1. **实验目的**

了解鲁棒性内容感知哈希技术的基本特点，设计并实现基于DCT的鲁棒性图像感知哈希。了解鲁棒性内容感知哈希技术在数字内容保护中的作用，掌握基于鲁棒性感知哈希的内容保护方法。

1. **实验内容**

（1）关于tau的选取，tau是一个经验值，其选取依据是根据测试集的准确率决定，给出选取过程。

（2）关于N的选取，N值越大Hash精度越高,与不同图像Hash碰撞的概率就越小,但鲁棒性会降低,因而需设定合适的N值，以满足Hash在精度和鲁棒性之间的折衷，给出选取过程。

1. **系统整体描述和分功能描述**

**系统整体描述**

通过使用感知hash算法，计算出输入的图片的hash值，并根据hash值的不可篡改性进行比对，计算出两张图片的汉明距离，从而判断相似度

**分功能描述**

1）分功能1

对输入图片进行灰度转化，尺寸修改，DCT变换等操作

用到的函数

MATLAB代码

% 对图片进行灰度转化  
 I1 = im2gray(I1);  
 I2 = im2gray(I2);  
% 修改图片尺寸为64\*64的标准化图  
 I1 = imresize(I1,[64,64]);  
 I2 = imresize(I2,[64,64]);  
% 划分成8\*8子块，进行二维离散余弦变换  
 fun = @dct2;  
 I1 = blkproc(I1,[8 8],fun);  
 I2 = blkproc(I2,[8 8],fun);  
% DC系数，（1,1）置为0  
 I1(1,1) = 0;  
 I2(1,1) = 0;  
% 通过密钥伪随机生成Hashlen个服从标准正态64\*64矩阵  
 randn('state',key);  
 N = cell(1,Hashlen);  
% 高斯低通滤波器进行迭代滤波  
 K = fspecial('gaussian');  
 Y = cell(1,Hashlen);  
  
for i = 1:Hashlen  
 N{i} = randn(64);  
 Y{i} = filter2(K,N{i});  
 end  
% DCT敏感度矩阵m,周期延拓至64\*64  
 m = [  
 71.43 99.01 86.21 60.24 41.67 29.16 20.88 15.24;  
99.01 68.97 75.76 65.79 50.00 36.90 27.25 20.28;  
86.21 75.76 44.64 38.61 33.56 27.47 21.74 17.01;  
60.24 65.79 38.61 26.53 21.98 18.87 15.92 13.16;  
41.67 50.00 33.56 21.98 16.26 13.14 11.48 9.83;  
29.16 36.90 27.47 18.87 13.14 10.40 8.64 7.40;  
20.88 27.25 21.74 15.92 11.48 8.64 6.90 5.78;  
15.24 20.28 17.01 13.16 9.83 7.40 5.78 4.73];

JAVA代码示意如下：

*\*\*  
 \* 离散余弦变换  
 \** ***@param*** *pix 原图像的数据矩阵  
 \** ***@param*** *n 原图像(n\*n)的高或宽  
 \** ***@return*** *变换后的矩阵数组  
 \*/*public static int[] DCT(int[] pix, int n) {  
 double[][] iMatrix = new double[n][n];  
 for(int i=0; i<n; i++) {  
 for(int j=0; j<n; j++) {  
 iMatrix[i][j] = (double)(pix[i\*n + j]);  
 }  
 }  
 double[][] quotient = *coefficient*(n); //求系数矩阵  
 double[][] quotientT = *transposingMatrix*(quotient, n); //转置系数矩阵  
  
 double[][] temp = *matrixMultiply*(quotient, iMatrix, n);  
 iMatrix = *matrixMultiply*(temp, quotientT, n);  
  
 int newpix[] = new int[n\*n];  
 for(int i=0; i<n; i++) {  
 for(int j=0; j<n; j++) {  
 newpix[i\*n + j] = (int)iMatrix[i][j];  
 }  
 }  
 return newpix;  
}  
  
*/\*\*  
 \* 矩阵转置  
 \** ***@param*** *matrix 原矩阵  
 \** ***@param*** *n 矩阵(n\*n)的高或宽  
 \** ***@return*** *转置后的矩阵  
 \*/*private static double[][] transposingMatrix(double[][] matrix, int n) {  
 double nMatrix[][] = new double[n][n];  
 for(int i=0; i<n; i++) {  
 for(int j=0; j<n; j++) {  
 nMatrix[i][j] = matrix[j][i];  
 }  
 }  
 return nMatrix;  
}  
  
*/\*\*  
 \* 求离散余弦变换的系数矩阵  
 \** ***@param*** *n n\*n矩阵的大小  
 \** ***@return*** *系数矩阵  
 \*/*private static double[][] coefficient(int n) {  
 double[][] coeff = new double[n][n];  
 double sqrt = 1.0/Math.*sqrt*(n);  
 for(int i=0; i<n; i++) {  
 coeff[0][i] = sqrt;  
 }  
 for(int i=1; i<n; i++) {  
 for(int j=0; j<n; j++) {  
 coeff[i][j] = Math.*sqrt*(2.0/n) \* Math.*cos*(i\*Math.*PI*\*(j+0.5)/(double)n);  
 }  
 }  
 return coeff;  
}  
  
*/\*\*  
 \* 矩阵相乘  
 \** ***@param*** *A 矩阵A  
 \** ***@param*** *B 矩阵B  
 \** ***@param*** *n 矩阵的大小n\*n  
 \** ***@return*** *结果矩阵  
 \*/*private static double[][] matrixMultiply(double[][] A, double[][] B, int n) {  
 double nMatrix[][] = new double[n][n];  
 double t;  
 for(int i=0; i<n; i++) {  
 for(int j=0; j<n; j++) {  
 t = 0;  
 for(int k=0; k<n; k++) {  
 t += A[i][k]\*B[k][j];  
 }  
 nMatrix[i][j] = t;  
 }  
 }  
 return nMatrix;  
}  
  
*/\*\*  
 \* 求灰度图像的均值  
 \** ***@param*** *pix 图像的像素矩阵  
 \** ***@param*** *w 图像的宽  
 \** ***@param*** *h 图像的高  
 \** ***@return*** *灰度均值  
 \*/*public static int averageGray(int[] pix, int w, int h) {  
 int sum = 0;  
 for(int i=0; i<h; i++) {  
 for(int j=0; j<w; j++) {  
 sum = sum+pix[i\*w + j];  
 }  
 }  
 return sum/(w\*h);  
}  
  
*/\*\*  
 \* 简化色彩  
 \** ***@param*** *rgb  
 \** ***@return*** *\*/*private int gray(int rgb) {  
 int a = rgb & 0xff000000;//将最高位（24-31）的信息（alpha通道）存储到a变量  
 int r = (rgb >> 16) & 0xff;//取出次高位（16-23）红色分量的信息  
 int g = (rgb >> 8) & 0xff;//取出中位（8-15）绿色分量的信息  
 int b = rgb & 0xff;//取出低位（0-7）蓝色分量的信息  
 rgb = (r \* 77 + g \* 151 + b \* 28) >> 8; // NTSC luma，算出灰度值  
 //(int)(r \* 0.3 + g \* 0.59 + b \* 0.11)  
 return a | (rgb << 16) | (rgb << 8) | rgb;//将灰度值送入各个颜色分量  
}  
  
*/\*\*  
 \* 改变图片尺寸  
 \** ***@param*** *src 原图片  
 \** ***@param*** *height 目标高度  
 \** ***@param*** *width 目标宽度  
 \** ***@return*** *\*/*private BufferedImage resize(BufferedImage src, int height, int width) {  
 BufferedImage image = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_BGR*);  
 Graphics graphics = image.createGraphics();  
 graphics.drawImage(src, 0, 0, width, height, null);  
 return image;  
}

2）分功能2

对Hash值的计算，以及已知Hash值，计算二者汉明距离

用到的函数

% 对N个伪随机矩阵遍历计算  
for k = 1:Hashlen  
 I1\_sum = 0;  
 I2\_sum = 0;  
 for i = 1:64  
 for j = 1:64  
 I1\_sum = I1\_sum + I1(i,j) \* Y{k}(i,j) \* M(i,j);  
 I2\_sum = I2\_sum + I2(i,j) \* Y{k}(i,j) \* M(i,j);  
 end  
 end  
 if I1\_sum < 0  
 I1\_Hash(k) = 0;  
 end  
 if I2\_sum < 0  
 I2\_Hash(k) = 0;  
 end  
 end  
% 汉明距离计算  
 dis = norm((I1\_Hash-I2\_Hash)/(2\*sqrt(norm(I1\_Hash)\*norm(I2\_Hash))));

1. 分功能3

循环判断测试集，设定基准图，循环判断以测试出相对准确的阈值

用到的函数

% 获取测试集中图片路径  
 img\_path = dir('D:/matlab/Test/DogsVsCats\_dogs-vs-cats-redux-kernels-edition/test/small/\*');  
 img\_path = img\_path(~[img\_path.isdir]);

% 获取测试集中所有图片  
 imgList = fullfile({img\_path.folder}.', {img\_path.name}.');  
% 设置基准图  
 I1 = imread('D:/matlab/Test/DogsVsCats\_dogs-vs-cats-redux-kernels-edition/test/small/1.bmp');  
 subplot(4,ceil(length(imgList)/4),1);imshow(I1);title('基准图');  
% 循环比较 并输出结果  
for i = 2:length(imgList)  
 I2 = imread(imgList{i});  
 res = imgHashSimilar(I1,I2,tau,key,Hashlen);  
 disp(imgList{i})  
 subplot(4,ceil(length(imgList)/4),i);  
 imshow(I2);  
 title(res);  
 end

**JAVA 代码实现如下**

for (int i = 0; i <ori ; i+=20) {  
 //绝对路径取出图片文件  
 File oriImage = new File("D:/minIO/Files/images1/frame\_"+i+".jpg");  
 //System.out.println(oriImage.getAbsolutePath());  
 for(int j=0;j<mon;j+=20){  
 File monImage = new File("D:/minIO/Files/images2/frame\_"+j+".jpg");  
 //System.out.println(monImage.getAbsolutePath());  
  
 //获取两个图片文件的dhash值  
 String oriHash = ImageDHash.*getDHash*(oriImage);  
 String monHash = ImageDHash.*getDHash*(monImage);  
  
 //计算二者的汉明距离  
 long hammingDistance = ImageDHash.*getHammingDistance*(oriHash, monHash);

}

1. **实验步骤、结果及分析**

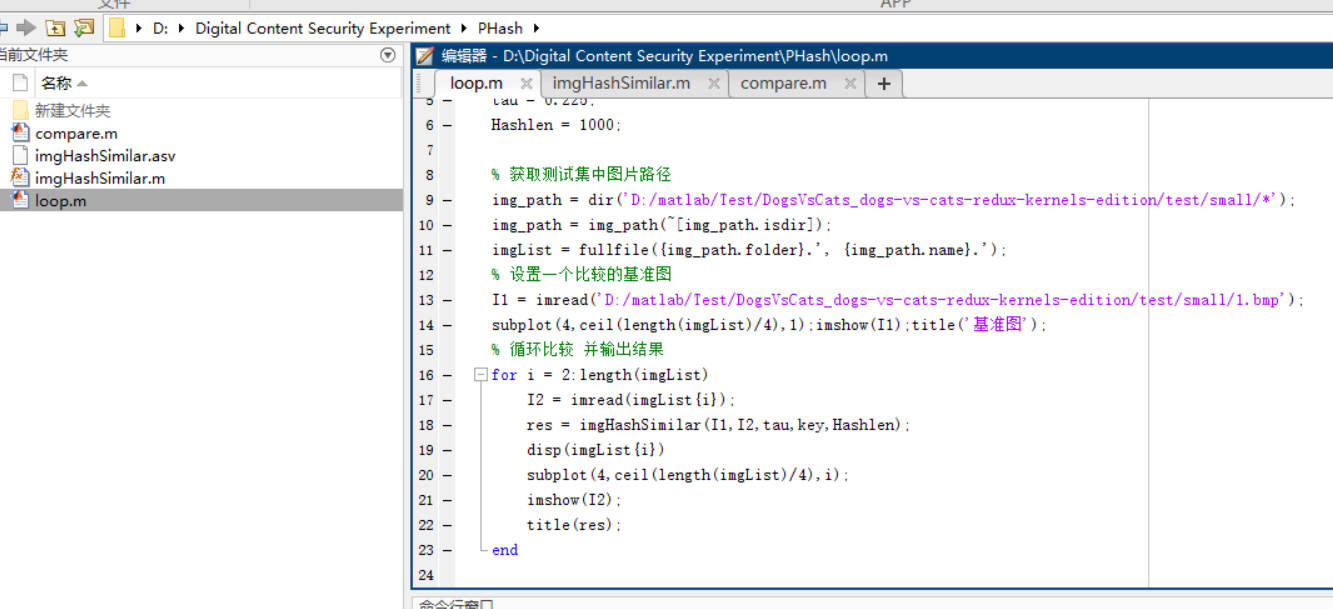
**实验步骤：**

1. **下载配置MATLAB**

**注意事项：对图片处理时，MatLab 需要Image Processing Toolbox 工具库，需勾选配置下载。**

****

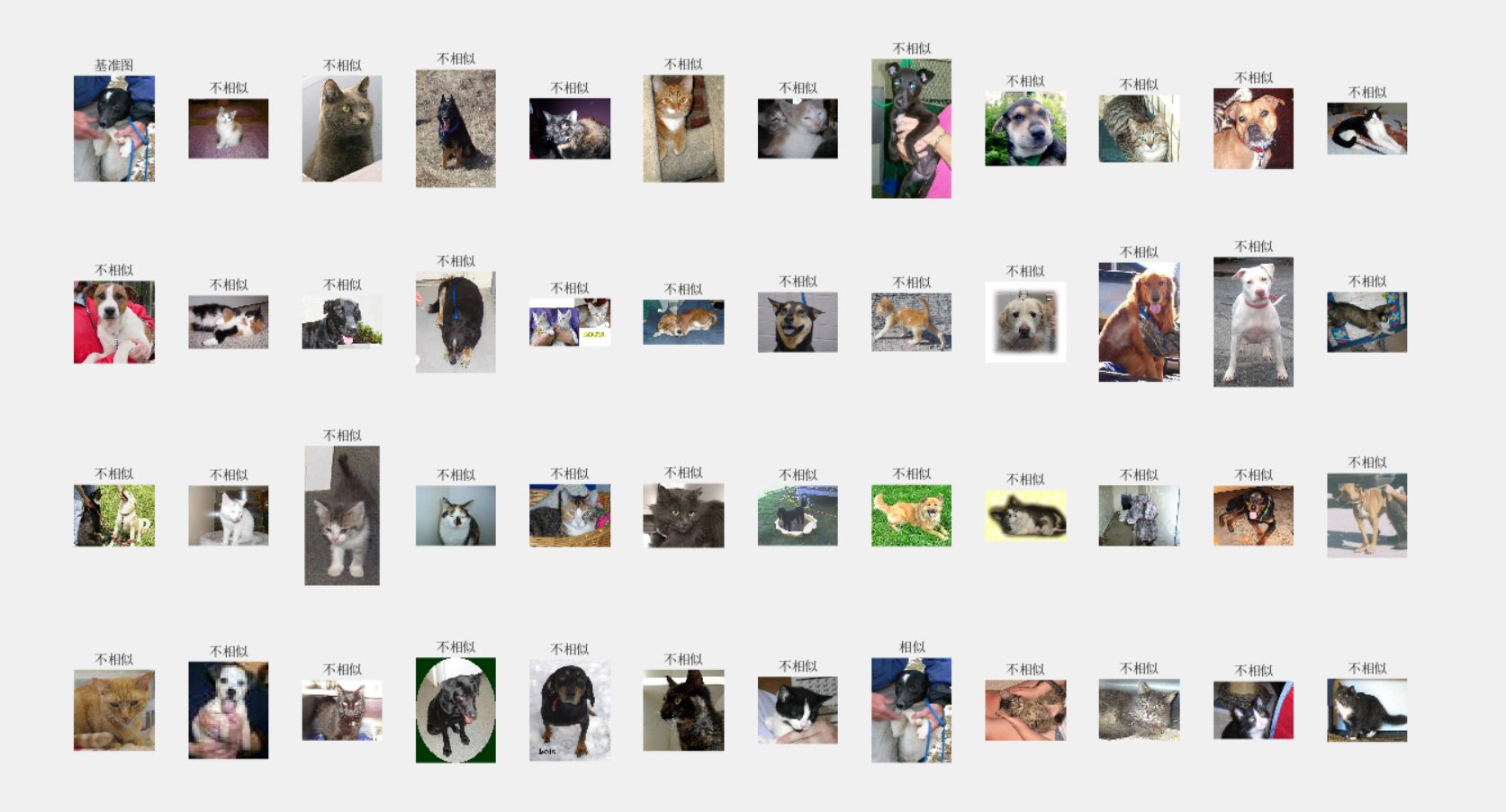
1. **根据实验参考资料，编写MatLab代码以及Java代码**

****

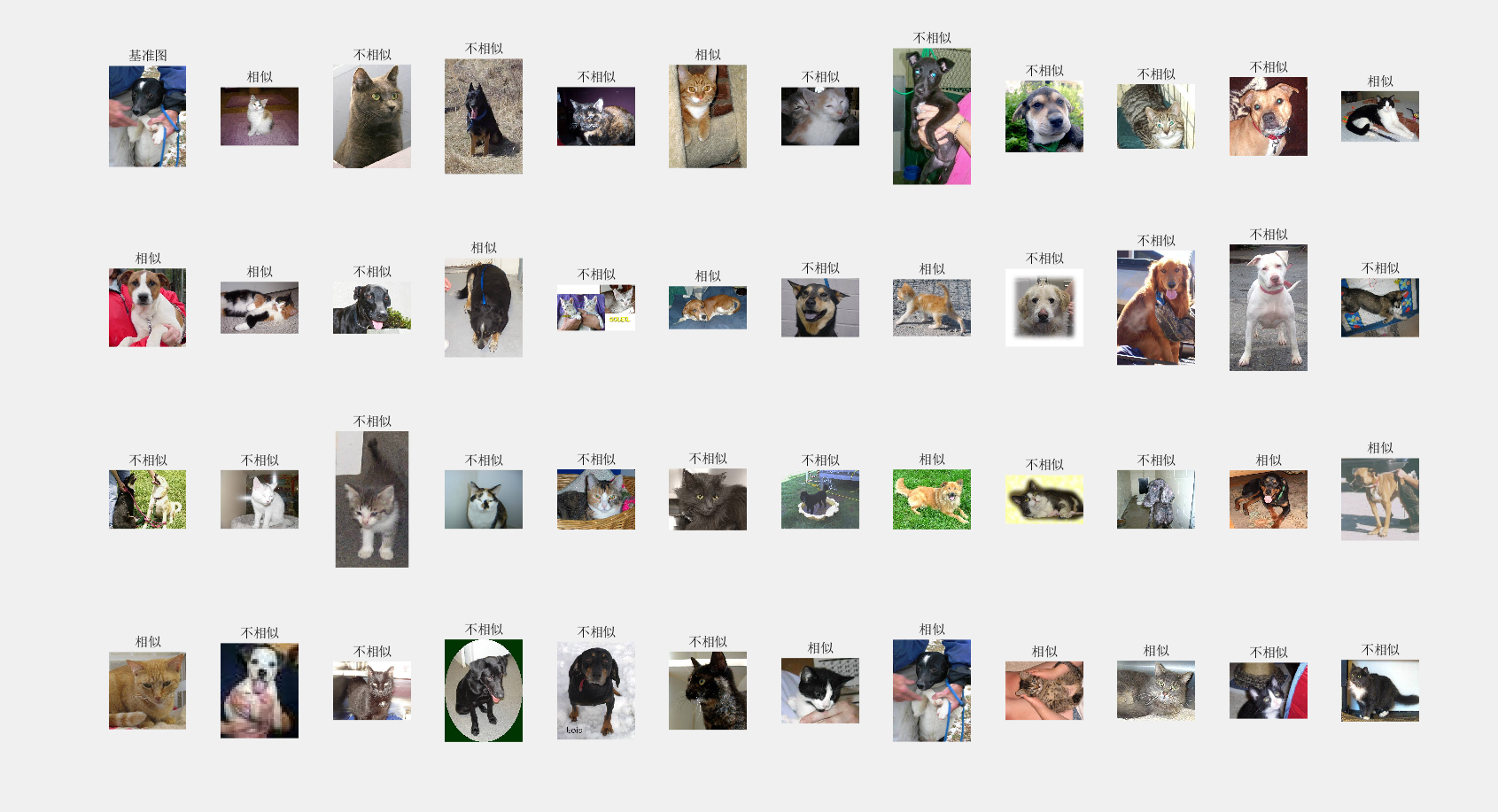
****

1. **设定不同阈值，对设定同一测试集进行判断，并统计正确率**

**Tau=0.2**

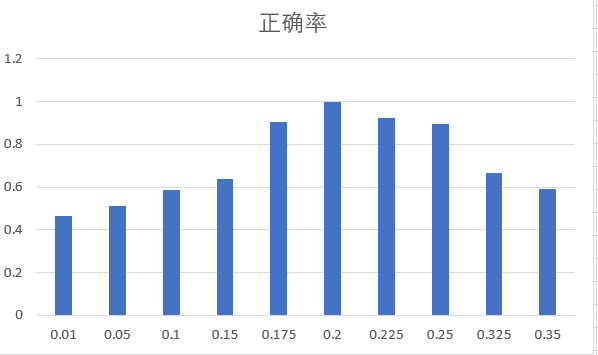
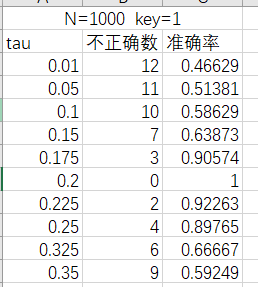
****

**Tau=0.3**

****

**这里采用的是对一个视频的关键帧图片相似度检测（个人项目，GitHub地址：https://github.com/Shawnicsc/Login.git）  
**

1. **统计正确率，选出相对准确阈值**

****

1. **N值的选取**

**查阅资料可知，N 值越大Hash精度越高,与不同图像Hash碰撞的概率就越小,但鲁棒性会降低,因而需设定合适的N值，以满足Hash在精度和鲁棒性之间的平衡。Hash的位数决定了其能够表示不同图形的个数，即最大为2^n个。通过实验可以测得在tau和key不变的情况下，Hash的位数越大图像感知越精密，图片相似度判断越精准。因此，处于均衡考虑，我们可以选择比2^n 稍大的数作为N.**

1. **实验中遇到的问题及改正的方法**
2. **matlab下载后，没选择所需求的工具库，导致部分函数无法调用**

**官网下载工具库后，配置即可（或者重装matlab，选择工具库一并安装）。**