人工智能实验与阅读报告

班级：计算机2102　姓名：申程宇　学号：2216515195

主要内容：

本次实验包括主观贝叶斯插值法，重排九宫问题以及阅读报告。

目录

人工智能实验—主观贝叶斯方法

一、问题描述

本实验旨在通过实现主观贝叶斯方法，对知识不确定性进行建模和处理。具体而言，通过定义充分性度量LS、必要性度量LN、证据的先验概率PE和结论的先验概率PH等参数，计算了在给定证据或没有给定证据的情况下，假设为真的概率。通过分段线性插值方法实现了证据概率对假设为真概率的影响。

二、实验目的

- 理解和应用贝叶斯定理，特别是在处理主观概率时。根据一些先验概率，根据插值法得到最终的证据不确定情况的后验概率。
- 学习在Python中实现主观贝叶斯线性插值法，并使用matplotlib库来可视化结果。

三、实验内容

1. 使用Python来实现一个简单的主观贝叶斯模型。首先定义一些参数，包括LS（充分性度量）、LN（必要性度量）、PE（证据的先验概率）和PH（结论的先验概率）。然后，计算PH_E（在给定证据的情况下假设为真的概率）和PH_negE（在没有给定证据的情况下假设为真的概率）。
2. 接下来，定义一个函数PH_S，它将根据输入的证据的概率（x）返回假设为真的概率。这个函数是分段的，当x小于PE时，它返回PH_negE和PH之间的线性插值；当x大于或等于PE时，它返回PH和PH_E之间的线性插值。
3. 最后，使用matplotlib库来绘制PH_S函数的图像。可以看到，随着证据的概率的增加，假设为真的概率也会增加。这就是贝叶斯定理的核心思想：当我们有更多支持假设的证据时，我们对假设为真的信念就会增强。
4. 此外，还可以输入一个查询点的横坐标（即证据的概率），然后程序会在图中显示这个点，并打印出对应的纵坐标。这将帮助我们更直观地理解贝叶斯定理是如何工作的。

四、实验原理

1. 知识不确定性的表示：

   在主观Bayes方法中，知识是用产生式规则表示的，具体形式为：

   ```
   IF  E  THEN  (LS,LN)  H (P(H))
   ```

   其中，E是知识的前提条件，既可以是简单条件，也可以是复合条件。
   P(H)是结论H的先验概率，由专家根据经验给出。
   LS称为充分性度量，用于指出E对H的支持程度，取值范围为[0,∞)，其定义为：

$$LS = P(E|H)/P(E|\neg H)$$

LN称为必要性度量，用于指出¬ E对H的支持程度，取值范围为[0,∞)，其定义为：

$$LN = P(\neg E|H)/P(\neg E|\neg H) = (1 - P(E|H))/(1 - P(E|\neg H))$$

LS和LN的值由领域专家给出，代表知识的静态强度。

2. 证据不确定时

   当0<P(E|S)<1时，可以证明:

$$P(H|S) = P(H|E)\times P(E|S) + P(H|\neg E)\times P(\neg E|S)$$

> 当P(E|S)=1时，证据肯定存在，此时P(H|S)=P(H|E)。
> 当P(E|S)=0时，证据肯定不存在，此时P(H|S)=P(H|¬E)。
>
> 当P(E|S)=P(E)时，证据E与观察S无关。

由全概率公式得：

$$P(H|S) = P(H|E)\times P(E) + P(H|\neg E)\times P(\neg E) = P(H)$$

当P(E|S)为其它值时，通过分段线性插值计算P(H|S)，即

$$P(H|S)=\begin{cases} P(H|\neg E)+\dfrac{P(H)-P(H|\neg E)}{P(E)}\times P(E|S) & ,0\le P(E|S)<P(E) \\ P(H)+\dfrac{P(H|E)-P(H)}{1-P(E)}\times[P(E|S)-P(E)] & ,P(E)\le P(E|S)\le 1 \end{cases}$$

该公式称为EH公式。本次实验即使用该分段函数公式，完成主观贝叶斯方法的线性插值。

五、实验结果及分析

实验的自变量包括 `LS, LN, PE, PH`,然而在插值公式中还需要用到 `P(H|E)` 以及 `P(H|¬E)`,因此首先推导二者的表达式如下

$$P(H|E) = LS\times P(H)/[(LS-1)\times P(H)+1]$$
$$P(H|\neg E) = LN\times P(H)/[(LN-1)\times P(H)+1]$$

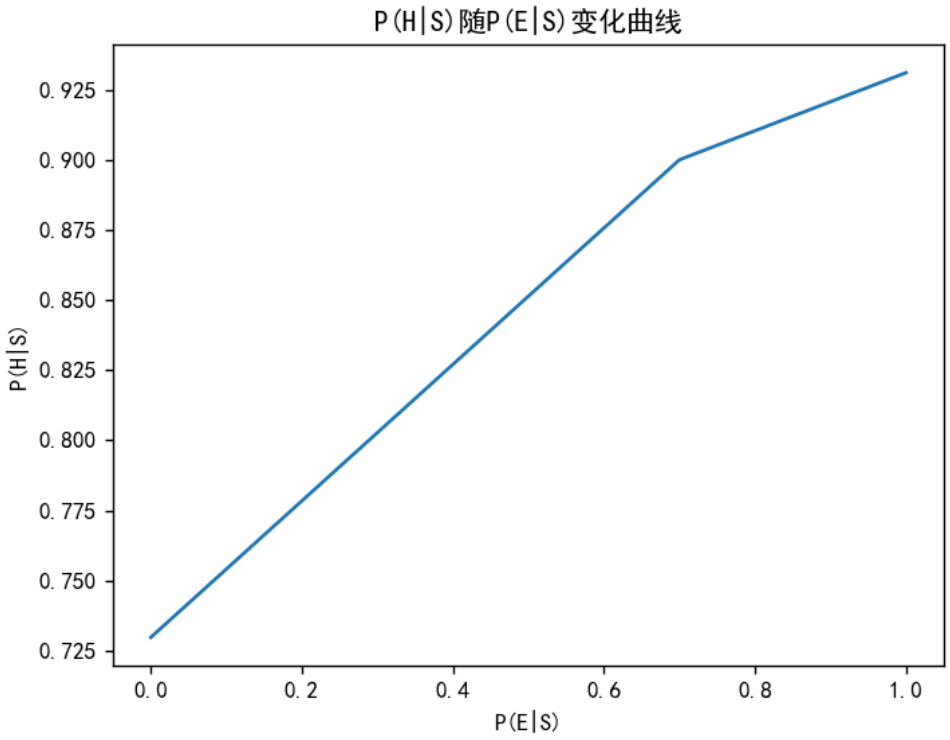
具体推导过程见附录。

用python编写代码如下

```
import numpy as np
import matplotlib.pyplot as plt

# LS,LN,PE,PH
LS, LN, PE, PH = input("请依次输入LS,LN,PE,PH,用空格隔开:").split()
LS = float(LS)
LN = float(LN)
PE = float(PE)
PH = float(PH)
print(f"你输入的值分别是LS={LS},LN={LN},PE={PE},PH={PH}")

PH_E = LS * PH / ((LS - 1) * PH + 1)
PH_negE = LN * PH / ((LN - 1) * PH + 1)


def PH_S(x):
    return np.where(x < PE, PH_negE + (PH - PH_negE) * x / PE, PH + (PH_E - PH)
* (x - PE) / (1 - PE))


plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用SimHei字体
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题

x = np.linspace(0, 1, 300)
```

```
plt.plot(x, PH_S(x))
plt.title("P(H|S)随P(E|S)变化曲线")
plt.xlabel("P(E|S)")
plt.ylabel("P(H|S)")

# plt.show()

x_test = input("请输入查询点横坐标P(E|S):")
x_test = float(x_test)
y = PH_S(x_test)
plt.scatter(x_test,y,color='purple')
plt.annotate(f"({x_test},{y})",(x_test,y))
plt.show()
```

测试数据如下(LS,LN,PE,PH) = (1.5, 0.3, 0.7, 0.9),作图如下

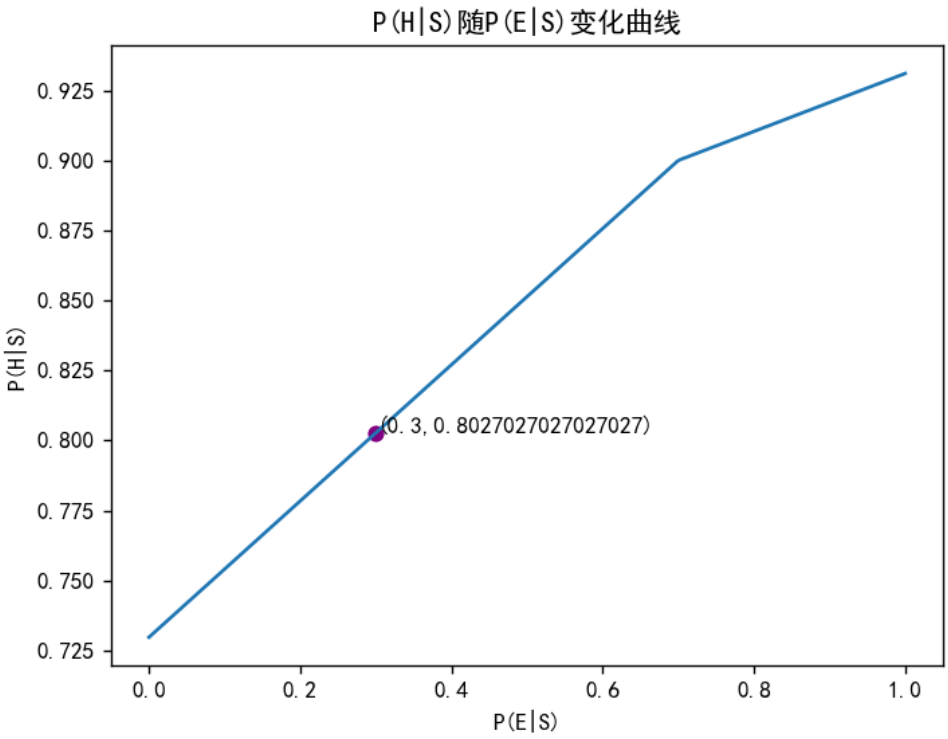


显然，该图像中转折点为P(E)，并且趋势是递增的，即随着证据存在的概率越大（或证据的可信度越高），结论发生的概率越大，符合我们的预期。

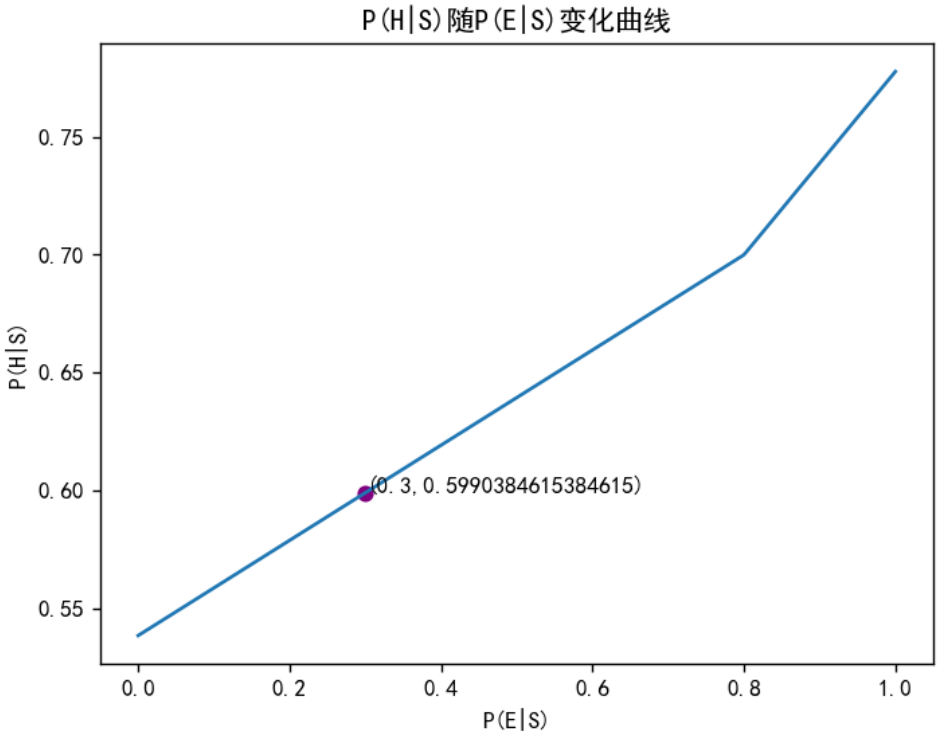
接下来输入一个`P(E|S)`，如下

```
请依次输入LS,LN,PE,PH,用空格隔开:1.5 0.3 0.7 0.9
你输入的值分别是LS=1.5,LN=0.3,PE=0.7,PH=0.9
请输入查询点横坐标P(E|S):0.3
```

输出对应情况的后验概率如图



修改一组参数(1.5, 0.5, 0.8, 0.7)，画出斜率变化不同的情况，并再次查询`P(E|S)=0.3`处，如下

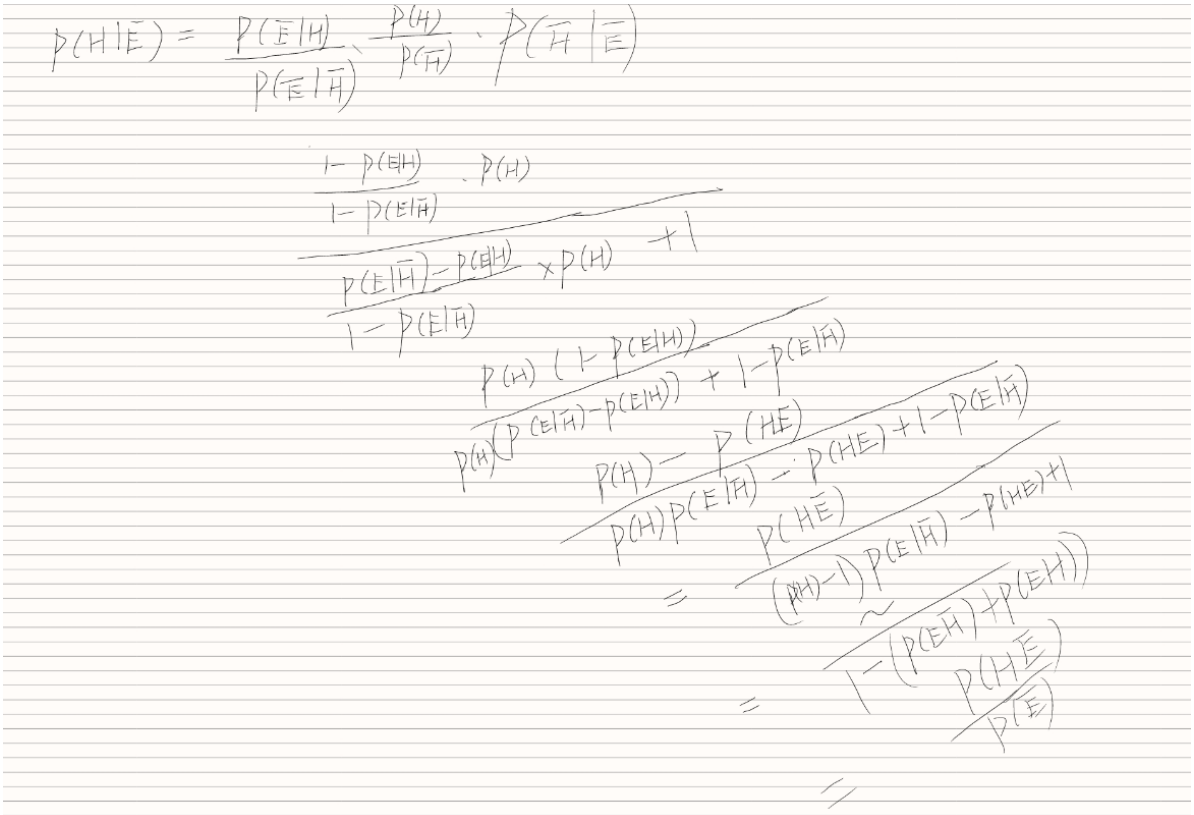


六、实验总结

本次人工智能实验通过主观贝叶斯方法实现了对知识不确定性的建模和处理。通过定义充分性度量LS、必要性度量LN、证据的先验概率PE和结论的先验概率PH，成功计算了在给定或没有给定证据的情况下，假设为真的概率。通过分段线性插值方法构建了PH_S函数，能够根据输入的证据概率实时计算后验概率。通过matplotlib库可视化了PH_S函数，直观展示了证据概率增加时假设为真的概率增强的趋势。整个实验深入剖析了主观贝叶斯方法的原理和应用，加深了对贝叶斯定理在处理不确定性问题中的理解。

附录

`P(H|¬E)` 推导如下，`P(H|E)` 同理。



人工智能实验—重排九宫问题

一、问题描述

九宫重排问题，是在一个3x3的方格盘上放置1-8八个数字宫格和一个空的宫格，开始的数字状态随机给出，通过控制空格的上下左右移动实现数字形成有序的或指定的顺序。

二、实验原理

本实验本质就是一个状态搜索的过程，根据人工智能的内容，一般的搜索过程如下：

> 1. 把初始节点S0放入OPEN表，并建立目前只包含S0的图，记为G；
> 2. 检查OPEN表是否为空，若为空则问题无解，退出；
> 3. 把OPEN表的第一个节点取出放入CLOSE表，并记该节点为n；
> 4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出；
> 5. 扩展节点n，生成一组子节点。把其中不是节点n先辈的那些子节点记做集合M，并把这些子节点作为节点n的子节点加入G中；
> 6. 针对M中子节点的不同情况，分别进行如下处理：（动态更新open表及图G）
>   6.1 对于那些未曾在G中出现过的M成员，设置一个指向父节点（即节点n）的指针，并把它们放入OPEN表；（子节点不在OPEN表）
>   6.2 对于那些先前已经在G中出现过的M成员，确定是否需要修改它指向父节点的指针；（子节点在OPEN表中，或在CLOSE表中）
>   6.3 对于那些先前已在G中出现并且已经扩展了的M成员，确定是否需要修改其后继节点指向父节点的指针；（子节点在CLOSE表中）
> 7. 按某种搜索策略对OPEN表中的节点进行排序；
> 8. 转第2步。

本次笔者选择首先使用**A*算法**实现该问题求解过程，然后横向对比其他算法，常见的算法包括深度优先搜索（DFS）、广度优先搜索（BFS）、A搜索等。搜索算法的选择会影响解的找到速度和内存占用。

三、算法描述

1. **输入阶段**：用户在图形界面输入九宫格的初始状态，输入完成后点击确认按钮。
2. **初始化**：将输入的初始状态转换成一个九宫格状态，用一个3x3的二维数组或列表表示。同时，初始化Astar类的实例。
3. **A*搜索**：调用Astar类的`solvePuzzle`方法，传入初始状态和目标状态。该方法使用A*搜索算法，通过启发式函数（曼哈顿距离）和深度优先搜索，找到从初始状态到目标状态的最短路径。
4. **显示路径**：将搜索得到的移动路径在控制台打印输出，也可在图形界面上显示。
5. **界面刷新**：根据用户输入和搜索路径，刷新图形界面上的九宫格显示，展示每一步的移动过程。

整体流程：

用户在输入框中输入初始状态，点击确认按钮后，算法通过A*搜索找到最优解的移动路径，并在控制台输出移动方向（例如："lrdul"）。同时，图形界面显示初始状态的九宫格，随着移动路径的执行，实时更新显示，直至达到目标状态。

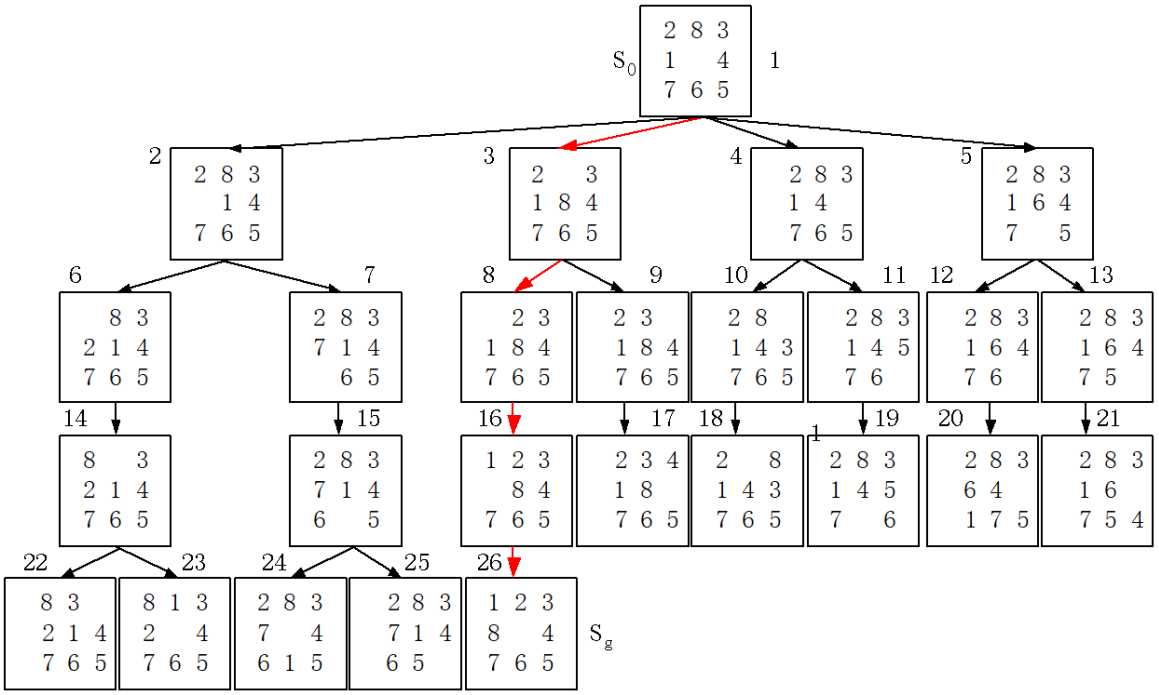
四、实验结果及分析

尝试对样例九宫格进行搜索，样例为"283 104 765",运行代码后输入如下

输出结果如下

```
E:\anaconda\envs\cs231n\python.exe E:\人工智能PPT\大作业\reformation.py
成功找到最优解!
SearchPath-> uldr
```

根据路径作图，画出该搜索空间如下



1. **输入九宫格状态**：初始九宫格状态为"283 104 765"，其中数字0表示空格。
2. **搜索路径**：A*算法找到的最优解路径为"rulddr"，即向右、上、左、下、下、右的顺序移动。
3. **深度图**：深度图显示了每个状态的深度，即从初始状态到该状态的移动步数。深度图中数字表示深度值。
4. **总耗散值**：每个节点都有一个总耗散值（F值），由启发式函数和深度值决定。在深度图中，数字下方的值即为总耗散值。
5. **结果分析**：通过搜索路径"rulddr"，从初始状态"283 104 765"到达目标状态"123 804 765"。在搜索过程中，每一步都选择总耗散值最小的节点进行扩展，直到找到目标状态。路径中的移动顺序是右、上、左、下、下、右。
6. **搜索空间图**：图中展示了搜索空间的一部分，节点之间的连接表示从一个节点到另一个节点的移动关系。颜色深浅表示深度，越深的节点越靠近目标状态。

总体而言，A*算法通过启发式搜索有效地找到了从初始状态到目标状态的最优解路径，实现了九宫格的重排。

五、不同算法性能对比

5.1 A*算法

为了对比不同算法的效率，本次实验从运行时间的角度（即时间效率）上分别对比不同算法，首先是我们已经完成的A*算法，加上计时模块如下

```
start_time = time.time()
test = Astar()
input_map = "".join(numbers)
test.salvePuzzle(input_map, "123804765")
end_time = time.time()
print(f"本次A*算法用时为{end_time-start_time}秒")
```

再次运行，使用相同样例，时间如下

```
E:\anaconda\envs\cs231n\python.exe E:\人工智能PPT\大作业\reformation.py
成功找到最优解!
SearchPath-> uldr
本次A*算法用时为0.0009970664978027344秒
```

本次大约只用了不到0.001秒。

5.2 BFS

接着尝试使用广度优先搜索，运行如下：

```
E:\anaconda\envs\cs231n\python.exe E:\人工智能PPT\大作业\bfs.py
移动路径：  1034
运行时间：  0.00017079999815905467 秒
```

可以看到，广度优先搜索用时比A*算法用时略长，但仍然效率很高。

5.3 DFS

运行深度优先搜索结果如下：

```
E:\anaconda\envs\cs231n\python.exe E:\人工智能PPT\大作业\dfs.py
移动路径： 7852147852147852147852147854785214785214763014785214785214785214785214785478521478521476301478521478521478521478521478547852147852147630147852147852
运行时间： 0.0013534999998228159 秒

进程已结束，退出代码为 0
```

可以看到，运行时间是远大于前两种算法（在该样例中，大概相差10倍），由于是深度优先，因此在搜索过程中可能会在某一个非最优解分支一直向下搜索，因此可以看到，搜索空间是十分庞大的（这里展示了部分，实际更长）。

小结

通过对A*算法、广度优先搜索（BFS）、深度优先搜索（DFS）的实验对比，我们可以得到以下结论：

> ### A*算法
>
> - **优势：** A*算法在样例中表现出色，用时仅为不到0.001秒，具有较高的搜索效率。
> - **原因：** A*算法综合考虑了启发式估值（曼哈顿距离）和实际代价，通过优先级队列（`open`表）筛选节点，以期望更快地找到最优解。
>
> ### 广度优先搜索（BFS）
>
> - **性能：** BFS在样例中用时较短，但略长于A*算法，表现仍然较为良好。
> - **原因：** BFS按照层级逐层扩展搜索空间，因此能够找到最短路径，但在启发式较好的情况下，A*算法有望更快地找到解。
>
> ### 深度优先搜索（DFS）
>
> - **性能：** DFS用时远大于前两者，搜索空间庞大，效率较低。
> - **原因：** DFS采用深度优先的策略，可能沿着某一非最优路径一直搜索下去，导致搜索空间爆炸。在某些情况下可能找到解，但不保证是最优解。
> - **A*算法**综合了启发式信息和实际代价，具有较高的搜索效率，适用于解决最短路径问题。
> - **广度优先搜索（BFS）**通过逐层扩展搜索空间，能够找到最短路径，但在搜索空间庞大时可能效率下降。
> - **深度优先搜索（DFS）**在搜索空间较大时，效率较低，不保证找到最优解。
>
> 在实际问题中，选择适当的搜索算法取决于问题的特性和问题实例。在有启发式信息的情况下，A*算法通常是一个不错的选择。

六、源代码

6.1A*算法代码

```
import tkinter as tk
import time

class Astar:
    def salvePuzzle(self, init, targ):
        # 传入初始状态init,目标状态targ，返回移动路径（string形式）
        open = [(init, self.calcDistH(init, targ))]
        # open表元素(状态，f值)
        closed = {}
        dict_depth = {init: 0}
        # 深度表，用来记录节点是否被访问过及对应的深度
        dict_link = {}
        # 关系表，用来记录父子关系，孩子-->父亲
        dict_dirs = {'u': [-1, 0], 'd': [1, 0], 'l': [0, -1], 'r': [0, 1]}
        # 移动方向，对应二维坐标的变化
        dirs = ['l', 'r', 'u', 'd']
        path = []
        # 用来记录移动路径
        while (len(open)):
            open.sort(key=lambda open: open[1])
            # 每循环一次对列表按由小到大排序一次
            while (open[0][0] in closed):
                open.pop([0])
            # 如果表头元素在closed表中，移出open表
            if (open[0][0] == targ):
                print("成功找到最优解！")
                break
            top = open[0]
            open[0:1] = []
            closed[top[0]] = top[1]
            # 取表头元素，并将表头元素移出open表，压入closed表
            cur_index = top[0].index('0')
            for i in range(4):
                x = cur_index // 3 + dict_dirs[dirs[i]][0]
                y = cur_index % 3 + dict_dirs[dirs[i]][1]
                if (x >= 0 and x < 3 and y >= 0 and y < 3):
                    next_index = x * 3 + y
                    next_state = self.moveMap(top[0], cur_index, next_index)
                    depth = dict_depth[top[0]] + 1
                    # 当前节点不在深度表中，压入深度表和open表，并建立父子关系
                    if ((next_state in dict_depth) == False):
                        dict_depth[next_state] = depth
                        open.append(
                            (next_state, depth + self.calcDistH(next_state,
targ)))
                        dict_link[next_state] = top[0]
                    else:
                        # 当前节点在深度表中且当前深度小于深度表中的值，更新深度表，建立父子
关系
                        if (depth < dict_depth[next_state]):
                            dict_depth[next_state] = depth
                            dict_link[next_state] = top[0]
```

```
                            # 如果当前节点在closed表中，将其移出closed表，将更新后的节点移入open表
                            if (next_state in closed):
                                del closed[next_state]
                                open.append(next_state, depth + \
                                            self.calcDistH(next_state, targ))
        # 循环结束，路径关系全部在dict_link中，从目标状态出发寻找路径
        s = targ
        while (s != init):
            move = s.index('0') - dict_link[s].index('0')
            if (move == -1):
                path.append('l')
            elif (move == 1):
                path.append('r')
            elif (move == -3):
                path.append('u')
            else:
                path.append('d')
            s = dict_link[s]
        path.reverse()
        # 将path逆序(如果想要打印出路径每一步的状态，只需要按照path和init就能实现)
        print("SearchPath->", "".join(path))
        return "".join(path)

    def calcDistH(self, src_map, dest_map):
        # 采用曼哈顿距离作为估值函数
        cost = 0
        for i in range(len(src_map)):
            if (src_map[i] != '0'):
                cost += abs(int(src_map[i]) // 3 - i // 3) + \
                        abs(int(src_map[i]) % 3 - i % 3)
        return cost

    def moveMap(self, cur_map, i, j):
        cur_state = [cur_map[i] for i in range(9)]
        cur_state[i] = cur_state[j]
        cur_state[j] = '0'
        return "".join(cur_state)

def get_input():
    global numbers
    input_text = entry.get()
    numbers = list(input_text)
    if len(numbers) != 9:
        print("请输入9个数字")
        return
    display_grid(frame, numbers)
    # 在这里添加你想在点击按钮后立即执行的代码
    start_time = time.time()
    test = Astar()
    input_map = "".join(numbers)
    test.salvePuzzle(input_map, "123804765")
    end_time = time.time()
    print(f"本次A*算法用时为{end_time-start_time}秒")

def display_grid(frame, numbers):
    for widget in frame.winfo_children():
        widget.destroy()
```

```
    for i in range(3):
        for j in range(3):
            label = tk.Label(frame, text=numbers[3*i+j], width=10, height=5)
            label.grid(row=i, column=j)

if __name__ == "__main__":
    window = tk.Tk()
    window.title('请按次序输入初始状态')
    entry = tk.Entry(window)
    entry.place(relx=0.5, rely=0.1, anchor='center')  # 输入框跨越3列
    button = tk.Button(window, text='确认输入', command=get_input)
    button.place(relx=0.5, rely=0.2, anchor='center')  # 按钮在输入框的右边
    frame = tk.Frame(window)
    frame.place(relx=0.5, rely=0.5, anchor='center')
    window.mainloop()
```

6.2 广度优先搜索代码

```
import time
from collections import deque

def bfs(start, target):
    queue = deque([(start, "")])
    visited = set([start])

    while queue:
        state, path = queue.popleft()
        if state == target:
            return path

        zero_pos = state.index('0')
        for d in [-1, 1, -3, 3]:
            new_zero_pos = zero_pos + d
            if (new_zero_pos >= 0 and new_zero_pos < 9 and
                not (zero_pos == 2 and new_zero_pos == 3) and
                not (zero_pos == 3 and new_zero_pos == 2) and
                not (zero_pos == 5 and new_zero_pos == 6) and
                not (zero_pos == 6 and new_zero_pos == 5)):
                new_state = list(state)
                new_state[zero_pos], new_state[new_zero_pos] =
new_state[new_zero_pos], new_state[zero_pos]
                new_state = "".join(new_state)
                if new_state not in visited:
                    queue.append((new_state, path + str(new_zero_pos)))
                    visited.add(new_state)

    return "无解"

if __name__ == "__main__":
    start_state = "283104765"
    target_state = "123804765"

    start_time = time.perf_counter()
    result = bfs(start_state, target_state)
    end_time = time.perf_counter()
```

```
    print("移动路径：", result)
    print("运行时间：", end_time - start_time, "秒")
```

6.3 深度优先搜索代码

```
import time

def dfs(start, target):
    stack = [(start, "")]
    visited = set([start])

    while stack:
        state, path = stack.pop()
        if state == target:
            return path

        zero_pos = state.index('0')
        for d in [-1, 1, -3, 3]:
            new_zero_pos = zero_pos + d
            if (new_zero_pos >= 0 and new_zero_pos < 9 and
                not (zero_pos == 2 and new_zero_pos == 3) and
                not (zero_pos == 3 and new_zero_pos == 2) and
                not (zero_pos == 5 and new_zero_pos == 6) and
                not (zero_pos == 6 and new_zero_pos == 5)):
                new_state = list(state)
                new_state[zero_pos], new_state[new_zero_pos] =
new_state[new_zero_pos], new_state[zero_pos]
                new_state = "".join(new_state)
                if new_state not in visited:
                    stack.append((new_state, path + str(new_zero_pos)))
                    visited.add(new_state)

    return "无解"

if __name__ == "__main__":
    start_state = "283104765"
    target_state = "123804765"

    start_time = time.perf_counter()
    result = dfs(start_state, target_state)
    end_time = time.perf_counter()

    print("移动路径：", result)
    print("运行时间：", end_time - start_time, "秒")
```

人工智能阅读报告

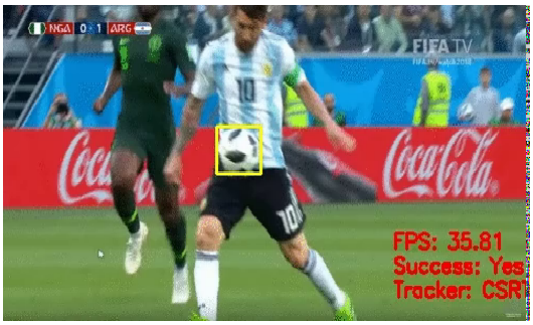
笔者将最近自己复现过的一篇cv领域object tracking的一次汇报工作总结为本次的阅读报告。

本人*github*：*www.github.com/scuuy* 之后可能会上传相关的*pytorch*代码(源代码是*matlab*,打算转写）。

我将从目标追踪的背景出发介绍该领域的相关技术与存在的问题，并解析一个基于高光谱进行跟踪的方法。

论文出处:[material based object tracking]http://arxiv.org/abs/1812.04179

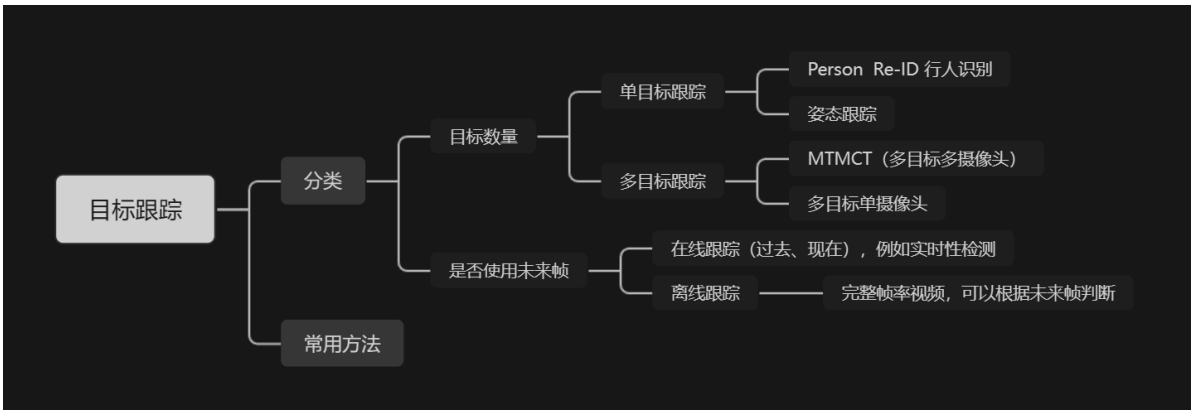
Background



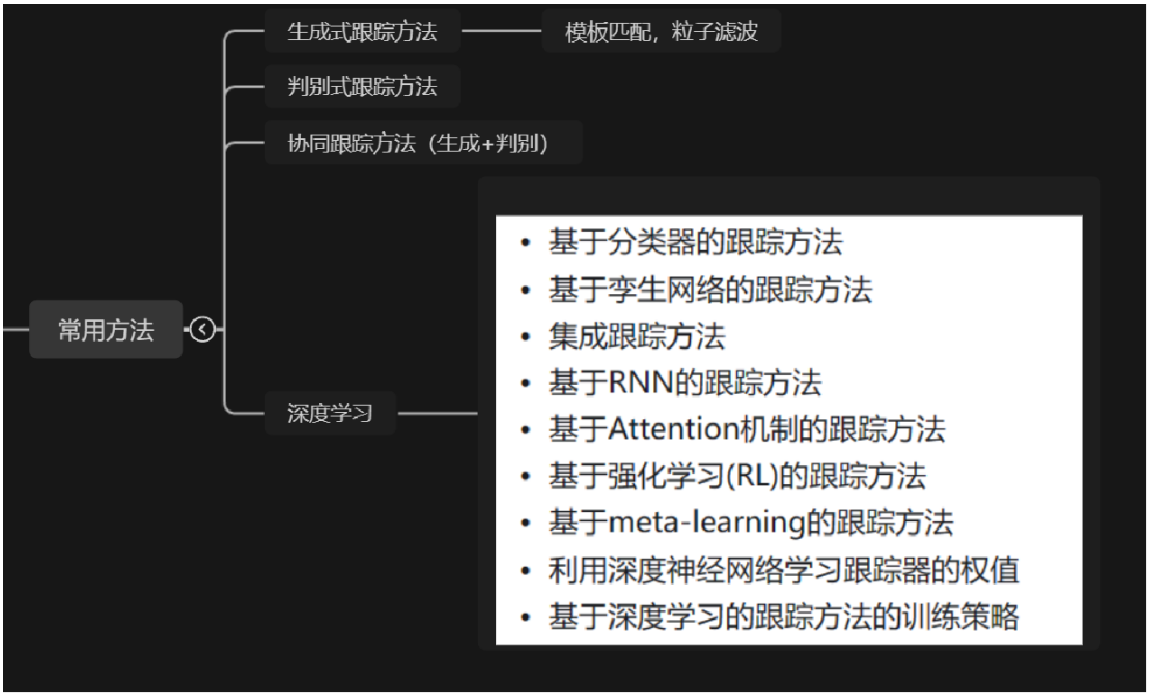
给定一个视频序列，视觉目标跟踪的任务是预测指定目标在该序列中的轨迹，包括目标的位置和大小。目标跟踪可以应用于多种场景，包括视频监控、运动分析、车辆导航、自动驾驶和机器人导航等。在跟踪过程中，跟踪器的性能容易受目标周围环境和目标自身变化的影响，例如目标的外观表示、光线的明暗变化、目标的运动速度、相机的稳定性和目标所处的场景变化等。

笔者复现的文章中基于的问题是：传统的color videos有不可逾越的上限，在一些challenging condition表现糟糕，例如背景混乱，背景和目标相似度高，目标的旋转和变形。传统的目标检测基本基于RGB 3个channel的一般摄像头捕获的图像，这种方式无法利用目标的物质特性，如果能够利用物体的成像光谱特性，获得目标的物质特性，对目标的定位和追踪会更精确。

目前目标跟踪可以进行如下**分类**:

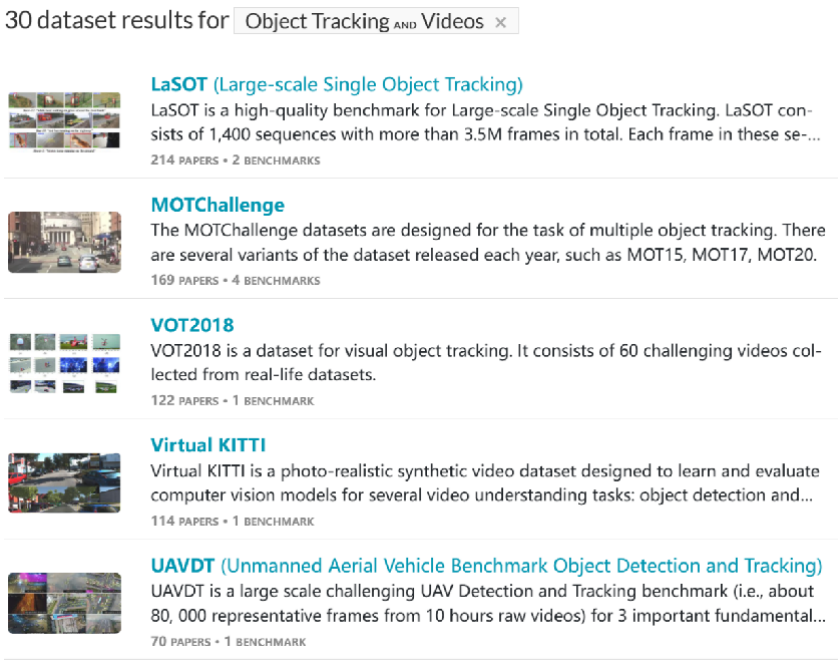


常用的方法如下:



Datasets

目前主流的object tracking数据集如下



Some trouble

> - pose - 姿态变化是目标跟踪中常见的干扰问题。运动目标发生姿态变化时, 会导致它的特征以及外观模型发生改变, 容易导致跟踪失败。例如:体育比赛中的运动员、马路上的行人。
> - scale - 尺度的自适应也是目标跟踪中的关键问题。当目标尺度缩小时, 由于跟踪框不能自适应跟踪, 会将很多背景信息包含在内, 导致目标模型的更新错误:当目标尺度增大时, 由于跟踪框不能将目标完全包括在内, 跟踪框内目标信息不全, 也会导致目标模型的更新错误。因此, 实现尺度自适应跟踪是十分必要的。
> - occlusion - 目标在运动过程中可能出现被遮挡或者短暂的消失情况。当这种情况发生时, 跟踪框容易将遮挡物以及背景信息包含在跟踪框内, 会导致后续帧中的跟踪目标漂移到遮挡物上面。若目标被完全遮挡时, 由于找不到目标的对应模型, 会导致跟踪失败。

> - low resolution - 光照强度变化, 目标快速运动, 低分辨率等情况会导致图像模型, 尤其是在运动目标与背景相似的情况下更为明显。因此, 选择有效的特征对目标和背景进行区分非常必要。
>
> ……

Paper

下面笔者详细分析该**文章的方法**

- 首先，高光谱的信息获取是通过特殊的测量技术，介绍如下：

> 生成式
> 利用物质的光学反射特性。
> example:
> “Time-of-Flight”（ToF）是一种测量技术，通常用于深度感知或距离测量。ToF相机通过发射经调制的近红外光，当光线遇到物体后反射，传感器通过计算光线发射和反射的时间差或相位差，来换算被拍摄景物的距离，以产生深度信息。这种技术可以生成一个深度图像，其中每个像素的颜色代表了该像素对应的物体与相机的距离。
> 这种技术在许多领域都有应用，包括自动驾驶、虚拟现实（VR）、平衡车等。例如，iPhone X的深度感应摄像系统就使用了ToF技术

- 因此，通过对物体光谱特性的分析，可以得到RGB图像无法检测出的物质特性，类似下面这幅图

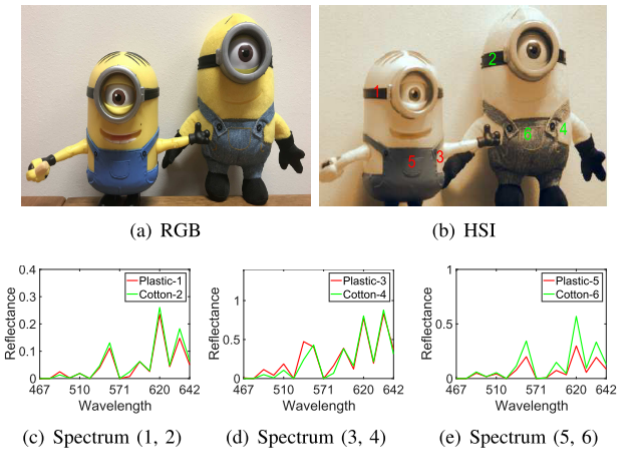


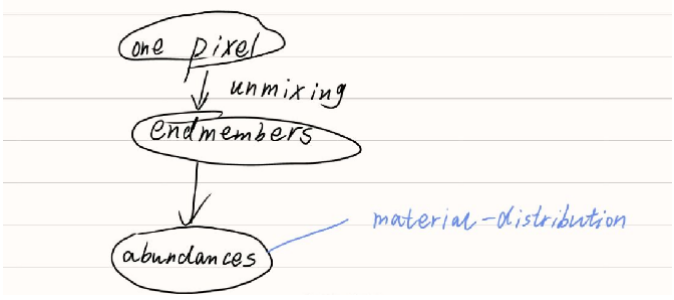
Fig. 1. An example of HSIs for material identification. (a) shows plastic (left) and cloth (right) toys. (b) shows their corresponding false-color image generated from an hyperspectral image. (c)-(e) demonstrate the spectral responses at several pixels. Though these pixels are similar in color, their spectra are different.

在RGB色彩中两个小黄人同样位置的RGB数值几乎相同，但是spectrum光强度有明显差异，因此将HSI图像引入来跟踪目标，并在文章中和RGB方法进行对比。

文中使用到的Framework，具体来说可以分为以下三个步骤：

method-First

two feature extractors （从HSI中提取）



- local spectral-spatial histogram of multidimensional gradients(SSHMG)
  captures the local spectral-spatial texture information in terms of the spatial and spectral gradients orientations
  捕获 光谱-空间纹理 信息（空间和光谱的梯度值）
- spatial distribution
  用abundances表示，abundances像是一个词袋，包含与识别相关的结构特性，是光谱特性的宏观表示。其中abundances通过hyperspectral unmixing获得。

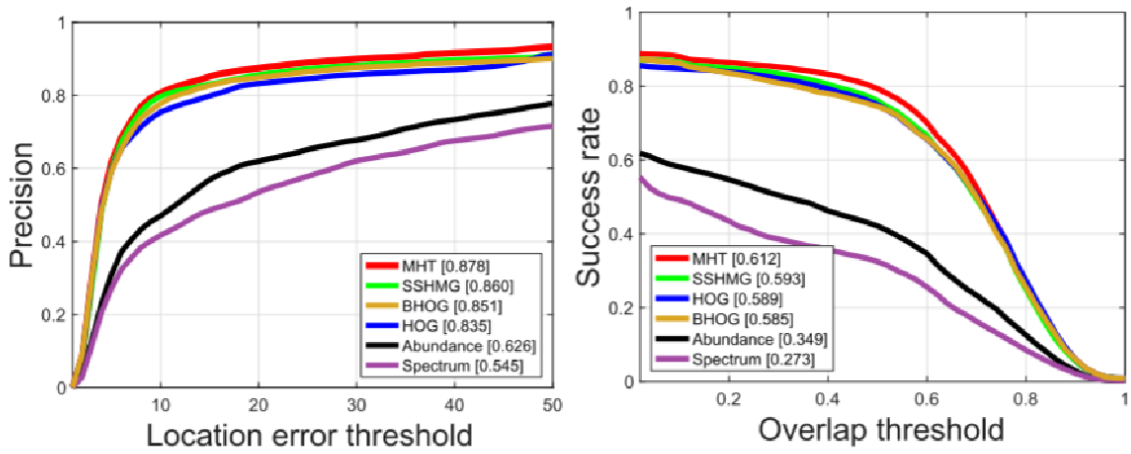
method-Second

评估结果-用全标注的视频dataset
两组视频

- 35个高光谱视频(hyperspectral videos)
  (captured by a commercial high-speed hyperspectral camera. To our best of knowledge, this is the first large-scale fully-annotated hyperspectral video dataset in close-range computer vision setting.)
- 35个同一场景的普通视频(color videos)

method-Third

- extensive experiments，通过对不同特征提取进行对比，类似如下对比



(a) Precision plot　　(b) Success plot

上面图例是一些目前主流的用于object tracking的特征，MHT是本次实验用到的结合基于HSI高光谱的特性曲线，可以看到在这些算法中达到了SOTA效果。

Tracking

具体的追踪过程如下

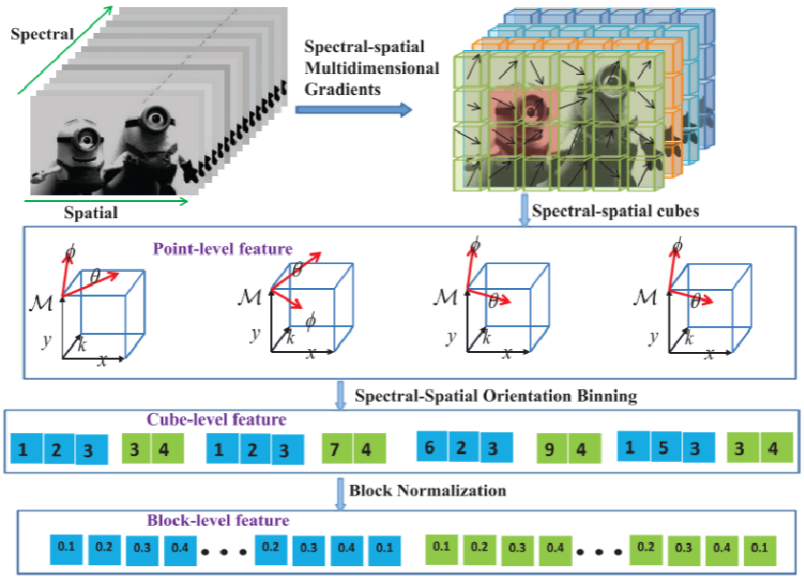
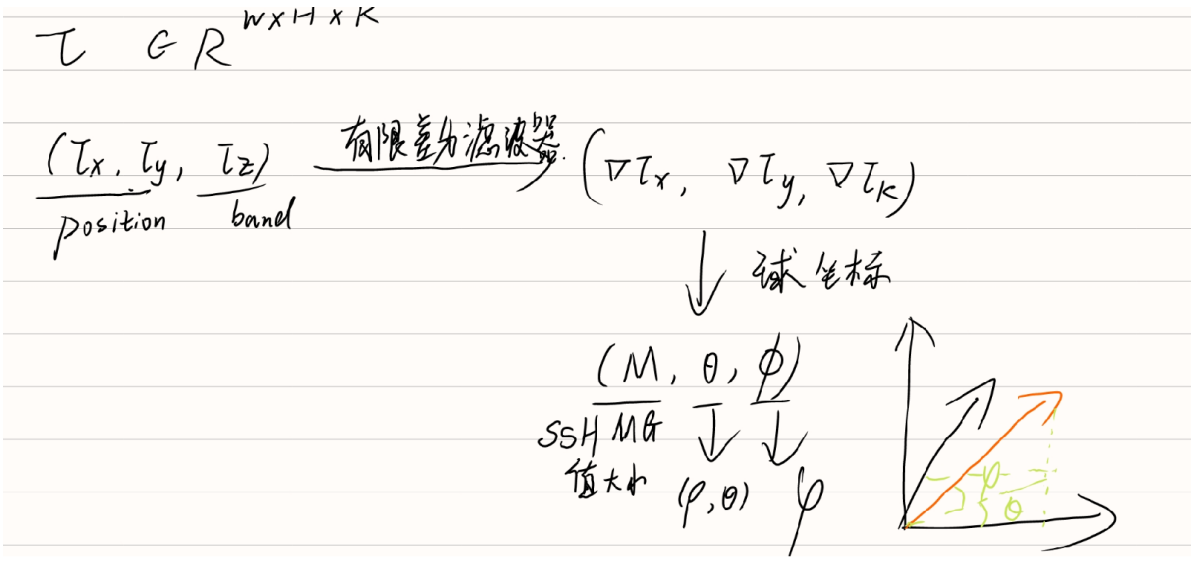
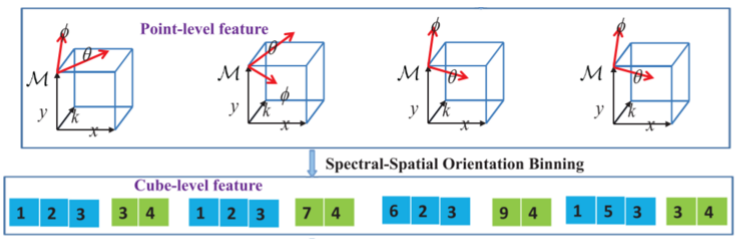


Fig. 3. Framework of the proposed SSHMG. We first calculate the spectral-spatial multidimensional gradients in an HSI in both spatial and spectral directions, which is represented by $(\mathcal{M}, \theta, \phi)$ in a spherical coordinate. After that, all the points in a cube are aggregated in spatial and spectral orientations, yielding cube-level features. Finally the cube-level features are normalized in a local block, resulting block-level features.



笔者做了一定的归纳，如下

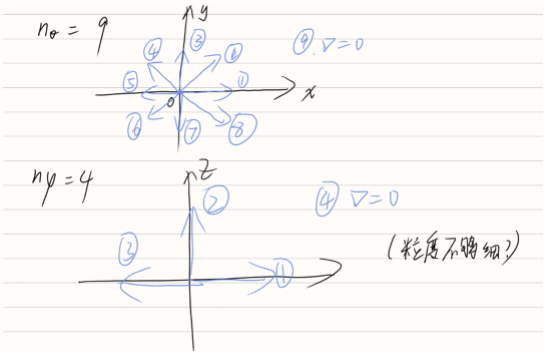


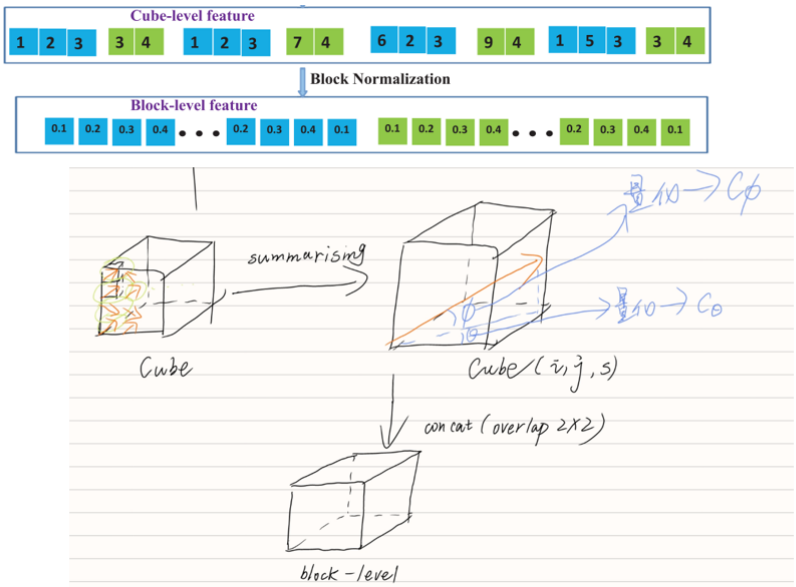
$$B_\theta(x,y,k) = \text{round}\left(\frac{n_\theta \theta(x,y,k)}{2\pi}\right) \bmod n_\theta$$
$$B_\phi(x,y,k) = \text{round}\left(\frac{n_\phi \phi(x,y,k)}{\pi}\right) \bmod n_\phi \quad (2)$$

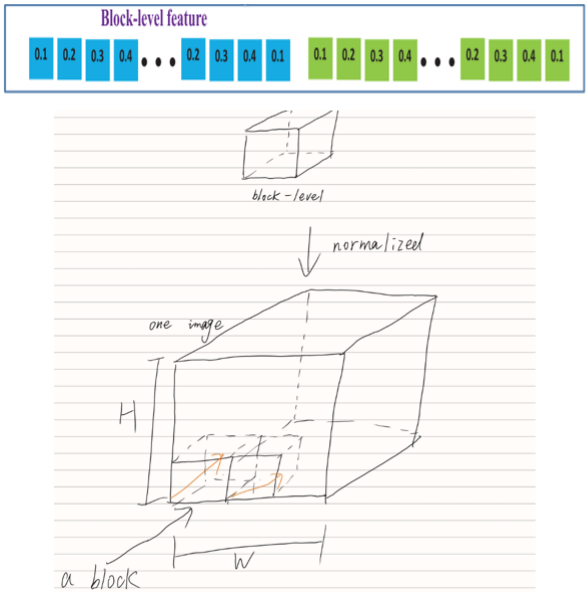
where n_θ and n_ϕ respectively denote the number of bins in the spatial and spectral directions. Based on such quantization, we can define point-level spatial feature $F_\theta(x,y,k)$ and spectral feature $F_\phi(x,y,k)$ for each position as follows:

$$F_\theta(x,y,k)_b = \begin{cases} \mathcal{M}(x,y,k), & \text{if } b = B_\theta(x,y,k) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$F_\phi(x,y,k)_b = \begin{cases} \mathcal{M}(x,y,k), & \text{if } b = B_\phi(x,y,k) \\ 0, & \text{otherwise} \end{cases} \quad (4)$$







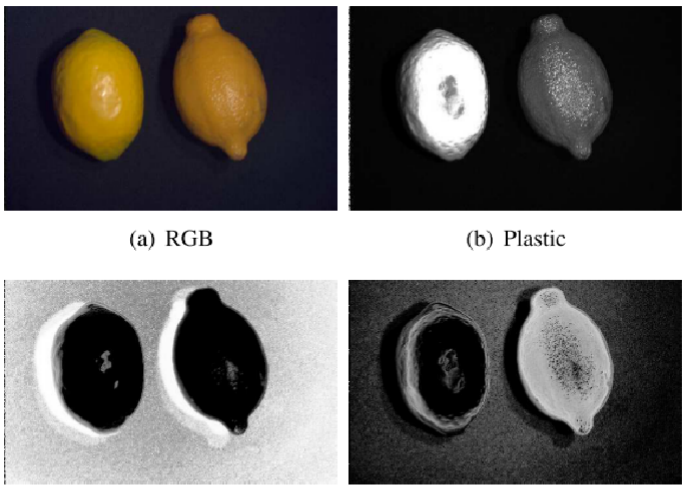
Finally, the block-level features are concatenated in the spectral direction, yielding proposed SSHMG.

此处还要提到一个学习物质分布的过程：

低像素，远距离的视频会导致一些光谱反应特性比较重叠的部分，即mixed pixels
通过unmixing技术，把pixel分解为一些endmembers，然后再回归拼接到每一个像素上，称之为一个abundance
现有的unmixing method有监督，无监督，半监督

- 监督式：分两步，先提取endmember，再整合估计abundance。
  优势：速度快。
  缺陷：复杂情况，纯净的endmember很难得到。
- 无监督式：合为一步，同步估计abundances，但是优化很耗时。
- 半监督式：把endmembers视为可以从光谱库(spectral lib)中预先知道的信息，在lib中做选择。
  优点： 库的信息很真实，接近真实世界的物质信息。（个人认为还有个优点就是这个库还能更新，变成一个API，新材料等。） 该文用的半监督方式，从一个offline library中选择。

效果即下图所示：



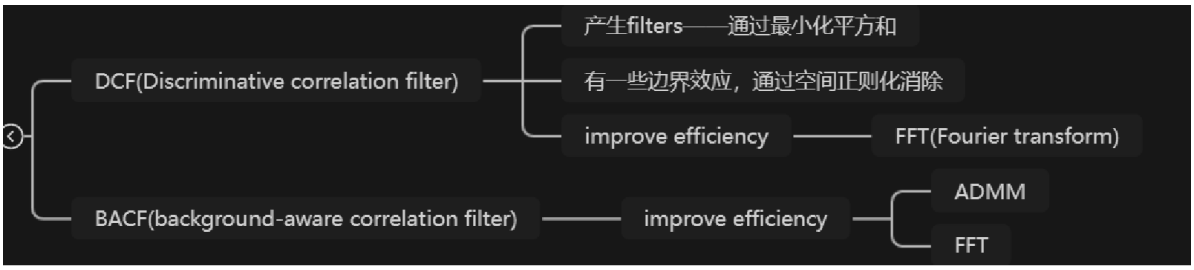
(a) RGB (b) Plastic

(c) Background (d) Real

Fig. 4. Unmixing results of a hyperspectral scene. (a) shows the color images of two lemons. The left lemon is plastic and the right one is real. (b)-(d) give the abundances generated by hyperspectral unmixing.

可以看到，原始的RGB图像对背景的剥离在这种情况下很糟糕，而使用HSI高光谱特性的剥离则很彻底（类似于语义分割）。

本文所用的是BACF基于背景感知的在DCF进一步改进的滤波追踪技术，目前可以分类如下：



Update

文章中对模型参数的更新主要是对滤波器的更新和特征的更新，更新方程如下

$$\min_{\mathbf{f}} \frac{1}{2}\sum_{d=1}^{D}\|\mathbf{y}^d - \sum_{i=1}^{N} w_i \sum_{k=1}^{K_i} \mathbf{f}_k^{\mathrm{T}}\mathbf{P}\mathbf{x}_k^d\|_2^2 + \frac{\lambda}{2}\sum_{i=1}^{N}\sum_{k=1}^{K_i}\|\mathbf{f}_k\|_2^2$$

对于不同特征，为其赋予各自的权值，衡量权值使用`reliability`参数，定义如下：

overlap reliability,overlap部分越多，score越高
distance reliability
距离最终中心位置越近，得分越高
self reliability
窗口移动的平滑度，连续帧位置越接近，窗口尺寸越大，得分越高

$$O(B_i^t, B^t) = \frac{B_i^t \cap B^t}{B_i^t \cup B^t}. \tag{10}$$

Then, the overlap reliability score is determined by:

$$O_i^{'t} = \exp(-(1 - O(B_i^t, B^t))^2) \tag{11}$$

$$D_i^{'t} = \exp\big(- \|C_i^t - C^t\|_2^2\big). \tag{12}$$

$$S_i^{'t} = \exp\left(\frac{-\|C_i^t - C_i^{t-1}\|_2^2}{W(B_i^t) + H(B_i^t)}\right). \tag{13}$$

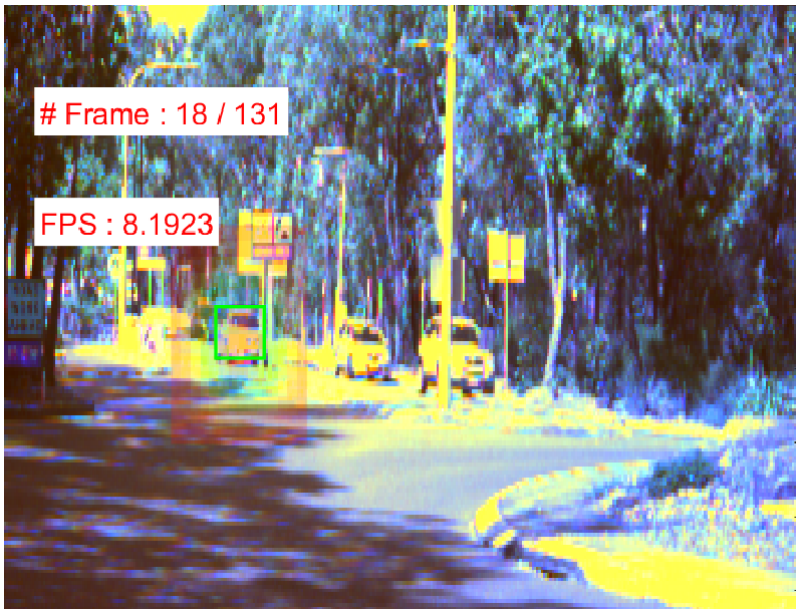
当然，文中还有一部对某特征权值综合的表达式，此处省略。

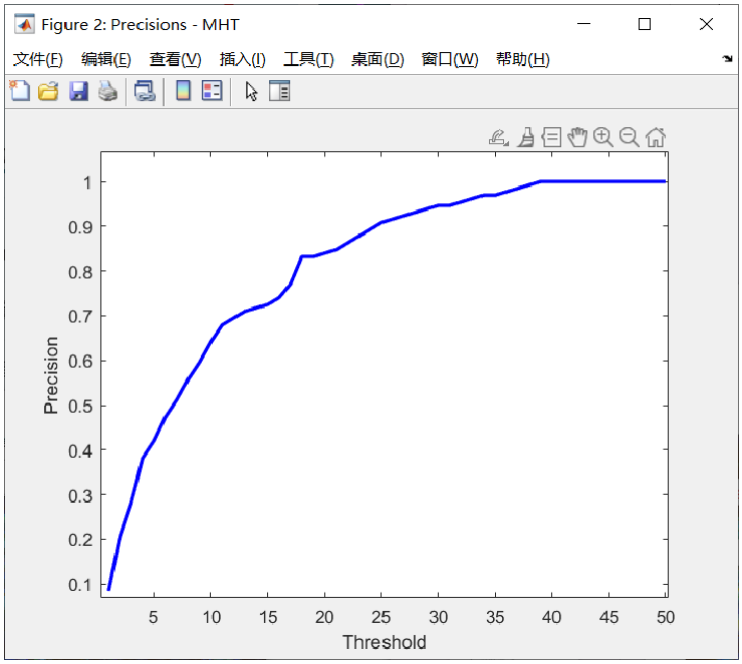
最终模型依据下式更新：

$$\begin{cases} \mathbf{w}_{model}^t = (1-\eta)\mathbf{w}^{t-1} + \eta\mathbf{w}^t \\ \hat{\mathbf{x}}_{model}^l = (1-\eta)\hat{\mathbf{x}}^{l-1} + \eta\hat{\mathbf{x}} \end{cases} \tag{15}$$

Experiments

笔者复现代码后，在具有高光谱特性的Dataset上运行结果如下：





其他视频类似，不再赘述。

Advantage

material based这种追踪很显然精确度会很高，因为根据光谱特性，背景和目标的差异会很大，因此定界是十分精确的，体现在postion以及对目标scale大小的精确度都极高（在一定阈值下）。

然而，数据的获取会比一般的RGB难度和成本更大一些，并且本方法还存在一些潜在的其他问题，这里不再赘述，笔者将在未来继续探索，在原作者的基础上进一步做一些可能的优化。

Appendix

目标追踪综述 https://zhuanlan.zhihu.com/p/152449665
单目标追踪综述 https://zhuanlan.zhihu.com/p/558962899
多目标追踪综述 https://zhuanlan.zhihu.com/p/670383588
多目标追踪参考文献 http://arxiv.org/abs/2209.04796
material based object tracking http://arxiv.org/abs/1812.04179