 →鸵鸟

R14：鸟类 ∧会游泳 ∧黑白二色 ∧ 不会飞 →企鹅

R15：鸟类 ∧善飞 →信天翁

3. 建立事实库（综合数据库），输入多条事实或结论。

编号	描述	编号	描述	编号	描述
0	有毛	12	有暗斑点	24	企鹅
1	产奶	13	有黑色条纹	25	信天翁
2	有羽毛	14	长脖	26	鸵鸟
3	会飞	15	长腿	27	斑马
4	会下蛋	16	不会飞	28	长颈鹿
5	吃肉	17	会游泳	29	虎
6	有犬齿	18	黑白两色	30	金钱豹
7	有爪	19	善飞		
8	眼睛盯前方	20	哺乳类		
9	有蹄	21	鸟类		
10	反刍	22	肉食类		
11	黄褐色	23	蹄类		

4. 运行推理，包括正向推理和反向推理，给出相应的推理过程、事实区和规则区。

（1）正向推理：

- 推理过程：将所有规则进行编号，组成规则库，遍历这些条件，根据用户给出的特征编号，进行匹配，同时计算每个条件的符合程度，推理出的特征加入到综合数据库中，重新遍历条件，更新符合度，如果符合度为 1，则输出推理结果。
- 示例：

  已知事实：产奶，有蹄，有暗斑点，长脖，长腿

  综合数据库：1.产奶，9.有蹄，12.有暗斑点，14.长脖，15.长腿

  从规则库取出规则，与综合数据库中的已知事实匹配

  根据 R2：动物产奶→ 哺乳类，更新综合数据库：

  1.产奶，9.有蹄，12.有暗斑点，14.长脖，15.长腿，20.哺乳类

  根据 R7：哺乳类 ∧有蹄 →蹄类，更新综合数据库：

  1.产奶，9.有蹄，12.有暗斑点，14.长脖，15.长腿，20.哺乳类，23.蹄类

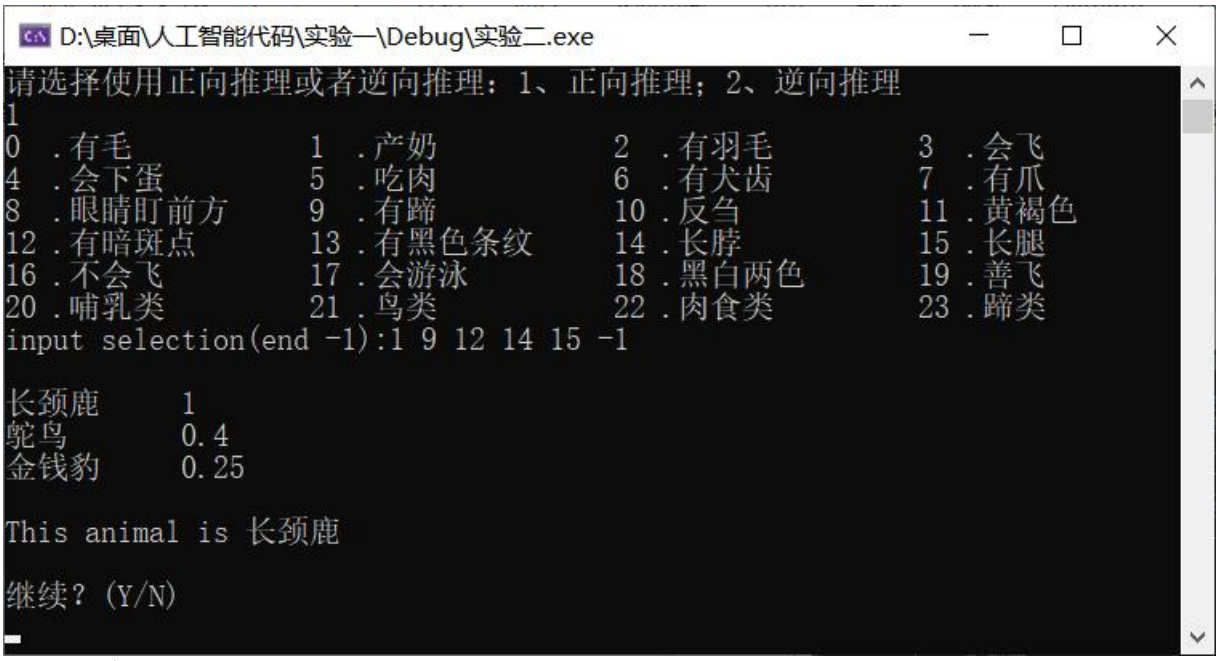
  根据 R11：蹄类 ∧ 长脖 ∧ 长腿 ∧ 有暗斑点→ 长颈鹿，更新综合数据库：

  1.产奶，9.有蹄，12.有暗斑点，14.长脖，15.长腿，20.哺乳类，23.蹄类，

  28.长颈鹿

检查综合数据库中的内容,发现要识别的动物长颈鹿包含在了综合数据库中,所以推出了“该动物是长颈鹿”这一最终结论。至此,问题的求解过程结束。

- 程序运行结果：



（2）逆向推理：

- 推理过程：输入一种动物，询问用户该动物满足的条件，得到条件后，正向推理进行验证。
- 示例：假设动物为虎

  根据规则 R10：哺乳类∧ 肉食类 ∧ 黄褐色 ∧ 有黑色条纹→虎

  运行反向推理，逐条询问：1. 该动物是否具有这个特征：哺乳类？

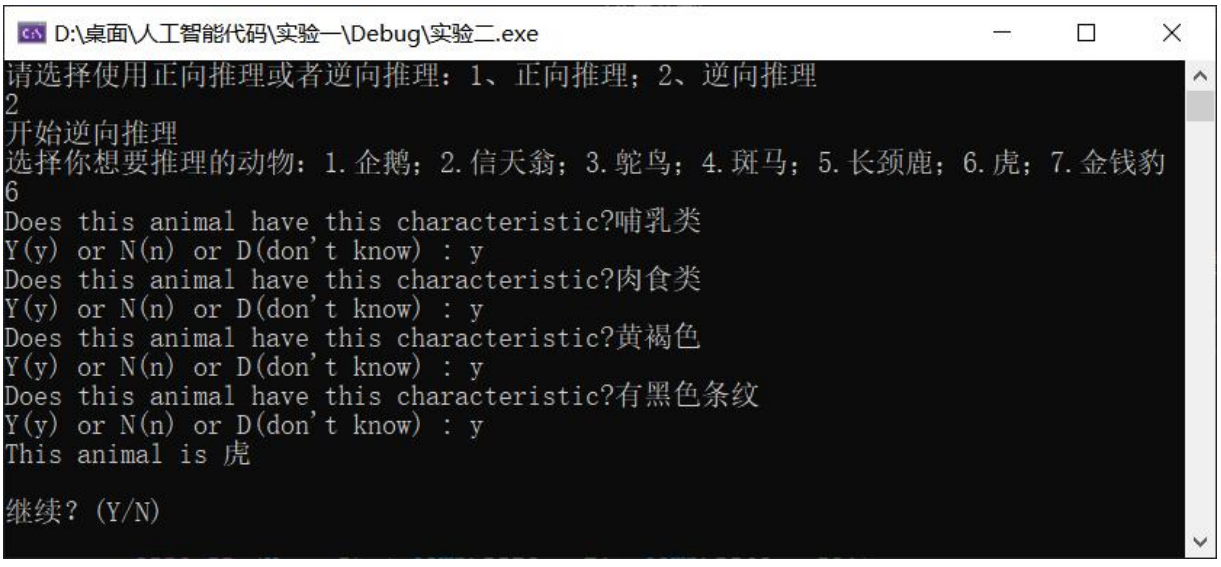
  2. 该动物是否具有这个特征：肉食类？

  3. 该动物是否具有这个特征：黄褐色？

  4. 该动物是否具有这个特征：有黑色条纹？

  用户回答确认，若所有条件都成立，则动物即为虎。

- 程序运行结果：



（3）混合推理：

- 推理过程：将所有规则进行编号，组成规则库，遍历这些条件，根据用户给出的特征编号，进行匹配，同时计算每个条件的符合程度，推理出的特征加入到综合数据库中，重新遍历条件，更新符合度，如果符合度最终不为 1，则进行逆向推理，补充询问可能的且没有在综合数据库中的条件，进行判断，将其加入综合数据库，重新进行正向推理。
- 示例：

  已知事实：黄褐色，哺乳类，肉食类

  综合数据库：11.黄褐色，20.哺乳类，22.肉食类

  从规则库取出规则，与综合数据库中的已知事实匹配

  R9：哺乳类 ∧ 肉食类∧ 黄褐色 ∧ 有暗斑点→ 金钱豹

  R10：哺乳类∧ 肉食类 ∧ 黄褐色 ∧ 有黑色条纹→虎

  由规则 R9 和 R10 知条件并不是全部满足,推出该动物为金钱豹或者虎的概率均为 0.75

  运行反向推理，询问用户补充条件:

  1.该动物是否具有这个特征：有暗斑点？是

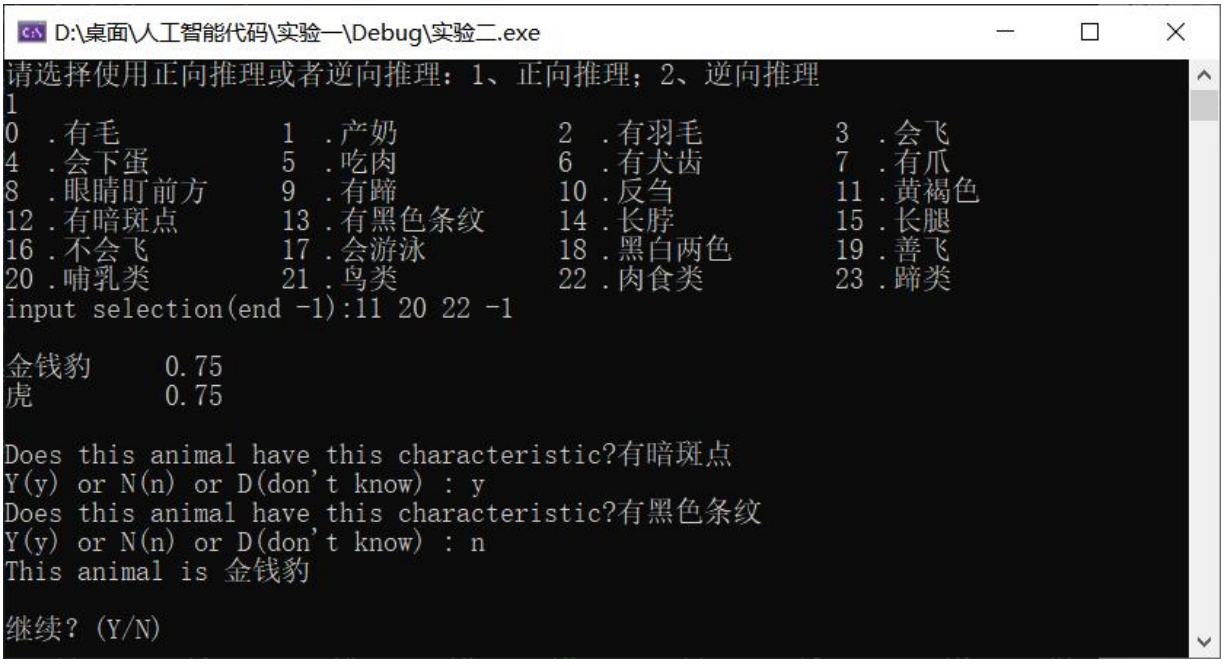
  2.该动物是否具有这个特征：有黑色条纹？否

  得到用户回复后，补充条件：

  综合数据库：11.黄褐色，12.暗斑点，20.哺乳类，22.肉食类

  重新进行正向推理，得出该动物为金钱豹。

- 程序运行结果：



五. 心得体会

在产生式系统的构建过程中，我实际构建了规则库、综合数据库和控制系统，设计了一个小型动物识别系统。在对程序的编写过程中，我遇到了很多问题：进行正向推理时，不知道如何保存推理后的结果，根据上网查找资料，通过数组记录标记将中间结论添加在综合数据库中；进行逆向推理时，更加给定动物推出的条件总存在差异，后面一步一步调试发现是规则和推导结果对应关系错误。

经过本次实际设计操作，我对产生式系统有了更深刻的认识，对正向推理、反向推理和混合推理的逻辑和过程更加清晰，巩固了课本相关知识，提高了我的实际应用能力。在遇到问题时，首先要进行独立思考，解决不了时再上网查找或者请教别人，这样可以提高我们解决问题的能力。

大作业 3：基于遗传算法的优化问题

遗传算法是一种通过模拟自然进化过程搜索最优解的方法，将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程。在求解较为复杂的组合优化问题时，相对一些常规的优化算法，通常能够较快地获得较好的优化结果。

随着网络购物的蓬勃发展，现代快递物流配送系统是一个城市提高其公共运输能力的关键。其中，车辆路径优化问题是城市配送系统（City Delivery System, CDS）的重要环节，通过对车辆行驶路径的优化管理可以有效地降低城市配送成本，同时可以提高城市交通效率，节省公共资源，无论对企业还是对社会，都具有重要的意义。本作业针对 CDS 的车辆路径规划问题，采用遗传算法(Genetic Algorithm, GA)作为优化方法，提出一种配送车辆路径规划的设计方案，以实现车辆路径选取的最优决策。

假设有一个无人配送物流车要送往 N 个快递站，它必须选择所要走的路径，路径的限制是每个快递站只能拜访一次，而且最后要回到原来出发的快递站（假设快递站之间的距离为坐标间的直线距离）。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

一、设计要求

用遗传算法求解不同规模（如 10 个快递站，20 个快递站，50 个快递站）的 CDS 问题，快递站坐标随机生成，不限制实现算法语言（可使用 C++、Python、Matlab 等）。

二、设计思路

1. 种群初始化

个体编码方法有二进制编码和实数编码，在解决 CDS 问题过程中个体编码方法为实数编码。对于 CDS 问题，实数编码为 1-N 的实数的随机排列，初始化的参数有种群规模 M、染色体基因个数 N（即快递站的个数）、迭代次数 C、交叉概率 Pc、变异概率 Pmutation。

2. 适应度函数

在 CDS 问题中，对于任意两个城市之间的距离 D(i,j)已知，每个染色体（即 n 个快递站的随机排列）可计算出总距离，因此可将一个随机全排列的总距离的倒数作为适应度函数，即距离越短，适应度函数越好，满足 CDS 要求,可以使用幂函数变换，加快收敛。

3. 选择操作

（1）轮盘赌法：

- 按适应度比例分配：目前遗传算法中最基本且最常用的算法，各个个体被选择的概率和其适应度值成比例。设群体规模大小为 M，个体 i 的适应度值为 f，则这个个体被选择的概率为

$$p_{si}=\frac{f_i}{\sum_{i=1}^{M} f_i}$$

- 线性排序法：首先按照适应度值对个体进行排序，最差个体排在第 1 位，最优个体排

在第 N 位，根据排位先后，线性地分配给染色体 i 的选择概率：

$$P_i = p_{\min} + (p_{\max} - p_{\min})\frac{i-1}{N-1}$$

所有个体得到不同的排名，分到不同的选择概率，即使它们拥有相同的适应度值。

- 非线性排序：

首先将群体成员按照适应度从好到坏依次排列，并按下式分配选择概率。

$$p_i = \begin{cases} q(1-q)^{i-1}, i = 1,2,...,M-1 \\ (1-q)^{M-1}, i = M \end{cases}$$

其中，q 是一个常数，表示最好的个体的选择概率。

（2）锦标赛竞争法：

从个体中随机选择 k 个个体，将其中适应度最高的个体保存到下一代。这一个过程反复执行，知道保存到下一代的个体数达到预先设定的数量为止。

注：由于 CDS 是最优路径选择问题，故最佳个体保存方法和以上方法结合使用，每次保存适应度最高的个体。

4. 交叉操作

遗传算法中交叉操作有多种方法。本程序对于个体，随机选择两个个体，在对应位置交换若干个基因片段，同时保证每个个体依然是 1-N 的随机排列，防止进入局部收敛。

5. 变异操作

(1)两点互换：随机选取染色体的两个基因进行简单交换

(2)逆转变异：在个体码串中随机选择两点（称为逆转点），然后将两个逆转点之间的基因值以逆向排序插入到原位置中。

(3)移动变异：随机选取一个基因，向左或向右移动一个随机位数。

三、matlab 部分代码

1. 适应度函数 fit.m

```
function fitness=fit(len,m)
fitness=len;
for i=1:length(len)
fitness(i,1)=(1/len(i,1))^m;
end
```

2. 距离计算函数 myLength.m

```
function len=myLength(D,p)%p 是一个排列
[N,NN]=size(D);
len=D(p(1,N),p(1,1));
    for i=1:(N-1)
        len=len+D(p(1,i),p(1,i+1));
    end
end
```

3. 个体选择策略

（1）适应度比例法

```
 p=fitness./sum(fitness);
q=cumsum(p);%累加
for i=1:(M-1)
      len_1(i,1)=myLength(D,popm(i,:));
      r=rand;
      tmp=find(r<=q);
      popm_sel(i,:)=popm(tmp(1),:);
end
[fmax,indmax]=max(fitness);%求当代最佳个体
popm_sel(M,:)=popm(indmax,:);%保存当代最佳个体，以保证不丢失最优个体
```

（2）锦标赛选择

```
 for i=1:(M-1)
      n=0.05*M;
      ran=randperm(M);
      for j=1:n %选出 n 个个体，进行竞争
            fitness_insert(j,1)=fitness(ran(j),1);
            popm_insert(j,:)=popm(ran(j),:);
      end
      [fimax,indexi]=max(fitness_insert);
      popm_sel(i,:)=popm_insert(indexi,:);
end
[fmax,indmax]=max(fitness);%求当代最佳个体
popm_sel(M,:)=popm(indmax,:);%保存当代最佳个体，以保证不丢失最优个体
```

（2）线性排序法

```
 Fit=[fitness,popm];
Fit=sortrows(Fit,1);
Fit(:,1)=[];
pmax=max(fitness)/sum(fitness);
pmin=min(fitness)/sum(fitness);

for i=1:M
      p(i,1)=pmin+(pmax-pmin)*(i-1)/(M-1);
end
p=p/sum(p);
q=cumsum(p);%累加
for i=1:(M-1)
      r=rand;
      tmp=find(r<=q);
      popm_sel(i,:)=Fit(tmp(1),:);
end
[fmax,indmax]=max(fitness);%求当代最佳个体
popm_sel(M,:)=popm(indmax,:);%保存当代最佳个体，以保证不丢失最优个体
```

（3）非线性排序法选择

```
 Fit=[fitness,popm];
Fit=sortrows(Fit,-1);
Fit(:,1)=[]
Fitmax=max(fitness)/sum(fitness);
```

```
for i=1:(M-1)
    p(i,1)=Fitmax*(1-Fitmax).^(i-1);
end
p(M,1)=(1-Fitmax).^(M-1);
q=cumsum(p);%累加
for i=1:(M-1)
    r=rand;
    tmp=find(r<=q);
    popm_sel(i,:)=Fit(tmp(1),:);
end
[fmax,indmax]=max(fitness);%求当代最佳个体
popm_sel(M,:)=popm(indmax,:);%保存当代最佳个体，以保证不丢失最优个体
```

4. 交叉函数 exchange.m

```
function [A,B]=cross(A,B)
%二点交叉
L=length(A);
if L<10
    W=L;
elseif ((L/10)-floor(L/10))>=rand&&L>10%floor 函数向负无穷大方向取整
    W=ceil(L/10)+8;%ceil 朝正无穷大方向取整
else
    W=floor(L/10)+8;
end
%%W 为需要交叉的位数
p=unidrnd(L-W+1);%随机产生一个交叉位置
%fprintf('p=%d ',p);%交叉位置
for i=1:W
    x=find(A==B(1,p+i-1));
    y=find(B==A(1,p+i-1));
    [A(1,p+i-1),B(1,p+i-1)]=exchange(A(1,p+i-1),B(1,p+i-1));
    [A(1,x),B(1,y)]=exchange(A(1,x),B(1,y));
end
end
```

5. 变异函数

（1）互换变异

```
index1=0;
index2=0;
nnper=randperm(size(A,2));%返回矩阵的列数，并随机取 N 个 1-N 之间的数
index1=nnper(1);
index2=nnper(2);
temp=0;
temp=A(index1);
A(index1)=A(index2);
A(index2)=temp;
a=A;
```

（2）逆转变异

```
index1=0;
index2=0;
nnper=randperm(size(A,2));%返回矩阵的列数，并随机取 N 个 1-N 之间的数
```

```
index1=nnper(1);
index2=nnper(2);
C=A;
if index1>index2
    [index1,index2]=exchange(index1,index2);
end
for k=1:(index2-index1+1)
    A(index1)=C(index2);
    index1=index1+1;
    index2=index2-1;
end
a=A
```

（3）移动变异

```
%随机选取一个基因，向右移动一个移动位数。
long=length(A);
index1=0;%随机移动的基因
index2=0;%随机移动的位数
nnper=randperm(size(A,2));%返回矩阵的列数，并随机取 N 个 1-N 之间的数
index1=nnper(1);
index2=nnper(2);
temp=A(index1);%保存这个位点
i=index1;
j=index1+1;
for k=1:index2
    if i>long
        i=mod(i,long);
    end
    if j>long
        j=mod(j,long);
    end
        A(i)=A(j);
    i=i+1;
    j=i+1;
end
index=index1+index2;
if index1+index2>20
    index=mod((index1+index2),long);
end
A(index)=temp;
a=A
end
```

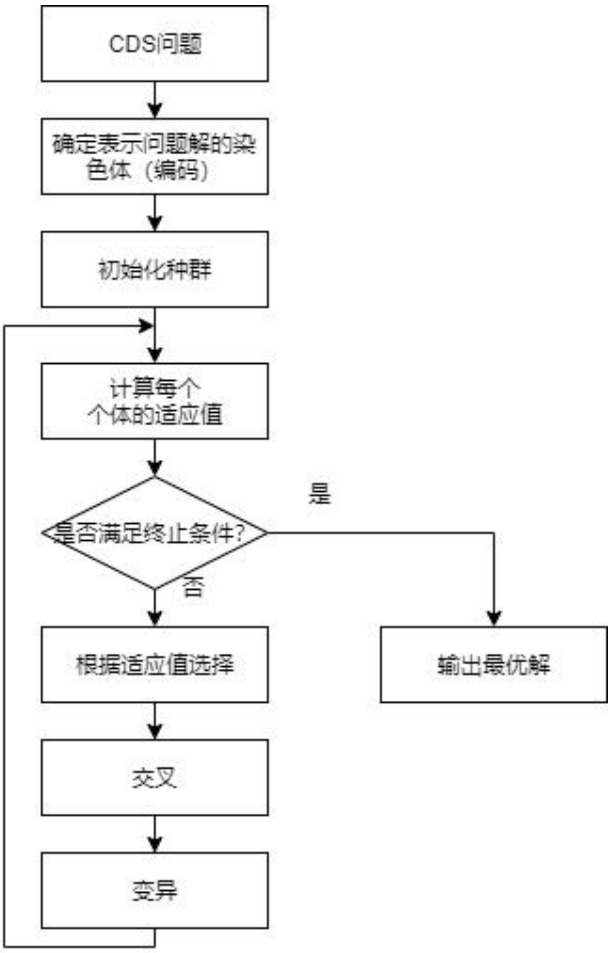
6. 画图函数

```
function plot_route(a,R)
scatter(a(:,1),a(:,2),'rx');
hold on;
plot([a(R(1),1),a(R(length(R)),1)],[a(R(1),2),a(R(length(R)),2)]);
hold on;
    for i=2:length(R)
        x0=a(R(i-1),1);
        y0=a(R(i-1),2);
        x1=a(R(i),1);
```

```
            y1=a(R(i),2);
            xx=[x0,x1];
            yy=[y0,y1];
            plot(xx,yy);
            hold on;
        end
    end
```

四、问题分析

1. 画出遗传算法求解 CDS 问题的流程图。



2. 对比分析遗传算法求解不同规模（不同快递站数量）的 CDS 问题的算法性能。

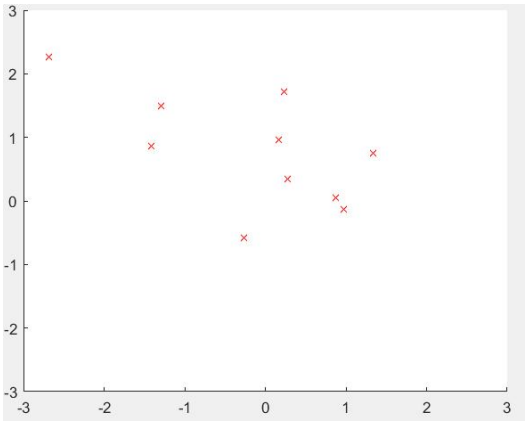
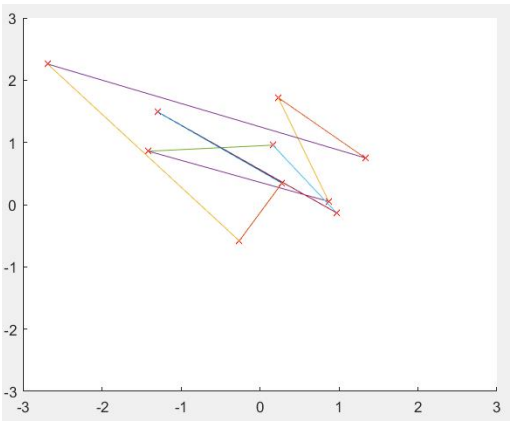
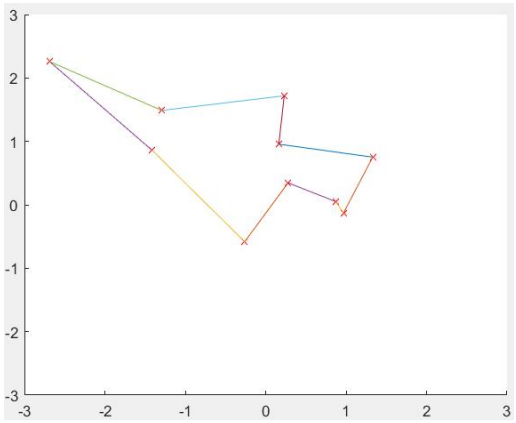
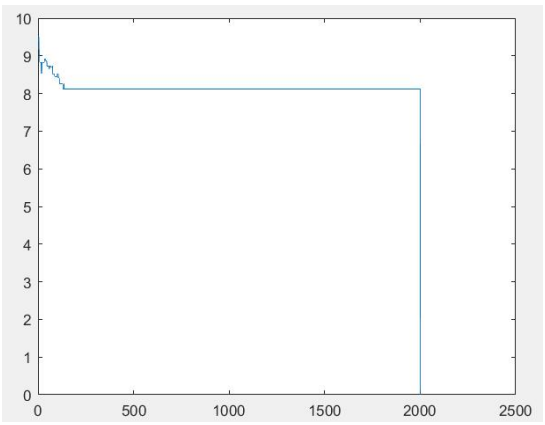
在初始种群数量为 M=100，交叉概率为 Pc=0.85，变异概率为 Pm=0.15 的条件下，迭代次数为 N=2000，改变配送体系规模分别为 10、20、50，求解结果如表 1 所示：

表 1 遗传算法求解不同规模的 CDS 问题的结果

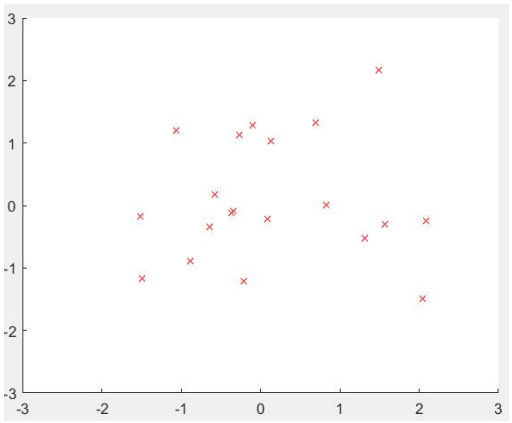
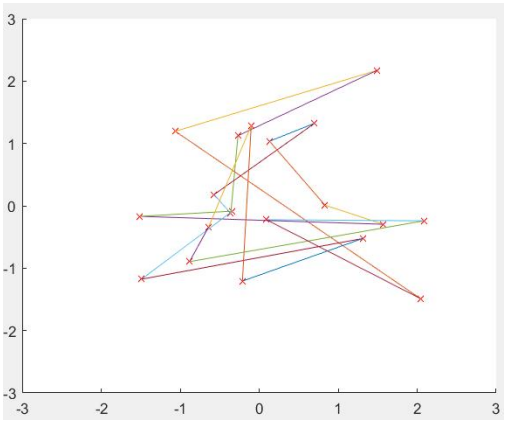
配送体系规模	最好适应度	最差适应度	平均适应度	平均运行时间/ms
10	0.0145	0.0034	0.0115	1
20	0.0027	4.0717e-04	7.4021e-04	1.4
50	0.0013	2.7023e-04	7.1585e-04	1.6

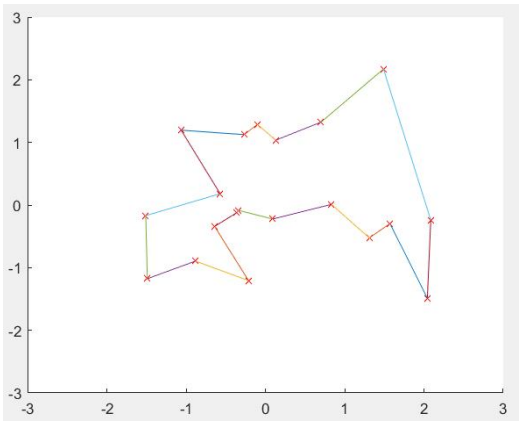
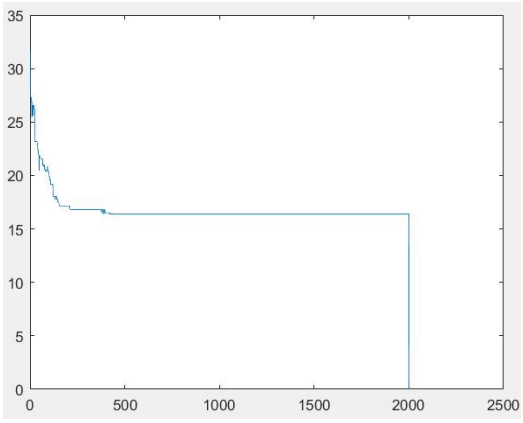
随着配送体系规模增大，最好适应度、最差适应度和平均适应度减小，平均运行时间增长，路径总长达到收敛的迭代次数随之增加。

➢ 当 N=10 时，matlab 仿真结果如下所示：

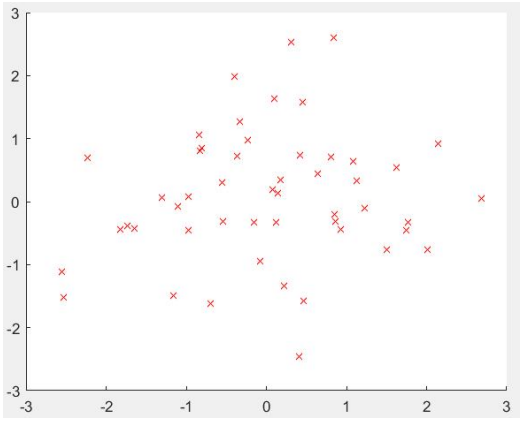
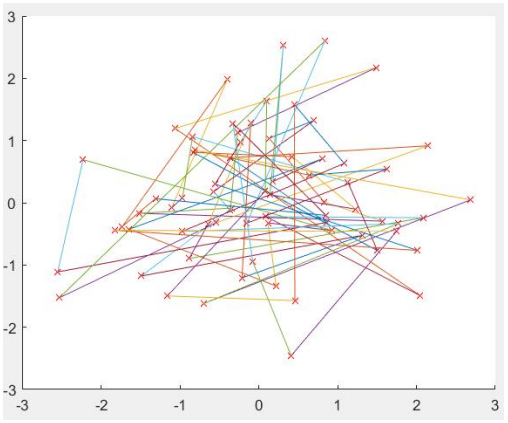
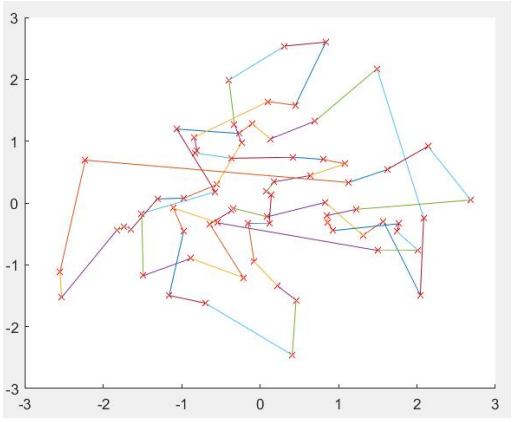
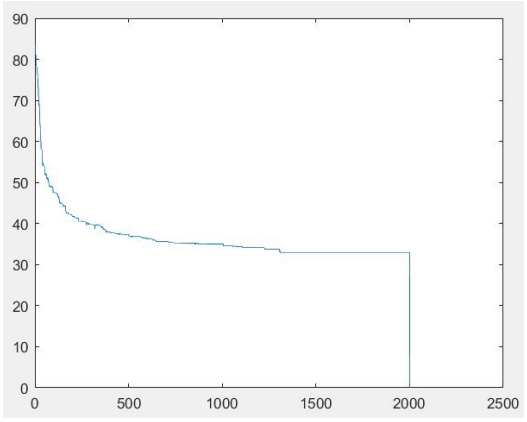
N=10	其中一个初始路径
	
求解结果	**最短路径变化曲线**
	

➢ 当 N=20 时，matlab 仿真结果如下所示：

N=20	其中一个初始路径
	

求解结果	最短路径变化曲线
	

➢ 当 N=50 时，matlab 仿真结果如下所示：

N=50	其中一个初始路径
	
求解结果	**最短路径变化曲线**
	

有以下结论：

- 当配送体系规模较小时，使用遗传算法求解 CDS 问题容易求出最优解，随着配送体系规模增大，求解时间增长，迭代次数增大，适应度值变小，遗传算法的求解性能良好。
- 当配送体系规模较大时，遗传算法求解 CDS 的性能变差，容易陷入局部最优解，如 N=50，根据图示，显然不是最优解。

3. 对于同一个 CDS 问题（快递站数量及对应坐标不变），对比分析种群规模、交叉概率和变异概率对算法结果的影响。

快递站数量 N=20，保持对应坐标不变，迭代次数 C=2000，结果如表 2 所示：

表 2 不同的种群规模、交叉概率和变异概率的求解结果

种群规模	交叉概率	变异概率	最好适应度	最差适应度	平均适应度	平均运行时间/ms
10	0.85	0.15	0.0032	0.0019	0.0027	0.4171
20	0.85	0.15	0.0036	0.0013	0.0030	0.5347
50	0.85	0.15	0.0042	0.0017	0.0035	0.8508
100	0.85	0.15	0.0042	0.0012	0.0029	1.3787
100	0	0.15	0.0033	0.0016	0.0029	0.8819
100	0.5	0.15	0.0035	0.0015	0.0031	1.2271
100	1	0.15	0.0028	0.0008	0.0012	1.4356
100	0.85	0	0.0027	0.0008	0.0012	1.4358
100	0.85	0.5	0.0037	0.0008	0.0014	1.5586
100	0.85	1	0.0031	0.0009	0.0012	1.5749

总结：

- 改变种群个数后的影响：种群规模增大，最好适应度上升，算法结果的精确度会有一定的提升，平均适应度先增加后降低，平均适应度降低在于种群规模增大带来的益处没有规模增长的快，平均运行时间相较之前更久。种群规模不能过低，否则找不到最优解。
- 改变交叉概率后的影响：随着交叉概率的增大，最好适应度和平均适应度先增大后减小，平均运行时间更长。改变交叉概率会直接影响算法的收敛，交叉概率越大，新个体产生的速度就越快。然而，过大的交叉概率对遗传模式被破坏的可能性也越大，使得具有高适应度的个体结构很快就会被破坏；但是如果过小，会使搜索过程很慢，以至停滞不前。交叉概率是影响遗传算法行为和性能的关键所在。
- 改变变异概率概率后的影响：随着变异概率的增加，种群最好适应度和平均适应度先增大后减小，平均运行时间增大。变异概率过低容易陷入局部最优解，变异概率过高容易破坏高适应度个体，变异概率过高或过低都会影响运行得到最优解。

4. 对于同一个 CDS 问题（快递站数量及对应坐标不变），对比分析采用不同的变异策略和个体选择概率分配策略对算法结果的影响

快递站数量 N=20,种群规模为 M=100，保持对应坐标不变，交叉概率为 Pc=0.85，变异概率为 Pm=0.15,迭代次数 C=2000，结果如表 3 所示：

表 3 不同的变异策略和个体选择概率分配策略的求解结果

变异策略	个体选择概率分配	最好适应度	最差适应度	平均适应度	平均运行时间/ms
两点互换	按适应度比例分配	0.0041	0.0012	0.0032	1.3787
两点互换	线性排序法	0.0042	0.0008	0.0033	1.4269
两点互换	非线性排序法	0.0024	0.0009	0.0012	1.4939
两点互换	锦标赛竞争法	0.0042	0.0022	0.0040	2.0798
逆转变异	按适应度比例分配	0.0039	0.0008	0.0029	1.5207
逆转变异	线性排序法	0.0040	0.0009	0.0031	1.5105
逆转变异	非线性排序法	0.0030	0.0007	0.0012	1.5333
逆转变异	锦标赛竞争法	0.0042	0.0028	0.0040	2.1555
移动变异	按适应度比例分配	0.0041	0.0008	0.0013	1.5008
移动变异	线性排序法	0.0040	0.0008	0.0014	1.5166
移动变异	非线性排序法	0.0033	0.0008	0.0012	1.4386
移动变异	锦标赛竞争法	0.0042	0.0027	0.0041	2.0356

在 CDS 问题中，有以下总结：

- 对于按适应度比例分配个体选择概率分配，变异策略两点互换的平均适应度最高，逆转变异的平均适应度较低，而移动变异的平均适应度最低，原因在于两点变异对于个体解的路径顺序改变最小，逆转变异次之，而移动变异对顺序影响最大，可见不同的变异策略对概率选择有一定的影响，实际应用应该选择合适的变异策略。
- 线性排序法的平均适应度比锦标赛竞争法低，比按适应度比例分配高，线性排序克服了适应值比例选择策略的过早收敛和停滞现象，不需要进行适应值的标准化和调节，可以直接使用原始适应值进行排名选择，排序方法比比例方法具有更好的健壮性，是一种比较好的选择方法。
- 非线性排序法由于其概率选择与适应度的大小线性无关，平均适应度在个体选择策略中最低。
- 锦标赛竞争法经过迭代后得到的平均适应度最高，有利于最优解的求解，但是由于每次需要先随机选择部分群体进行竞争，平均运行时间相对于其他策略更长。
- 除锦标赛竞争法外，对于其余的个体选择策略来说，不同变异策略和不同选择分配策略几乎不影响算法的平均运行时间，但是会影响适应度。

五、心得体会

通过本次实际使用遗传算法解决 CDS 问题，我对于遗传算法的基本思路更加熟悉：遗传算法模仿生物的自然进化，通过对生物遗传和进化过程中选择、交叉、变异的模仿，来完成对问题最优解的自适应搜索过程。在遗传算法中，首先确定表示问题解的染色体（编

码），初始化种群，接着计算每个个体的适应值，判断是否满足终止条件，如果满足则输出最优解；否则根据适应值选择个体进行交叉和变异操作产生子代种群，再次计算个体的适应值。当然，纸上得来终觉浅，绝知此事要躬行，仅仅理解是远远不够的，更重要的是实际应用。

在本次最优路径求解问题中，无论是求解规模、种群规模，亦或是交叉概率、变异概率，还是变异策略和个体概率选择策略，对于求解结果的准确度和快速性均有一定程度的影响，这就要求在求解实际问题时，我们要选择合适的参数。不同的求解问题所需要的参数不同，通过调节相关参数，遗传算法的性能会更好。

在本次程序的书写和修改过程中，由于考虑问题不够完善，运行过程中容易出现数组索引错误，经过单步调试输出相关数组，我成功地解决了程序存在的漏洞。这也提醒我编写程序需要尽可能完善地考虑问题，如果出现运行错误，要一步一步的找寻问题所在。另外，当出现解决不了的问题时，可以通过在网上查阅资料拓宽思路。

总而言之，在本次大作业的设计中，我对于遗传算法的理解和实际应用能力都有了一定程度的提升，学习和收获了很多。当然，本次大作业涉及到的算法比较简单，还有很多优化算法和策略方法没有用到，希望之后有机会了解并应用。

大作业 4：基于人工神经网络的图像识别

随着 BP 神经网络和卷积神经网络的快速发展，基于人工神经网络的图像识别技术赋予了人工智能设备“看”的能力。本课程为培养学生以人工智能相关技术基础从事相关应用研究的能力，例如使用 MATLAB 深度学习工具箱或 Pytorch 深度学习框架等深度学习工具训练一个图像识别网络，用于物体分类识别（已提供垃圾分类、猫狗识别、食物识别等数据集）。

一．基于人工神经网络的图像识别领域前沿技术现状

目前，基于神经网络的图像识别是一种比较新型的技术，是以传统图像识别方式为基础，有效融合神经网络算法。针对基于神经网络的图像识别技术，主要有两类：一类是基于 BP 神经网络的图像识别技术，另一类是基于卷积神经网络的图像识别技术。

1.基于 BP 神经网络的图像识别技术

1.1 BP 神经网络发展现状

将 BP 算法用于具有非线性转移函数的三层前馈网络,可以以任意精度逼近任何非线性函数一,这一非凡优势使三层前馈网络得到越来越广泛的应用。然而标准的 BP 算法在应用中暴露出不少内在的缺陷。

1)收敛速度慢;为保证算法的收敛性，学习率 q 必须小于一个上限。这就决定了 BP 算法的收敛速度不可能较快，并且越是接近极小值处，由于梯度变化值逐渐趋于零，算法的收敛速度就越慢;

2)所得的网络容错能力差;

3)算法不完备，易陷入局部极小。不能保证收敛到全局最小点。实际问题的求解空间往往是极其复杂的多维曲面，存在着很多局部极小点，使得陷于局部极小的可能性大大增加，这使权值的初始值选择对网络学习结果有较大的影响，通过随机设置的初始权值经训练而达到全局最优点较难;

4)网络隐含层层数及隐含层单元数的选取尚无理论上的指导．而是根据经验确定。因此，网络往往有很大的冗余性，无形中增加了网络学习的时间。

如果增加BP神经网络隐含层的数量,可以降低误差,提高精度,但同时也使BP网络学习收敛时间变慢，使学习效率下降。BP 算法改进的主要目标是既能加快训练速度和网络收敛速率，又能使程序避免陷入局部极小值,常见的对 BP 网络改进方法有带动量因子算法、自适应学习速率、弹性 BP 算法、LM 算法、正则化方法以及作用函数后缩法等，而对 BP 极小值问题的改进方法有合理选择初始权值和阈值、模拟退火算法、遗传算法和改进激活函数等。

我们通常把网络中的节点比拟成人脑神经元,如同人记实际事物一样,BP 神经网络在样本数据学习中，节点系数权值的改变就像是人脑神经元轴突与树突连接阈值的改变，神经网络会把权系数调整到最佳；从而使分类做到最好。作为一个整体结构, BP 神经网络按整

个对象的特征向量来记忆图像的，只要大多数特征值符合曾学习过的训练样本数据特征，就可识别为同一类别,所以当样本存在较大噪声时神经网络分类器仍可正确识别。在图像识别领域，只要将图像的点矩阵向量作为神经网络的输入，经过网络的计算,BP 神经网络就能输出识别结果。

1.2 BP 神经网络在图像识别中应用

目前，基于 BP 神经网络的图像处理技术已经在各个行业普遍应用，在人脸识别、车牌识别等领域被广泛应用。诸如智能汽车监控中采用的拍照识别技术，若有汽车从该位置经过时，检测设备将产生相应的反应，检测设备启动图像采集装置，获取汽车正反面的特征图像，在对车牌字符进行识别的过程中，就采用了基于神经网络和模糊匹配的两类算法。

2.基于卷积神经网络的图像识别技术

2.1 卷积神经网络在图像识别领域的发展现状及挑战

目前卷积神经网络的设计思路基本上朝着更深，更宽（如 Xception 网络），更多支路，更轻量以及更有效的卷积方式等多个方向同时发展。目前。由于深度学习在图像识别的任务已经超过人类，卷积神经网络能够很好地从输入映射到隐层地特征表达，并且能够层级式地提取特征并通过最后通过内嵌的分类网络完成分类任务。并且卷积神经网络通过轻量话网络或者模型压缩能够在嵌入式或者移动端运行，已慢慢从实验室走向更多的商业化应用。

卷积神经网络在计算机视觉方面的应用，包括图像的分类识别、视频识别等都有着明显的优势，但是图像识别在实际运用中仍然存在一些挑战。

1）图像是识别的基础数据，图像识别中首要的挑战就是模糊图像、受环境影响的图像(如受光线、噪声影响)、遮挡的图像等情况。

2）卷积神经网络在检测中需要对数据进行标注，一个模型的训练过程中往往需要大量手工标注的数据，随着大规模数据量的涌现，无标签的未知数据占绝大多数，为这些数据做标签已经变得不够现实了。我们必须学会从无标注的数据里面进行学习，现有的研究方法包括生成对抗网络、主动学习等.

3）非欧空间数据不存在平移不变性，采用图卷积神经网络可以处理图数据。在该领域中，目前存在的主要方法为谱方法和空间方法，研究表明，虽然图卷积神经网络取得了一定的成果，但仍然有很多问题需要解决。

4）目前在深度学习中训练网络模型需要大量的数据样本，如何在少量样本情况下即保证识别精度又能大大提高网络的训练速度是很多研究者的一个重要目标。数据扩充是一项非常重要的技术，其可以从现有的数据中产生更多的有用数据。神经网络中 Dropout 的引入使得样本不足的条件下也能够比较高的识别率，文献提出采用滑动窗口技术增加训练数据的方法.Chen 等人提出了一种 Grid Mask 的数据扩增策略，该策略删除均匀分布的区域，最后形成网络形状，使用此形状删除信息比设置完全随机位置更加有效.

总之，CNN 目前还存在很多待解决的问题，这些问题不影响在各领域中图像识别的运

用和发展．仍然是研究的一大热点。

2.2 卷积神经网络在图像识别中的应用

在车牌识别中，特征提取和分类识别是图像识别的重点和关键，识别算法主要集中于特征提取任务，一般涉及大量的人工标记工作量和高昂的人工费用，且人工标记效率低。针对人工图像识别存在的弊端，可借助卷积神经网络实现端对端的图像分类识别。近年来，随着车辆识别技术不断发展，新的车牌识别技术不断涌现，如模板匹配法、特征匹配法、机器学习识别法等。卷积神经网络(CNN)是深度学习神经网络的代表算法之一，能够有效将大数据量图片降维为小数据量，在有效保留图片特征的前提下将大数据量图片降维为小数据量，目前已广泛应用于人脸识别、自动驾驶、安防等领域。

如今，由于最新的卷积神经网络在某种程度上解决了计算机视觉领域特征表达的问题，卷积神经网开始在诸多研究方向如目标检测、图像分割、实例分割、图像生成、人脸识别、车辆识别和人体姿态估计等大方光彩，取得的研究成果也是远超传统算法。

二．基于人工神经网络的图像识别的应用课题相关内容

1. 作业内容简介

1.1 目的：

实现在 ciffar-10 数据集上的图像分类,同时该模型也可以用于其他图像的分类识别，只需传入相应的训练集进行训练，保存为另一个模型即可，进行调用使用。

1.2 配置环境：

pycharm(python3.9)，torch，torchvision

1.3 知识预备：

了解 anaconda 环境配置，卷积神经网络的基本结构，torch、torchvision 包的内置函数的使用

2. 算法原理与分析

2.1 卷积神经网络的图像识别原理

卷积神经网络（Convolutional Neural Network，简称 CNN），是一种前馈神经网络，人工神经元可以响应周围单元，可以进行大型图像处理。卷积神经网络包括卷积层和池化层。卷积神经网络是受到生物思考方式启发的 MLPs（多层感知器），它有着不同的类别层次，并且各层的工作方式和作用也不同。

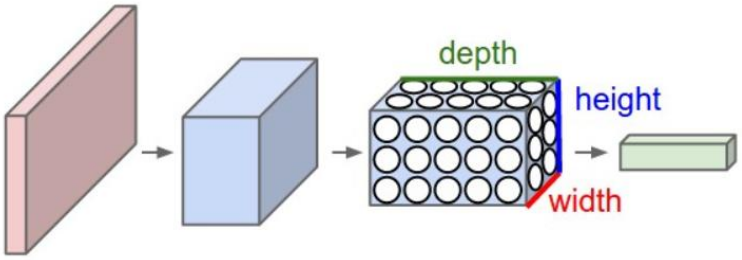


图 4.1 CNN 网络结构

CNN 网络一共有 5 个层级结构：输入层、卷积层、激活层、池化层、全连接 FC 层

（1）输入层

与传统神经网络/机器学习一样，模型需要输入的进行预处理操作，常见的 3 中预处理方式有：去均值、归一化、PCA/SVD 降维等。

（2）卷积层

局部感知：人的大脑识别图片的过程中，并不是一下子整张图同时识别，而是对于图片中的每一个特征首先局部感知，然后更高层次对局部进行综合操作，从而得到全局信息。通过局部感知特性，大大减少了模型的计算参数。但是仅仅这样还是依然会有很多参数。

权值共享机制：同一层下的神经元的连接参数只与特征提取的有关，而与具体的位置无关，因此可以保证同一层中所有位置的连接是权值共享的。

（3）激活层

所谓激励，实际上是对卷积层的输出结果做一次非线性映射。常用的激励函数有：Sigmoid 函数、Tanh 函数、ReLU、Leaky ReLU、ELU、Maxout

（4）池化层

池化：也称为欠采样或下采样。主要用于特征降维，压缩数据和参数的数量，减小过拟合，同时提高模型的容错性。主要有：Max Pooling（最大池化）和 Average Pooling（平均池化）。

（5）全连接层

经过若干次卷积+激励+池化后，终于来到了输出层，模型会将学到的一个高质量的特征图片全连接层。其实在全连接层之前，如果神经元数目过大，学习能力强，有可能出现过拟合。因此，可以引入 dropout 操作，来随机删除神经网络中的部分神经元，来解决此问题。当来到了全连接层之后，可以理解为一个简单的多分类神经网络（如：BP 神经网络），通过 softmax 函数得到最终的输出。

2.2 模型训练原理的分析

BP 算法(即误差反向传播算法)适合于多层神经元网络的一种学习算法，它建立在梯度下降法的基础上。BP 网络的输入输出关系实质上是一种映射关系：一个 n 输入 m 输出的 BP 神经网络所完成的功能是从 n 维欧氏空间向 m 维欧氏空间中一有限域的连续映射，这一映射具有高度非线性。它的信息处理能力来源于简单非线性函数的多次复合，因此具有很强的函数复现能力。这是 BP 算法得以应用的基础。

反向传播算法主要由两个环节(激励传播、权重更新)反复循环迭代，直到网络的对输入的响应达到预定的目标范围为止。

BP 算法的学习过程由正向传播过程和反向传播过程组成。在正向传播过程中，输入信息通过输入层经隐含层，逐层处理并传向输出层。如果在输出层得不到期望的输出值，则取输出与期望的误差的平方和作为目标函数，转入反向传播，逐层求出目标函数对各神经元权值的偏导数，构成目标函数对权值向量的梯量，作为修改权值的依据，网络的学习在

权值修改过程中完成。误差达到所期望值时，网络学习结束。

（1）激励传播

每次迭代中的传播环节包含两步：

- （前向传播阶段)将训练输入送入网络以获得激励响应；
- （反向传播阶段)将激励响应同训练输入对应的目标输出求差，从而获得隐层和输出层的响应误差。

（2）权重更新

对于每个突触上的权重，按照以下步骤进行更新：

- 将输入激励和响应误差相乘，从而获得权重的梯度；
- 将这个梯度乘上一个比例并取反后加到权重上；
- 这个比例将会影响到训练过程的速度和效果，因此称为“训练因子”。梯度的方向指明了误差扩大的方向，因此在更新权重的时候需要对其取反，从而减小权重引起的误差。

2.3 网络架构

本网络模型使用的是经典的 VGG-11 架构。VGG 网络可以分为两部分：第一部分主要由卷积层和汇聚层组成，第二部分由全连接层组成，VGG 网络结构如图：

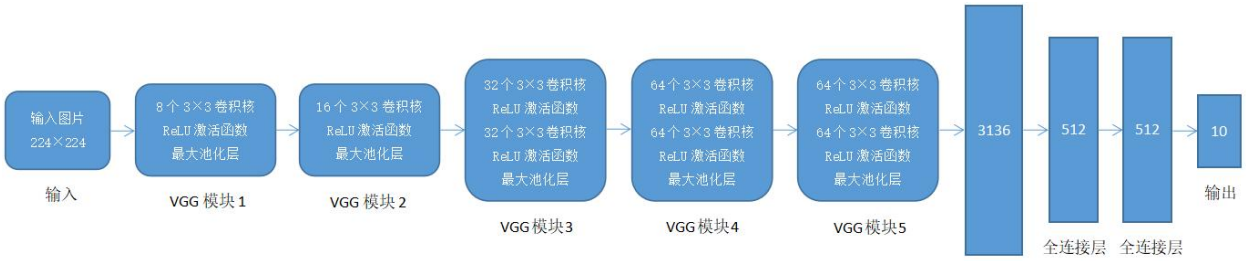


图 4.2 VGG-11 模型结构图

原始 VGG 网络有 5 个卷积块，其中前两个块各有一个卷积层，后三个块各包含两个卷积层。 第一个模块有 64 个输出通道，每个后续模块将输出通道数量翻倍，直到该数字达到 512。由于该网络使用 8 个卷积层和 3 个全连接层，因此它通常被称为 VGG-11，VGG 网络块的代码如下：

```
def vgg_block(num_convs, in_channels, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.Conv2d(in_channels, out_channels,
                                kernel_size=3, padding=1))
        layers.append(nn.ReLU())
        in_channels = out_channels
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)
```

该函数有三个参数，分别对应于卷积层的数量 num_convs、输入通道的数量

in_channels 和输出通道的数量 out_channels.这三个参数由一 List，conv_arch 指定

conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))

下面的代码实现了 VGG-11，通过 for 循环实现对 conv_arch 的提取

```
def vgg(conv_arch):
    conv_blks = []
    in_channels = 3
    # 卷积层部分
    for (num_convs, out_channels) in conv_arch:
        conv_blks.append(vgg_block(num_convs, in_channels, out_channels))
        in_channels = out_channels

    return nn.Sequential(
        *conv_blks, nn.Flatten(),
        # 全连接层部分
        nn.Linear(out_channels * 3 * 3, 4096), nn.ReLU(), nn.Dropout(0.4),
        nn.Linear(4096, 4096), nn.ReLU(), nn.Dropout(0.4),
        nn.Linear(4096, 10))
```

3. 所使用数据集简介

（1）数据集简介

图 4.3 CIFAR-10 数据集示意图

CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。CIFAR-10 的图片样例如图所示：

数据集由 6 万张 32*32 的彩色图片组成，一共有 10 个类别。每个类别 6000 张图片。其中有 5 万张训练图片及 1 万张测试图片。数据集被划分为 5 个训练块和 1 个测试块，每个块 1 万张图片。测试块包含了 1000 张从每个类别中随机选择的图片。训练块包含随机的剩余图像，但某些训练块可能对于一个类别的包含多于其他类别，训练块包含来自各个

类别的 5000 张图片。这些类是完全互斥的，及在一个类别中出现的图片不会出现在其它类中。

一共包含 10 个类别的 RGB 彩色图 片：飞机（ airplane ）、汽车（ automobile ）、鸟类（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、蛙类（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）。图片的尺寸为 32×32 ，数据集中一共有 50000 张训练图片和 10000 张测试图片。

（2）数据集下载以及读取

```
def load_data_cifar_10(batch_size, resize=None):
    """Download the cifar_10 dataset and then load it into memory."""
    trans = [transforms.ToTensor()]
    if resize:
        trans.insert(0, transforms.Resize(resize))
    trans = transforms.Compose(trans)
    mnist_train = torchvision.datasets.CIFAR10(
        root="../data", train=True, transform=trans, download=True)
    mnist_test = torchvision.datasets.CIFAR10(
        root="../data", train=False, transform=trans, download=True)
    return (data.DataLoader(mnist_train, batch_size, shuffle=True,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_test, batch_size, shuffle=False,
                            num_workers=get_dataloader_workers()))

def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):    #@save
    """绘制图像列表"""
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()
    # print("axes 的大小为{},axes 的 type 为{}".format(axes.shape,axes.astype))
    # imgs = imgs.reshape(6,3,32,32)
    for i, (ax, img) in enumerate(zip(axes, imgs)):
        # print('img1 shape = {}'.format(img.shape))
        if torch.is_tensor(img):
            #  图片张量
            if img.ndim == 3:
                img = img.numpy().transpose(1,2,0)
            if img.ndim == 2:
                img = img.numpy()
            ax.imshow(img)
            # print('img2 shape = {}'.format(img.shape))
        else:
            # PIL 图片
            ax.imshow(img)
            # print('is pil')
```

```
            ax.axes.get_xaxis().set_visible(False)
            ax.axes.get_yaxis().set_visible(False)
            if titles:
                ax.set_title(titles[i])
        plt.show()
        return axes
    def get_ciffar_labels(labels):    #@save
        """返回 ciffar 数据集的文本标签"""
        text_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                       'dog', 'frog', 'horse', 'ship', 'truck']
        return [text_labels[int(i)] for i in labels]

    X, y = next(iter(data.DataLoader(mnist_train, batch_size=20)))
    show_images(X.reshape(20, 3, 32, 32), 4, 5, titles=get_ciffar_labels(y))
```

4. 神经网络的模型参数设计

- 输入图片大小 resize=112 × 112
- 通道压缩因子 ratio=0.25
- 迭代次数 epochs=20
- 批量大小 batch_szie=128
- 权重衰退 weight_decay=0.001，
- 动量因子 momentum=0.9
- 初始学习率 base_lr=0.1。
- Droupout=0.4（注：防止过拟合）

5. 神经网络模型的训练与测试

（1）丢弃法

丢弃法是 ImageNet 中提出的一种方法，通俗一点讲就是在训练的时候让神经元以一定的概率不工作，防止过拟合。

（2）学习率指数衰减

指数衰减学习率是先使用较大的学习率来快速得到一个较优的解，然后随着迭代的继续,逐步减小学习率，使得模型在训练后期更加稳定。

（3）SGD 随机梯度下降法

在深度学习中，目标函数通常是训练数据集中每个样本的损失函数的平均值。给定个样本的训练数据集，我们假设$f_i(\mathbf{x})$是关于索引 i 的训练样本的损失函数，其中 x 是参数向量。然后我们得到目标函数:

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$$

x 的目标函数的梯度计算为:

$$\nabla f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x})$$

如果使用梯度下降法，则每个自变量迭代的计算代价会随线性增长。因此，当训练数

据集较大时，每次迭代的梯度下降计算代价将较高。

随机梯度下降（SGD）可降低每次迭代时的计算代价。在随机梯度下降的每次迭代中，我们对数据样本随机均匀采样一个索引 i，并计算梯度$\nabla f_i(\mathbf{x})$，以更新 x，其中η为学习率。

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f_i(\mathbf{x})$$

（4）损失函数

交叉熵的公式如下：

$$H(p,q) = -\sum_{i=1}^{n} p(x_i)\log\ (q(x_i))$$

在深度学习训练网络时，输入数据与标签常常已经确定，那么真实概率分布 P（ x ）也就确定下来了，所以信息熵在这里就是一个常量。由于 KL 散度的值表示真实概率分布 P（ x ）与预测概率分布 Q(x)之间的差异，值越小表示预测的结果越好，所以需要最小化 KL 散度，而交叉熵等于 KL 散度加上一个常量（信息熵），且公式相比 KL 散度更加容易计算，所以在深度学习中常常使用交叉熵损失函数来计算 loss。

交叉熵能够衡量同一个随机变量中的两个不同概率分布的差异程度，在机器学习中就表示为真实概率分布与预测概率分布之间的差异。交叉熵的值越小，模型预测效果就越好。交叉熵在分类问题中常常与 softmax 是标配，softmax 将输出的结果进行处理，使其多个分类的预测值和为 1，再通过交叉熵来计算损失。

（5）训练代码

```
def train(net, train_iter, test_iter, num_epochs, device, loss, optimizer, scheduler=None):
    def init_weights(m):
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(m.weight, mode="fan_out")
            if m.bias is not None:
                nn.init.zeros_(m.bias)
        elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
            nn.init.ones_(m.weight)
            nn.init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            nn.init.normal_(m.weight, 0, 0.01)
            nn.init.zeros_(m.bias)
    net.apply(init_weights)
    print('training on', device)
    net.to(device)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs],
                            legend=['train loss', 'train acc', 'test acc'])
    timer, num_batches = d2l.Timer(), len(train_iter)
    for epoch in range(num_epochs):
        # 训练损失之和，训练准确率之和，样本数
        print("epoch = {}, scheduler = {}".format(epoch, scheduler.get_last_lr()[0]))
        metric = d2l.Accumulator(3)
```

```
        net.train()
        for i, (X, y) in enumerate(train_iter):
            timer.start()
            optimizer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            l.backward()
            optimizer.step()
            with torch.no_grad():
                metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])
            timer.stop()
            train_l = metric[0] / metric[2]
            train_acc = metric[1] / metric[2]
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (train_l, train_acc, None))
        test_acc = evaluate_accuracy_gpu(net, test_iter)
        animator.add(epoch + 1, (None, None, test_acc))
        if scheduler:
            if scheduler.__module__ == lr_scheduler.__name__:
                # UsingPyTorchIn-Builtscheduler
                scheduler.step()
            else:
                # Usingcustomdefinedscheduler
                for param_group in optimizer.param_groups:
                    param_group['lr'] = scheduler(epoch)
    plt.show()
    print(f'loss {train_l:.3f}, train acc {train_acc:.3f}, '
          f'test acc {test_acc:.3f}')
    print(f'{metric[2] * num_epochs / timer.sum():.1f} examples/sec '
          f'on {str(device)}')
if __name__ == '__main__':
    lr, num_epochs, batch_size = 0.1, 20, 128
    optimizer = torch.optim.SGD(net.parameters(), momentum=0.9, lr=lr, weight_decay=1e-3)
    loss = nn.CrossEntropyLoss()
    scheduler = ExponentialLR(optimizer, gamma=0.9)
    train_iter, test_iter = d2l.load_data_ciffar_10(batch_size, resize=112)
    train(net, train_iter, test_iter, num_epochs, d2l.try_gpu(), loss, optimizer, scheduler)
```

6. 实验结果分析与讨论

- 实验结果如图：

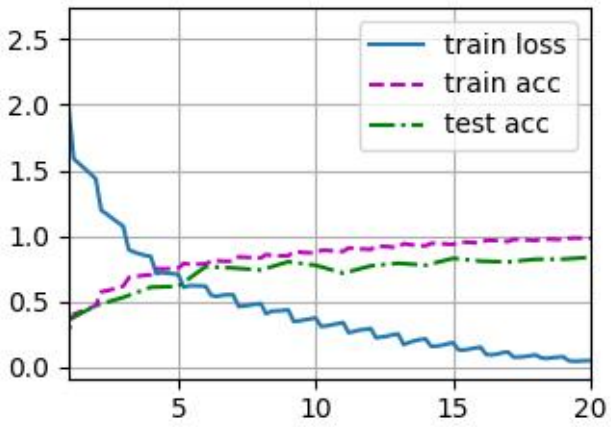


图 4.4 实验结果图示

- 输入卡车图片

预测结果为

```
ReLU output shape:   torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 10])
Files already downloaded and verified
Files already downloaded and verified
tensor([[ 0.3365, -0.8877, -0.3221, -0.2843, -2.2767, -3.2474,  2.1367, -0.0228,
          1.6577,  2.9091]], grad_fn=<AddmmBackward0>)
预测结果为['truck']
```

在经过 20 次迭代后，训练集的准确率为 95%，测试集的准确率为 85%，虽然存在过拟合，但是整体效果还算良好。

三、心得体会

本次通过实际构架 VGG-11 卷积神经网络模型，使用学习率指数衰减、梯度下降和交叉熵函数等进行神经网络训练，我对卷积网络进行图像识别的原理和有了进一步的了解，深入地学习了输入层、输入层、卷积层、激活层、池化层、全连接层的作用，对于神经网络模型的训练过程中的 BP 学习算法理解的更加清楚，对于深度学习的相关概念也更加理解。在本次的实际操作中，我将专业知识用于实践，无论是网络的架构，或者是训练算法的设计，还是相关参数的调试，都提升了我的实践能力。

当然，本次模型训练也存在不足，模型存在过拟合，虽然使用了如丢弃法等方法进行优化，但还是没能很好的解决这一问题，经过上网查资料发现该问题可以通过正则化方法、数据增强以及提前终止法优化，这些将会成为我未来重点学习的反向。

综上所述，在本次大作业中，我实际应用了人工智能课程的相关知识，查阅了人工神经网络在图片识别领域的现状，对人工神经网络的实际应用有了更深刻的认识。

评　　语

对课程大作业的评语:

大作业成绩：	评阅人签名：

注：1、无评阅人签名成绩无效；
　　2、必须用钢笔或圆珠笔批阅，用铅笔阅卷无效。