上海交通大學

程序设计团队项目报告


# 24PointCalculator——数字识别24点计算器

安泰经济与管理学院

2020 Python 程序设计一班

队伍名称：我们的代码都队

队长：侯时杰 519120910202

队员：王翊涵 519120910051

徐芊芊 519120910053

指导教师：鲍杨

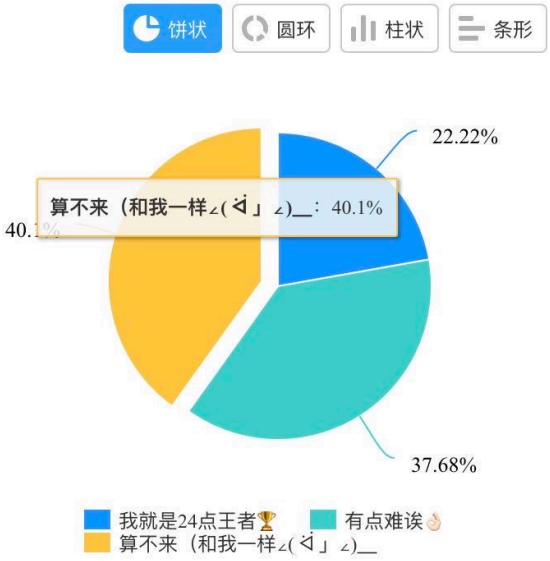
# 一、背景

## 1. 选题原因

24点游戏是一种使用扑克牌来进行的益智类游戏，游戏内容是：从一副扑克牌中抽去大小王剩下52张，任意抽取4张牌，把牌面上的数（A代表1）运用加、减、乘、除和括号进行运算得出24。每张牌都必须使用一次，但不能重复使用，具体规则可以自由随意变化。

虽然大多数24点存在很多解法，有相当一部分数字组合只存在唯一的解法。这种组合往往较有难度，也较为有趣。经过发放问卷进行调查，发现有近八成的被访者表示“24点”游戏具有相当难度。

| 选项 | 小计 | 比例 |
| --- | --- | --- |
| 我就是24点王者🏆 | 46 | 22.22% |
| 有点难诶👌 | 78 | 37.68% |
| 算不来（和我一样∠(ᐛ」∠)__ | 83 | 40.1% |
| **本题有效填写人次** | **207** | |




## 2. 项目构想

本项目使用 python 语言，希望将图像识别技术与经典的数学问题结合，对需解题人所拍摄的手写/印刷体的“24点”题目进行识别，进而进行一系列加法、减法乘法、除法四则运算。该项目核心功能简洁明了，应用情景具有趣味性，通过将本学期我们所学的程序设计知识与数学知识结合运用，充分展现了各学科知识融会贯通的核心素养。

① 数字识别

运用计算机视觉库——“OpenCV”，通过机器学习“MNIST”，“sklearn”识别数字“0～9”，以达到良好的手写数字识别效果。

② 数字生成

对图像数字进行切割，达到识别整串数字的效果。通过“Skimage”讲图片作为numpy数组进行处理。最终输出数字识别的反馈图像并把结果转化为list数据，以进行后续计算。

③ “24点”运算

根据四个数，三个运算符，构造三种中缀表达式，遍历，计算每一种可能，并使用内置的eval函数计算中缀表达式，使得代码尽可能地简介化。

## 3. 项目介绍

### 项目前身——Tesseract方法

Tesseract，一款由HP实验室开发由Google维护的开源OCR（Optical Character Recognition , 光学字符识别），可以识别多种语言与数字，在下载tesseract的基础上，通过安装第三方库pytesseract，可以在python的语言环境下使用。

（下图为对pytesseract的调用）

```
import pytesseract
from PIL import Image


if __name__ == '__main__':
    text = pytesseract.image_to_string(Image.open("0123456.jpeg"))
    print(text)
```

可是经过多次实验，我们发现tesseract的精度并不是特别高，经常会识别错或者识别不出手写数字，且识别的范围局限在标准的印刷体上，不适合我们所做的24点项目。

### 方法简介——HOG方法

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。

具体的实现方法是：在读取图片之后，首先对图片进行灰度化，颜色空间的标准化（归一化）等，以去除图片中噪音的影响；然后将图像分成小的连通区域，即细胞单元，之后采集细胞单元中各像素点的梯度的或边缘的方向直方图，最后把这些直方图组合起来构成特征描述器。

# 二、代码解释

## 1. 手写数字识别学习

由于本组的项目的主要内容为输入一张图片，针对图片上的四个手写数字进行24点的计算，因而不可避免的涉及到手写数字的识别问题，且项目对手写数字的精度要求较高。在做项目的前期，我们有考虑运用Google的tesseract库，但是发现tesseract更适合识别印刷体，且手写体识别的精度较低，不符合要求。在本次数字识别，我组选用了网上已有的MNIST数据集，从官网Yann LeCun's website下载。MNIST数据集包含了七万多张手写的数字图片，来进行更为准确的手写数字机器学习，数据集被分成两部分：60000行的训练数据集（mnist.train）和10000行的测试数据集（mnist.test），其中每张图片包含28像素X28像素。

```python
# Load the dataset
dataset = datasets.fetch_mldata("MNIST Original",data_home='./')

# Extract the features and labels
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')

# Extract the hog features
list_hog_fd = []
for feature in features:
    # fd = hog(feature.reshape((28, 28)), orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1), visualise=False)
    fd = hog(feature.reshape((28, 28)), orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1))
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')
```

（上图为读取MNIST，并用HOG方法识别，生成特征向量的代码截图）

在本次项目中，读取MNIST数据集后，运用HOG方法进行图像识别。根据HOG特征提取方法，先将所检验的图像灰度化，标准化，以抑制噪音的干扰；然后分割图像，计算每个像素的梯度。这里参数orientation指定bin的个数，一共有9个bin, 每20° 一个；pixels_per_cell是指每个cell的像素数，为14*14像素，cell_per_block 表示每个BLOCK内有多少个cell,将block划分为1*1均匀的块；之后每个手写数字产生相应的特征向量，经过机器学习后与它的标签（即数字名称）对应起来。最终，把包含有关于手写数字的HOG特征值的文件保存下来。

## 2. “24点”手写数字识别

```python
    # Load the classifier
    # clf, pp = joblib.load(args["classiferPath"])
    clf, pp = joblib.load("digits_cls.pkl")

    # Read the input image
    # im = cv2.imread(args["image"])
    im = cv2.imread(path)

    # Convert to grayscale and apply Gaussian filtering
    im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    im_gray = cv2.GaussianBlur(im_gray, (5, 5), 0)

    # Threshold the image
    ret, im_th = cv2.threshold(im_gray, 90, 255, cv2.THRESH_BINARY_INV)
```

首先，先加载出在上一步保存的包含HOG特征值的文件，通过引用第三方opencv读取所选择的包含24点的图像，应用灰度化以及高斯模糊尽可能减少图片上的噪音。

```python
    # Find contours in the image
    ctrs, hier = cv2.findContours(im_th.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Get rectangles contains each contour
    rects = [cv2.boundingRect(ctr) for ctr in ctrs]
```

```python
    for rect in rects:
        # Draw the rectangles
        cv2.rectangle(im, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 255, 0), 3)
        # Make the rectangular region around the digit
        leng = int(rect[3] * 1.6)
        pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
        pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
        roi = im_th[pt1:pt1+leng, pt2:pt2+leng]
        # Resize the image
        roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
        roi = cv2.dilate(roi, (3, 3))
        # Calculate the HOG features
        # roi_hog_fd = hog(roi, orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1), visualise=False)
        roi_hog_fd = hog(roi, orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1))
        roi_hog_fd = pp.transform(np.array([roi_hog_fd], 'float64'))
        nbr = clf.predict(roi_hog_fd)
```

在之前运用tesseract库的时候，发现在识别数字1的时候，可能总是因为数字1的笔画较少，特征不太明显，在识别方面有一定的困难，由于本次项目对数字精度要求较高，所以选择检测图像中能够检测到边界的地方画出矩形框，再逐一在每个矩形框中，识别新数字的

HOG向量后，与原有向量的特征值进行对比，从而识别出数字。这样的好处是可以不漏地识别出图像中包含的每一个数字，在矩形中再检测数字可以使得4个数字不一定要横向整齐排列就能识别，只要出现在图像中即可。最后将预测出来的数字显示在图片上的同时也保存在变量中，以便于之后24点算法的应用。

## 3. “24点”运算

```
ion_new.py
for nums in itertools.permutations(cards): # 四个数
    for ops in itertools.product('+-*/', repeat=3): # 三个运算符（可重复！）
        # 构造三种中缀表达式 (bsd)
        bds1 = '({0}{4}{1}){5}({2}{6}{3})'.format(*nums, *ops)  # (a+b)*(c-d)
        bds2 = '(({0}{4}{1}){5}{2}){6}{3}'.format(*nums, *ops)  # (a+b)*c-d
        bds3 = '{0}{4}({1}{5}({2}{6}{3}))'.format(*nums, *ops)  #  a/(b-(c/d))
```

在24点的运算之中，运用类似于穷举法的方法。由于24点的运算比较固定，即一共四个数字，四个数字之间存在加减乘除这四种运算，因而只需要将四个数字的顺序固定，将所有可能的运算符号组合遍历，看看是否存在一个结果趋近于零。这是由于在24点的计算中，可能涉及到分数的计算，因而需要运用浮点数，然而浮点数并不精确，因此在最后的比较中未用0，而是用接近于零的浮点数。

```
        for bds in [bds1, bds2, bds3]: # 遍历
            try:
                if abs(eval(bds) - 24.0) < 1e-10:   # eval函数
                    return bds
            except ZeroDivisionError: # 零除错误!
                continue

return 'Not found!'
```

在最后的比较部分，运用try...except...语言，若运算的结果出现分母为零，使得代码报错的情况，则continue，若算出的所有结果皆没有得到24的数，则返回”Not Found”。

## 4. Python GUI

### ① GUI工具——“Tkinter”

Tkinter模块("Tk 接口")是Python的标准Tk GUI工具包的接口.Tk和Tkinter可以在大多数的Unix平台下使用,同样可以应用在Windows和Macintosh系统里.Tk8.0的后续版本可以实现本地窗口风格,并良好地运行在绝大多数平台中.

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import json
from performRecognition_new import test_func
```

### ② GUI模块创建

- 窗口创建

```
self.win = Tk()
self.win.title('"24Point"Calculator')
self.win.geometry("500x380+300+200")
self.win.configure(bg='mediumorchid')
```

- 选择文件按钮与开始按钮

```
# 打开文件的按钮
Label(self.win ,bg="mediumorchid").place(x=79, y=19, width=142, height=40)


open_btn = Button(self.win, text="Choose a Picture", font="arial 16", command=self.open_img, highlightbackground="mediumorchid",bg="mediumorchid",fg="mediumorchid")
open_btn.place(x=80, y=20, width=140, height=40)

# 开始按钮
Label(self.win, bg="mediumorchid").place(x=299, y=19, width=142, height=40)
start_btn = Button(self.win, text="START", font="arial 16", command=self.start_func,highlightbackground="mediumorchid",bg="mediumorchid",fg="mediumorchid")
start_btn.place(x=300, y=20, width=140, height=40)
```

- 分隔线

```
# 分隔线
Label(self.win, bg="plum").place(x=0, y=80, width=500, height=2)
```

- 结果显示

```
# 结果显示区域
Label(self.win, text="Answer :", font="arial 18",fg ="mediumorchid",bg="mediumorchid").place(x=20, y=85, height=40)
Label(self.win, bg="mediumorchid").place(x=20, y=125, height=100, width=200)
self.out_text = Text(self.win, font="arial 16", highlightbackground="violet",fg="violet", bg="plum")
self.out_text.place(x=20, y=125, width=460, height=240)
```

- 打开文件对话框

```
# 打开文件对话框
def open_img(self):
    self.path = filedialog.askopenfilename(title="打开单个文件",
                filetypes=[("全部文件", "*.*")],
                initialdir="E:\\")
    print(self.path)
```

对于Python GUI 工具Tkinter而言，创建一个适合于某样代码的UI界面分为如下四个步骤：

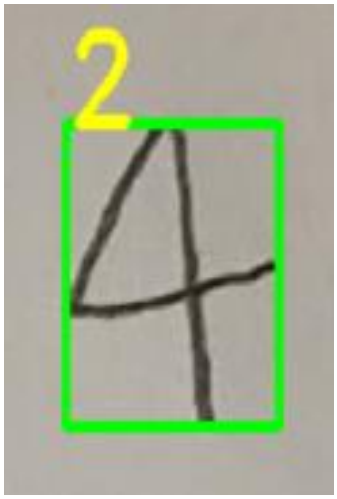
① 界面设计构思

② 控件创建：只要掌握好Tkinter常用控件相关命令及变量设置即可

③ 函数调用：将GUI代码与程序代码相关联
④ 完善与美化

## 三、项目局限

由于MNIST数据集基于西方传统的手写字体，字体特征符合国外的习惯，因而在本项目中，会出现识别错误的情况，如下图将数字4识别成数字2。




此外，由于之前为了防止数字不被检测到，运用了识别轮廓的方法，要求数字的边缘较为平滑。若出现边缘断裂的情况，则可能使得数字识别出现错误。

## 四、分工

1.代码：
徐芊芊—手写数字学习及数字识别编写与改进，代码整合
王翊涵--24点游戏内容及优化，代码整合
侯时杰--GUI界面编写，代码整合

2.Report：
徐芊芊—数字学习及识别代码解释，项目局限
侯时杰—GUI界面代码解释，方案思路
王翊涵--24点游戏代码解释

3.Demo video：
王翊涵—视频制作与剪辑
侯时杰—代码演示录制

4.Pre：
侯时杰，王翊涵，徐芊芊