# 实验心得

## 刘珏 2015201916

# 实验部分

## 第一次小车实验

### 实现功能

### AI Car拼接组装完成

### 超声波传感器功能启用

### 蓝牙模块启用

### 使用蓝牙模块用手机和电脑操控小车

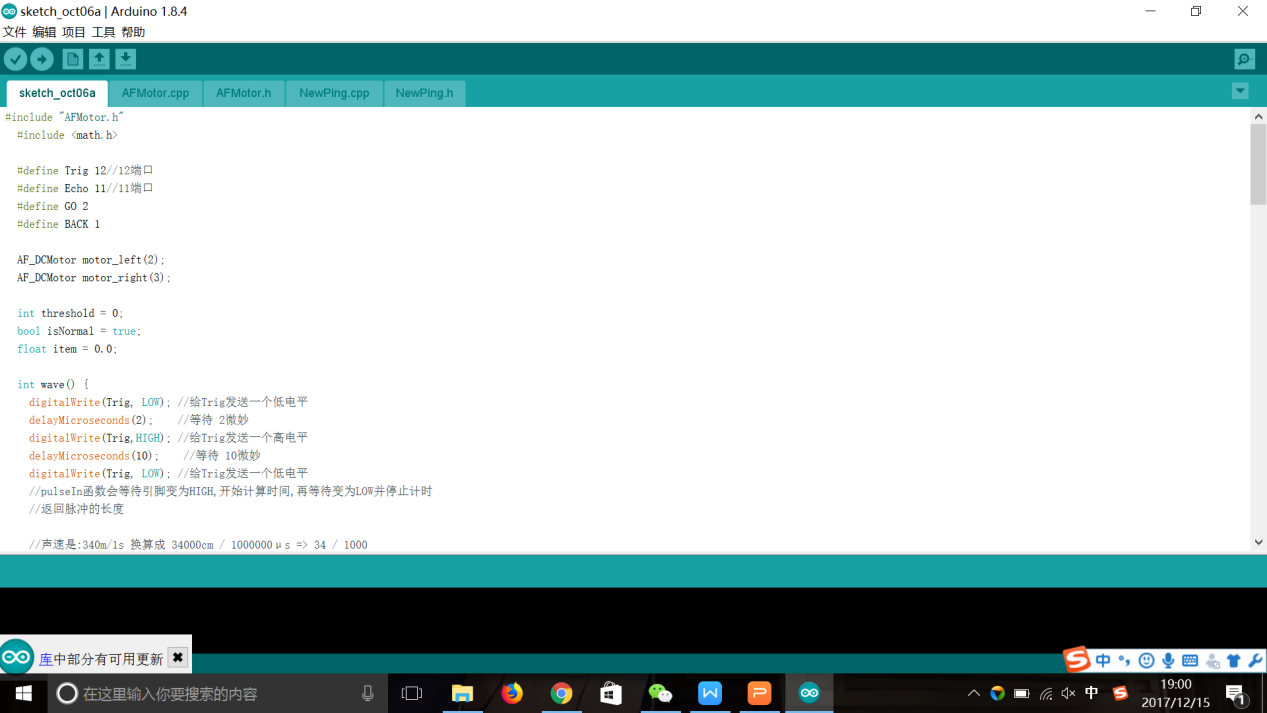
### 基于超声波传感器测距的随机游走

### 功能介绍



### 实验心得

本次实验中我主要负责硬件的部分，由于是第一次接触Arduino，我对Arduino板和Arduino的IDE，以及其变成语言都没有了解，于是我在网上查阅论文相关资料，我们一起根据网上的示例在面包板上接线，连接红外装置，还下载了Arduino的IDE，并学习了Arduino语言的基础语法。



尽管由于分工的不同我没有参与到程序的编写当中，我也了解了Arduino的原理并且知道了如何编写Arduino程序。这次的任务相对简单，但是为后续对小车功能的进一步完善打下了基础。

## 第二次小车实验

### 实现功能

* 将手机摄像头捕捉到的影像实时传输到电脑
* 图像识别
* 目标跟踪
* AI Car捕捉移动中小鼠
* 画出小鼠的运动轨迹

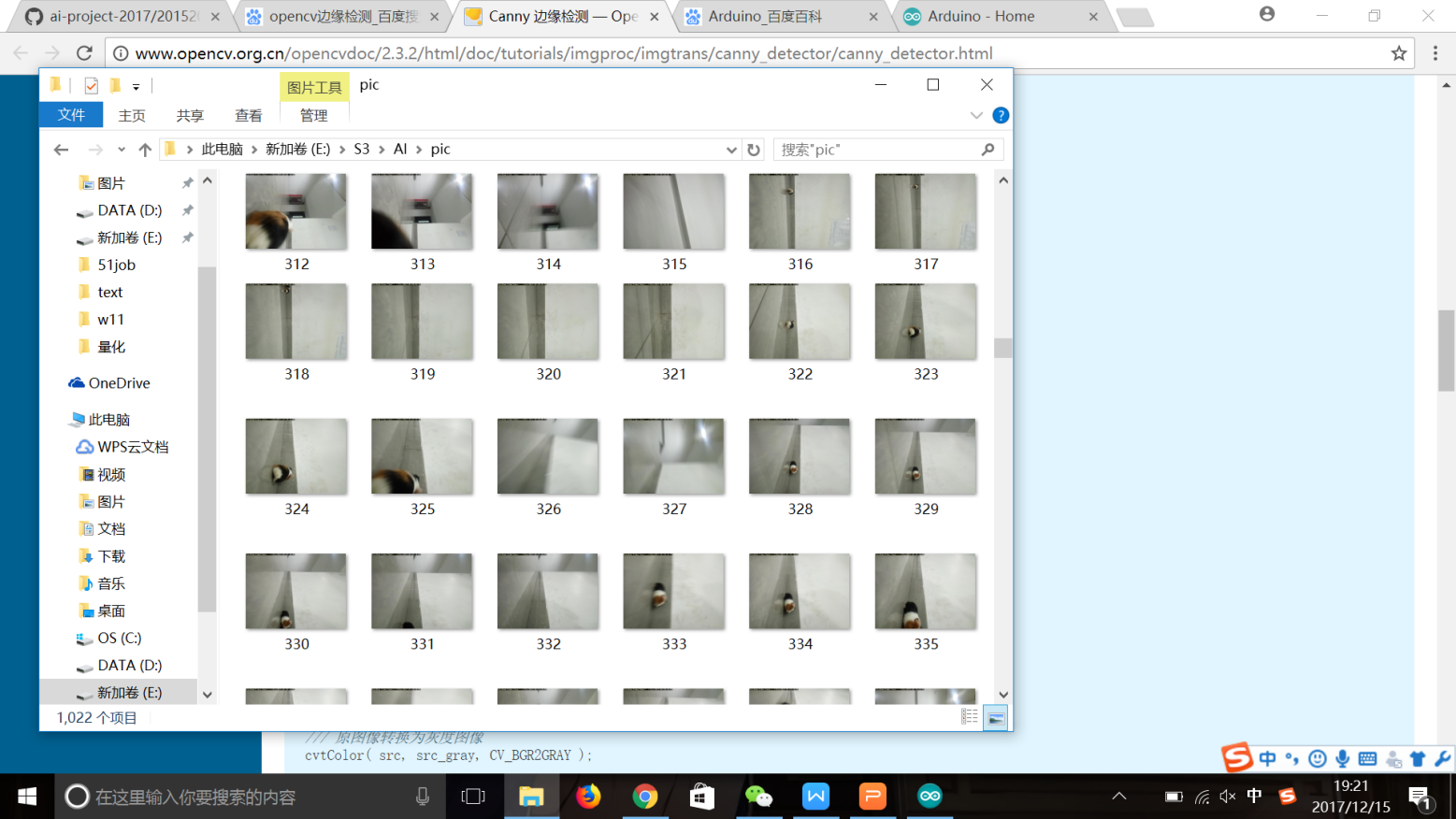
### 功能介绍



### 实验心得

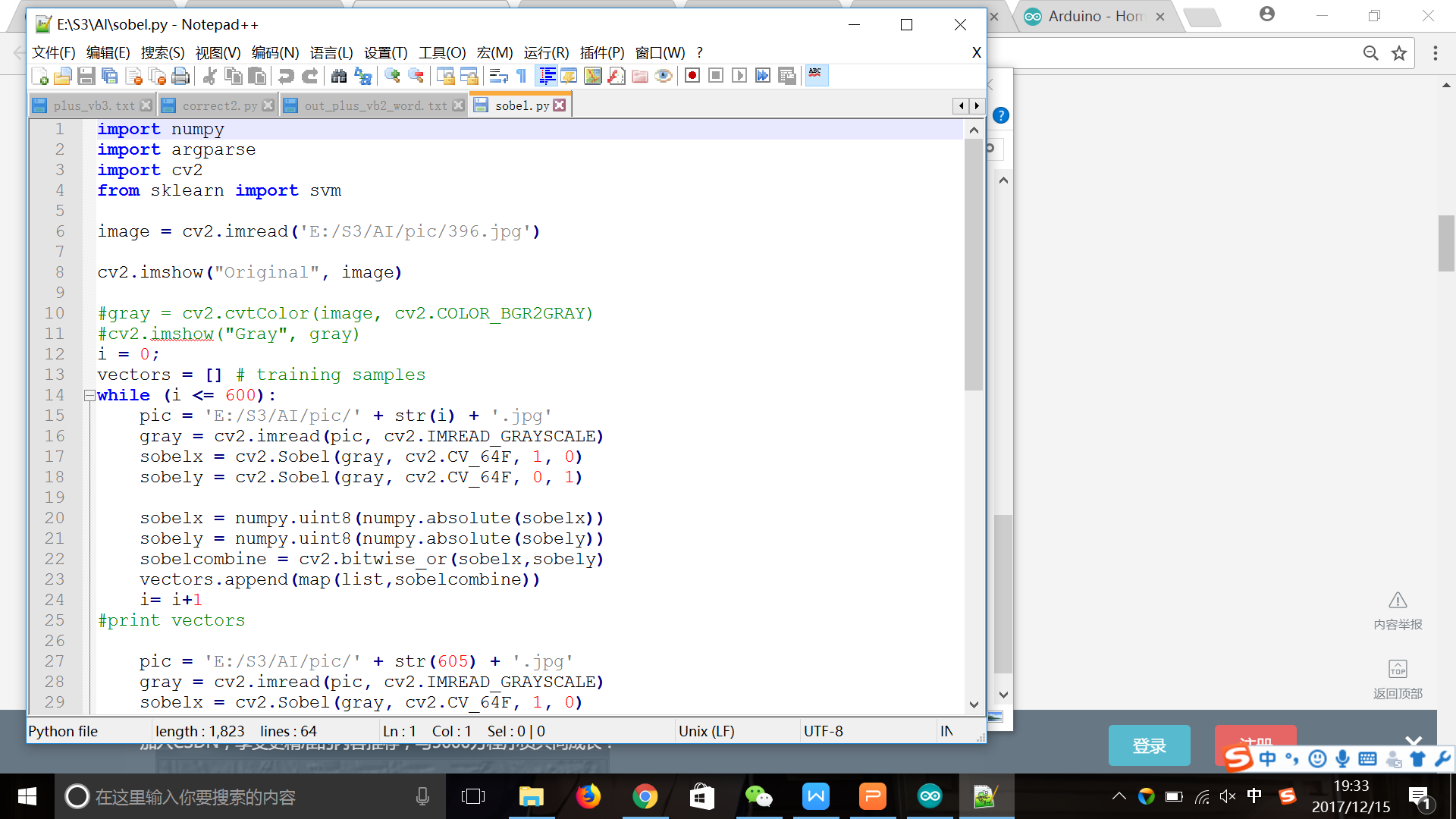
这次实验中我主要负责的部分是图像的识别。实验需要识别Droidcam实时传输过来的图片中的小鼠，确定小鼠相对小车的位置，根据小鼠相对小车的位置利用蓝牙来发送命令给小车，

从而实现小车追踪小鼠的功能。

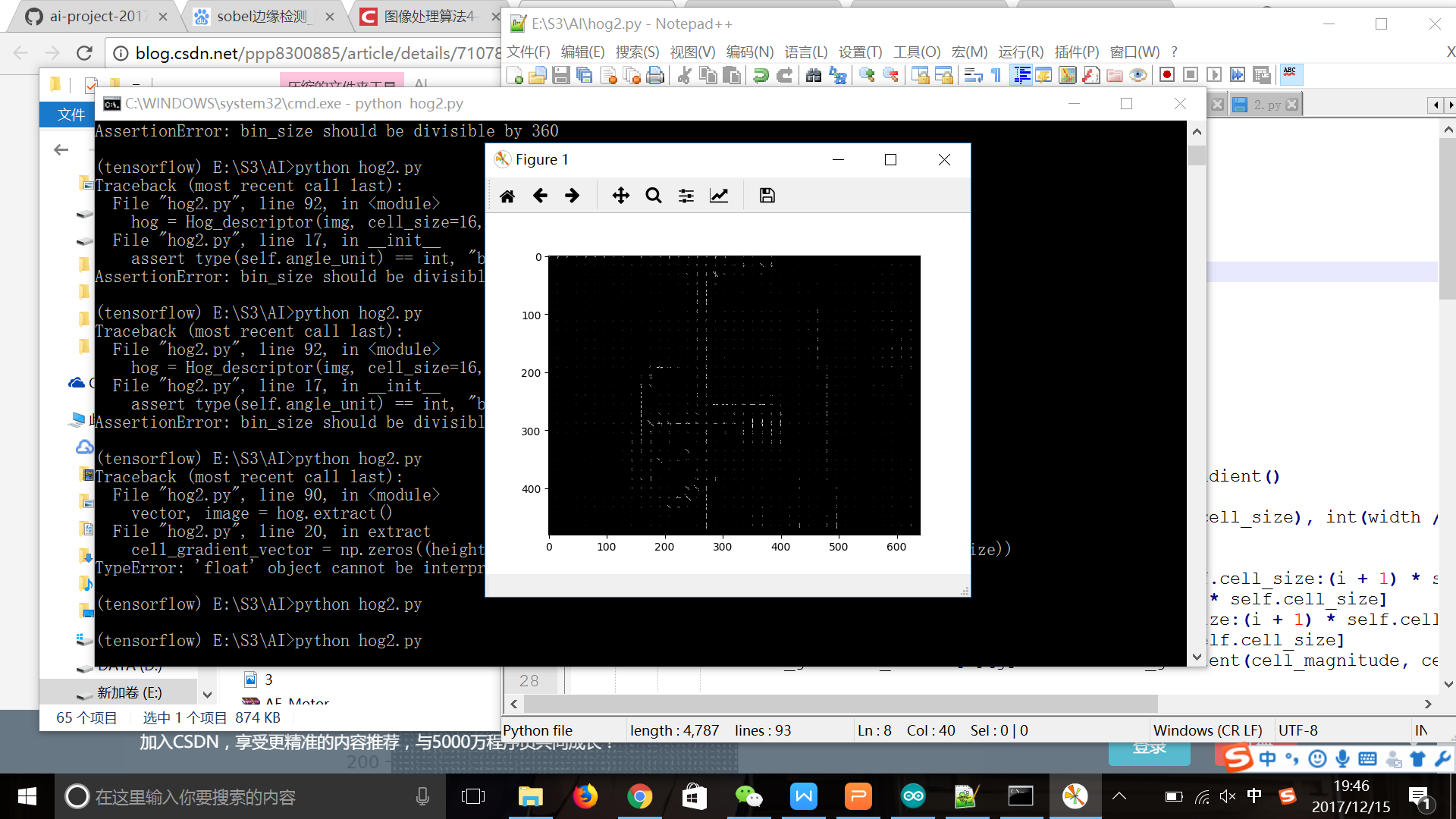


首先我们手动操纵小车前进左右转后退等来拍摄了许多小鼠的图片作为训练集，根据图片中小鼠的位置判断小车此时应该直走还是左转或者右转（小车不需要后退，当画面中没有小鼠时小车原地转圈直到画面中出现小鼠），手动给所有图片打上了标签，前进标为1，左转为3，右转为4。

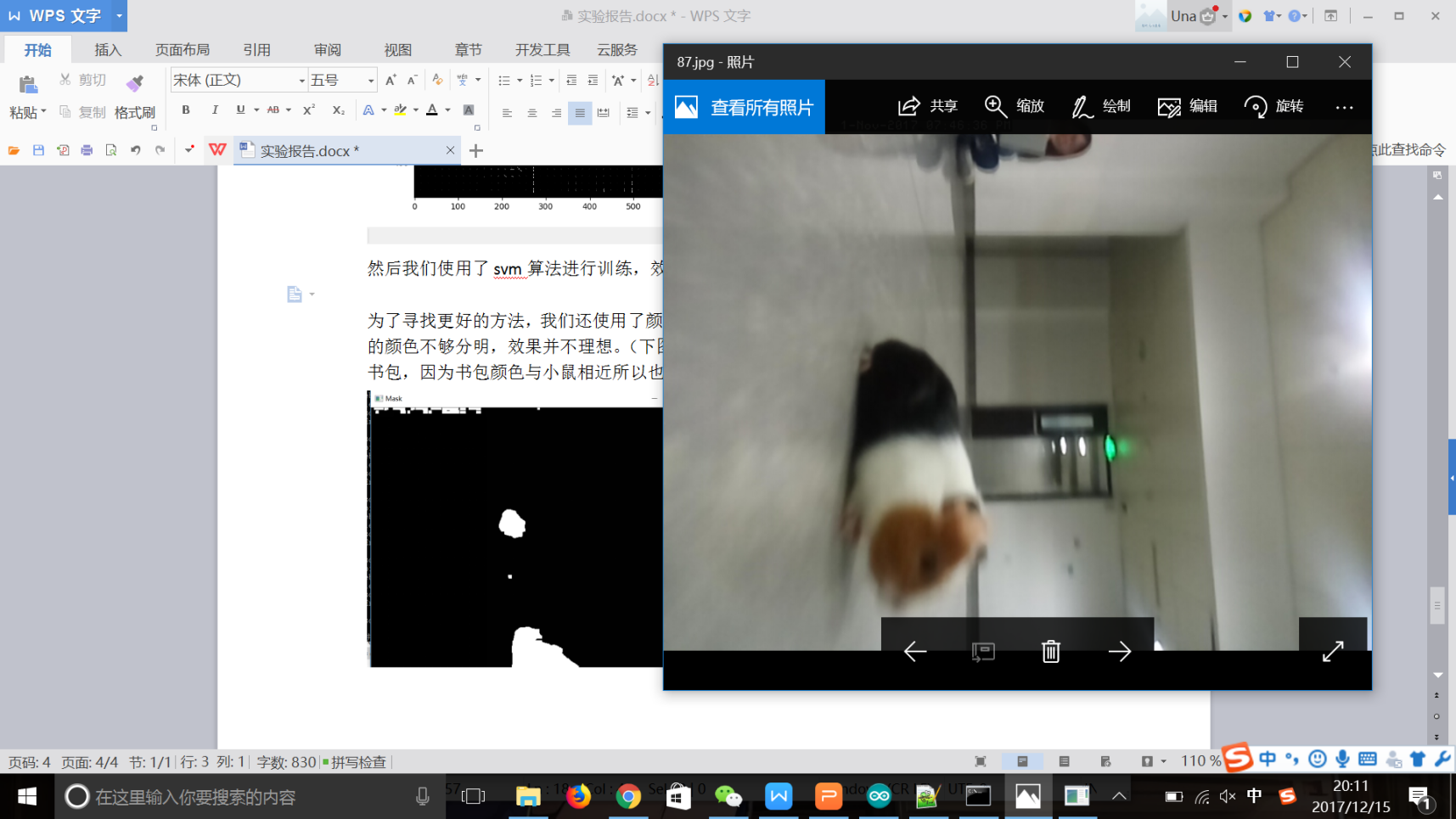
接下来我们使用了边缘检测算法来对图片进行处理。常见的边缘检测算法有Canny边缘检测算法、Sobel算子边缘检测算法、Laplace算子边缘检测算法，我们选择了Sobel算子算法进行试验，使用opencv并参考网上资料写出了代码，并使用了svm算法来进行训练，但是效果并不理想。



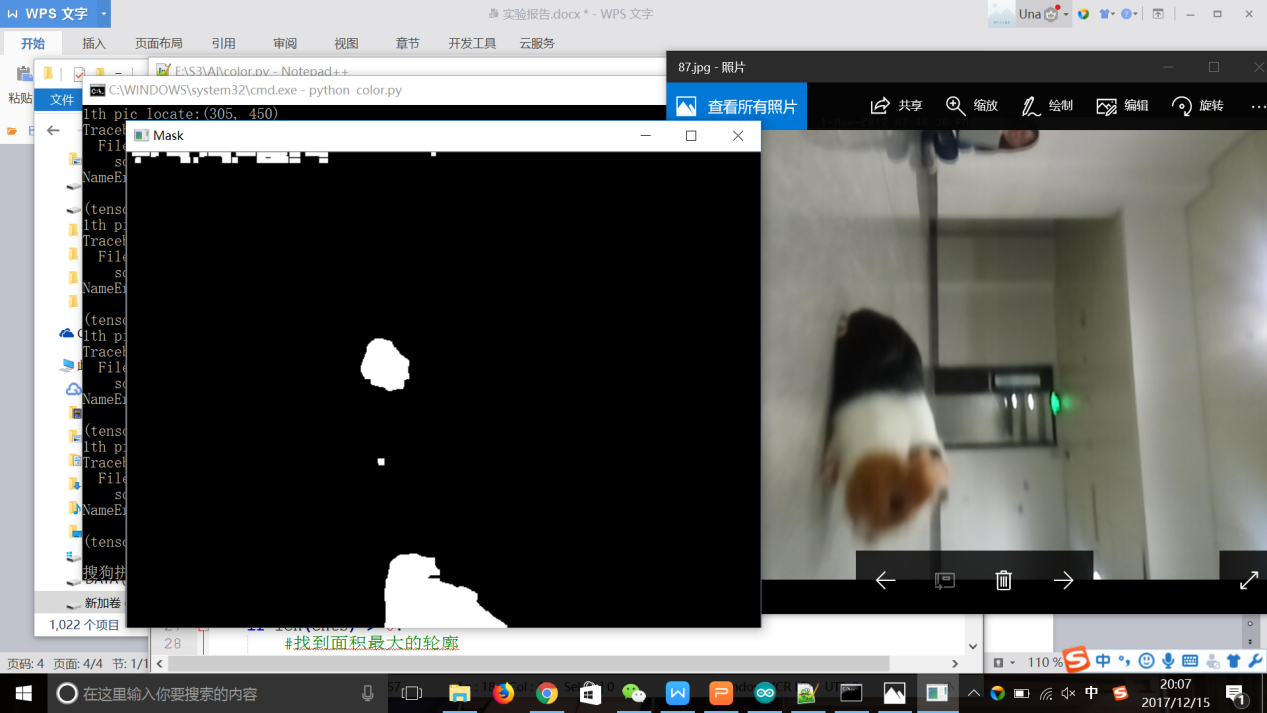
然后我们使用了hog特征检测的方法来处理图片，我们查阅网上的资料后写出了代码。



然后我们使用了svm算法进行训练，效果比使用Sobel算子的效果好。

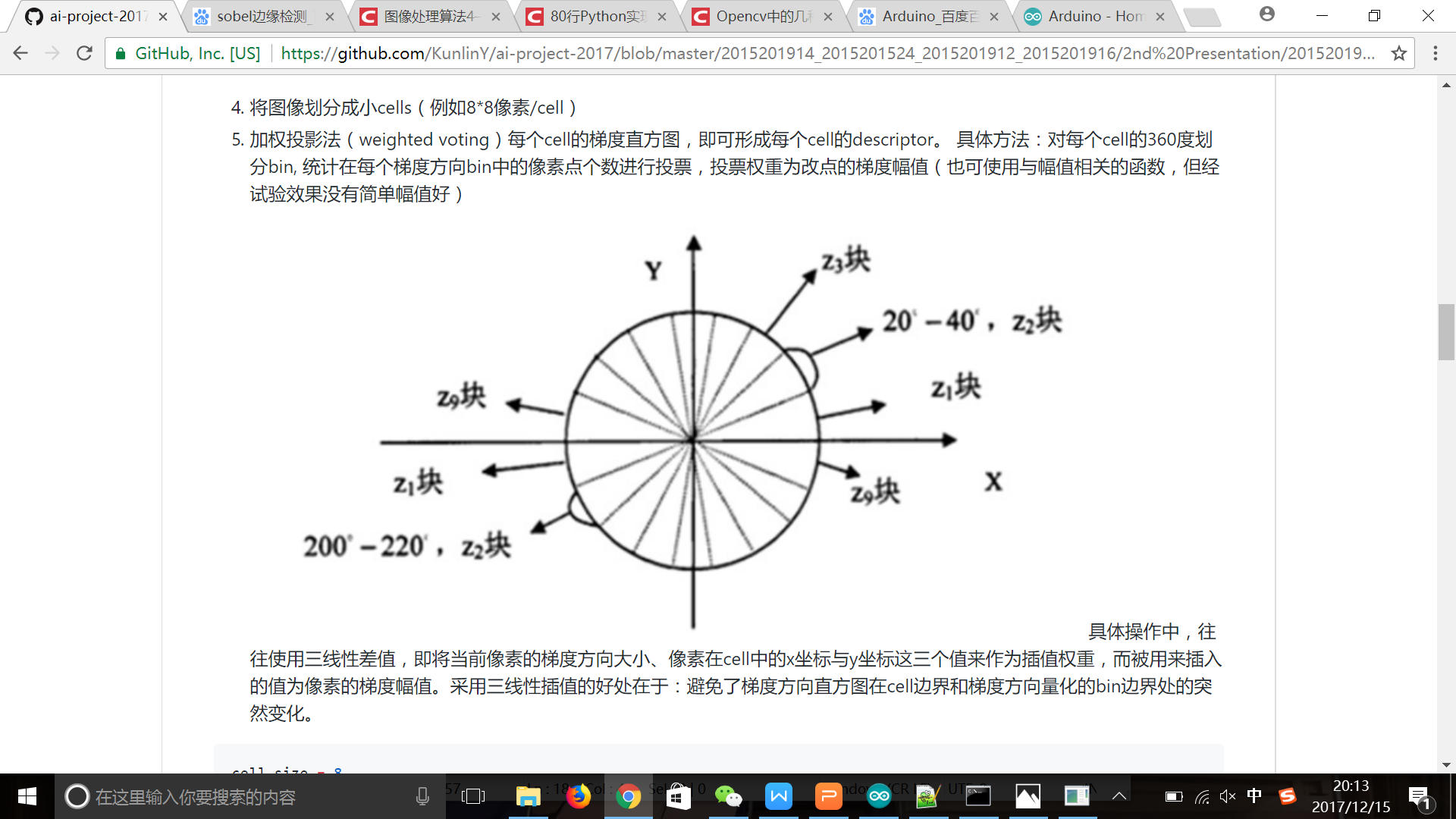


为了寻找更好的方法，我们还使用了颜色识别，但由于小鼠身上颜色较多，且背景色与小鼠的颜色不够分明，效果并不理想。（下图中间白色部分为小鼠，下方白色部分为我们黑色的书包，因为书包颜色与小鼠相近所以也被标识出来了）

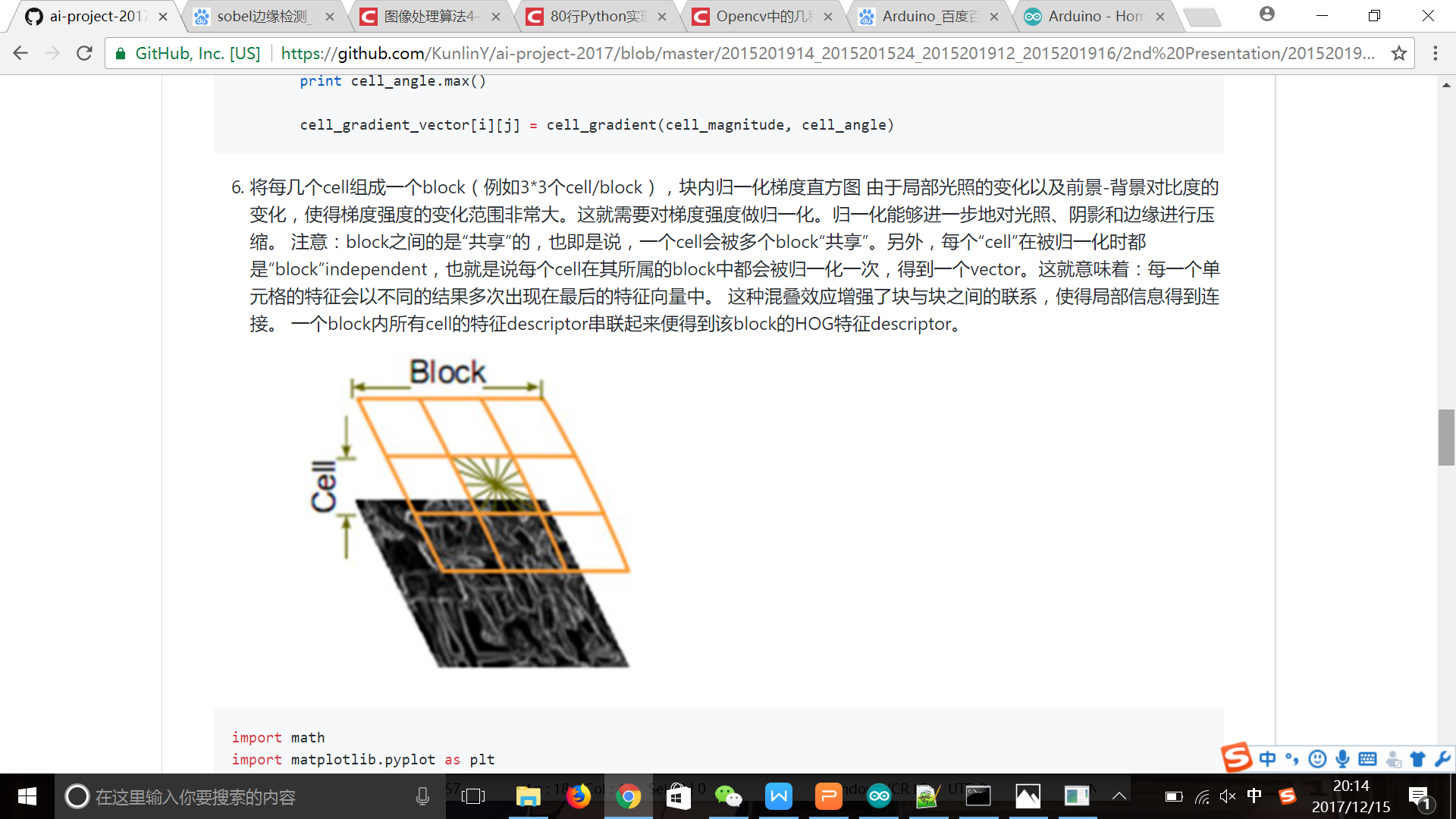


以下是hog算法的简介：











# Mnist数据集部分

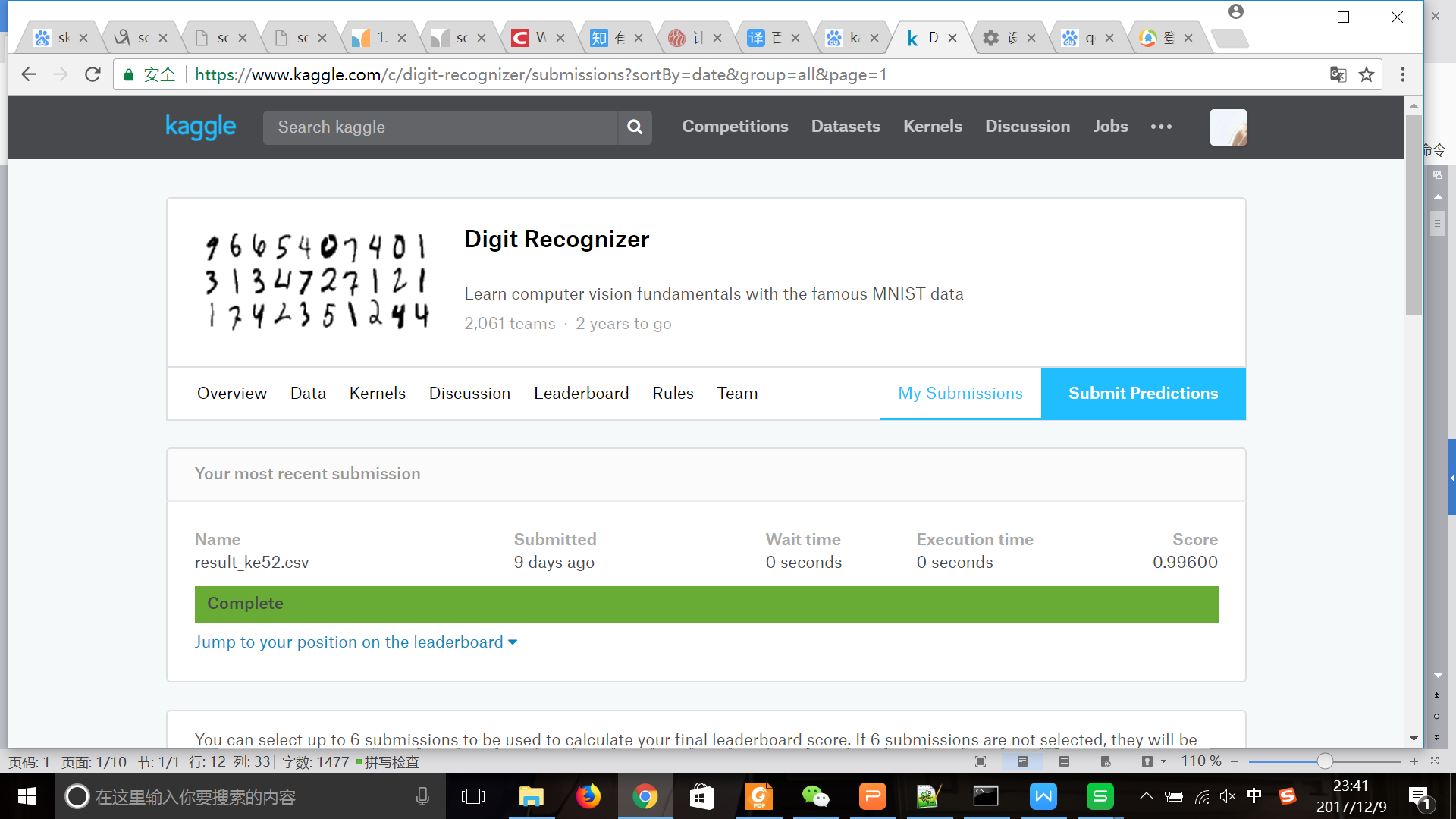
## 简述

本次实验的数据集是keras包中下载的mnist数据集，加上从kaggle平台上下载的42000条训练数据和28000条测试数据，使用的机器学习算法有决策树、随机森林、SVM、神经网络。

## 算法与准确率

### 决策树

在本次实验中，我首先使用了较为简单的决策树算法，在程序实践的过程中，我使用了sklearn，程序较为简单，且效果较好。使用了sklearn包中决策树的默认参数，使用了全部42000个训练数据作为训练集，28000个测试数据作为测试集，将结果在kaggle平台上提交后得到的准确率为85%。



### 随机森林

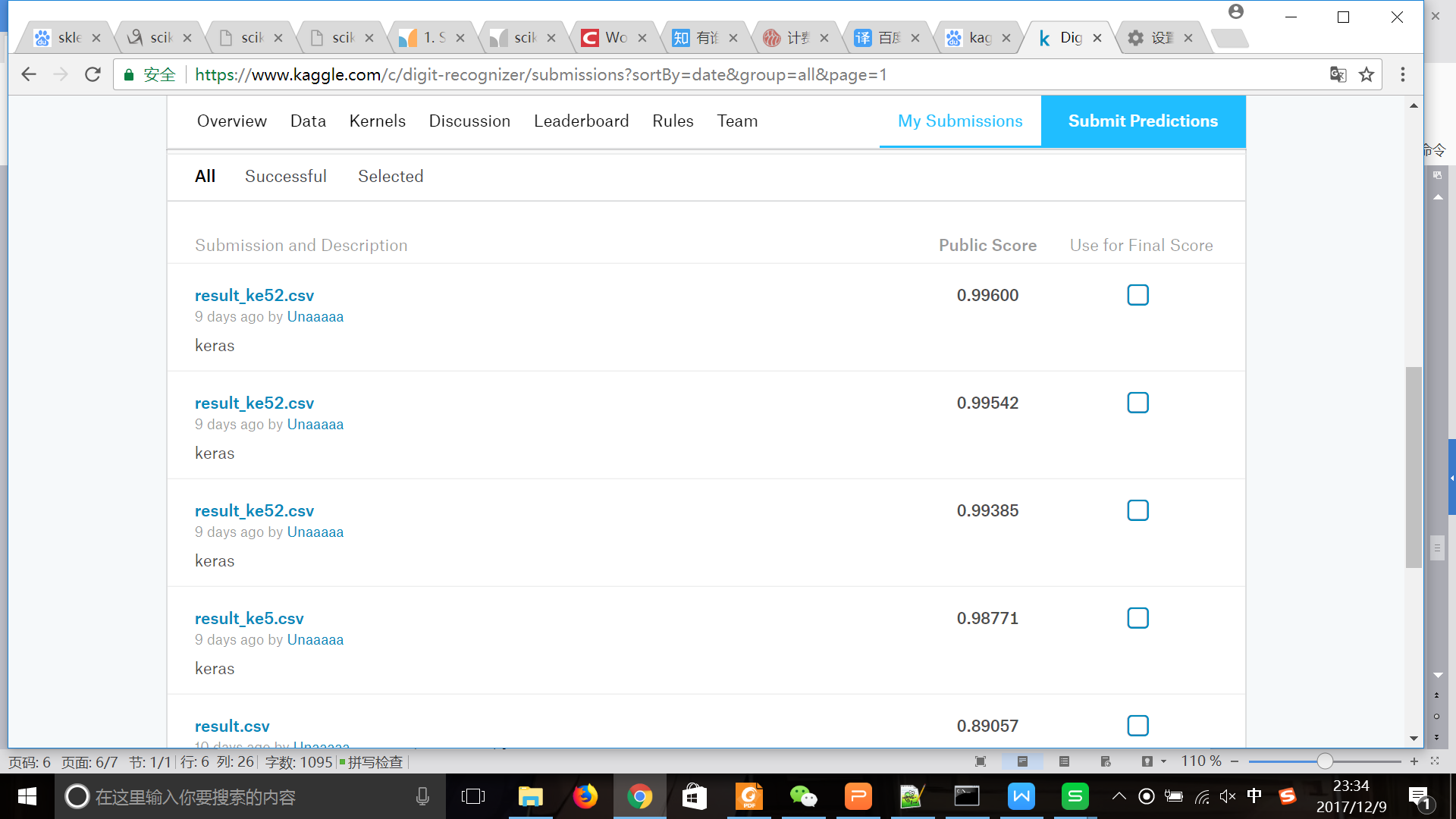
使用了sklearn包中的随机森林算法。由于需要调参，使用了42000条训练数据中的4000条作为验证集，剩余作为训练集，得到参数max\_depth=2, random\_state=1，然后使用测试集中28000条数据作为测试集，将结果提交到kaggle平台上得到准确率为54%，效果不佳。

### SVM+PCA

接下来使用了sklearn包中的SVM算法，由于SVM算法计算的速度较慢，于是使用了PCA方法给数据降维，在保留了数据的信息量的前提下加快计算的速度。同时也是先使用了训练集中的4000条数据作为验证集来调整SVM的参数，最后发现使用SVM的默认参数准确度最高，然后使用全体42000条训练数据来训练模型，全体28000条测试数据作为测试集，将结果提交到kaggle平台上得到的准确率为98%，效果很不错。

### 神经网络

使用了keras包中的神经网络框架来搭建神经网络，以下尝试了两种不同的搭建方法，由于神经网络本身效果比svm好，加之使用的训练数据是直接从keras.datasets中下载的，训练数据有60000条（使用原来的42000条训练数据的准确率也达到了99%以上），因此在kaggle平台上提交后准确率都达到了99%以上。代码一是算法一神经网络的搭建部分，代码二是算法二完整的代码，其中代码二的准确率是本实验中我使用的所有算法里最高的，达到了99.6%。



**代码一：**

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(num\_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy',

optimizer=RMSprop(),

metrics=['accuracy'])

**代码二：**

import datetime

import keras

import csv

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

now = datetime.datetime.now

batch\_size = 128

num\_classes = 10

epochs = 27

# input image dimensions

img\_rows, img\_cols = 28, 28

# number of convolutional filters to use

filters = 32

# size of pooling area for max pooling

pool\_size = 2

# convolution kernel size

kernel\_size = 3

if K.image\_data\_format() == 'channels\_first':

input\_shape = (1, img\_rows, img\_cols)

else:

input\_shape = (img\_rows, img\_cols, 1)

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# define two groups of layers: feature (convolutions) and classification (dense)

feature\_layers = [

Conv2D(filters, kernel\_size,

padding='valid',

input\_shape=input\_shape),

Activation('relu'),

Conv2D(filters, kernel\_size),

Activation('relu'),

MaxPooling2D(pool\_size=pool\_size),

Dropout(0.25),

Flatten(),

]

classification\_layers = [

Dense(128),

Activation('relu'),

Dropout(0.5),

Dense(num\_classes),

Activation('softmax')

]

# create complete model

model = Sequential(feature\_layers + classification\_layers)

x\_train = x\_train.reshape((x\_train.shape[0],) + input\_shape)

x\_test = x\_test.reshape((x\_test.shape[0],) + input\_shape)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

model.compile(loss='categorical\_crossentropy',

optimizer='adadelta',

metrics=['accuracy'])

t = now()

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test, y\_test))

print('Training time: %s' % (now() - t))

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test score:', score[0])

print('Test accuracy:', score[1])

csvFile2 = open("E:/S3/kaggle/test.csv", "r")

reader2 = csv.reader(csvFile2)

out = open('E:/S3/kaggle/result\_ke52.csv','w',newline='')

csv\_write = csv.writer(out,dialect='excel')

csv\_write.writerow(['ImageId','Label'])

data = []

label = []

test = []

count = 1

for i in reader2:

if reader2.line\_num == 1:

continue

i = np.array(i)

i = i.reshape(1,28,28,1)

i = i.astype('float32')

i /= 255

temp = str(int(model.predict\_classes( i, batch\_size=128)))

#out2.write(str(temp)+'\n')

csv\_write.writerow([count,temp])

count = count+1

print(temp)