**数字图像处理大作业报告**

**姓名 吴秀全 ­­­­ 学号 19220110 ； 22 级 01 班**

**成 绩\_\_\_\_\_\_\_\_\_\_\_\_**

**课程名称：数字图像处理 教师签字\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| 实验内容 | 基于matlab的鸟类数字图像处理及GUI界面设计 | 实验时间 | 2024.12.20 |
| 实验目的 | 1. 了解和掌握图像处理工具Matlab，熟悉基于Matlab的图像处理函数，并为下一步编程进行图像处理打下基础。 2. 实现针对鸟类图像的各种图像处理方法，并训练模型用于预测鸟类图像类别。 3. 进行GUI图形界面设计，进行交互式操作。 | | |
| 实验工具 | Matlab R2022a | | |

|  |
| --- |
| 目录  [数字图像大作业报告 1](#_Toc29256)  [（一）程序流程 4](#_Toc23987)  [（二）算法原理与实现流程 5](#_Toc18787)  [1灰度直方图、直方图均衡化和直方图匹配(规定化) 5](#_Toc8816)  [1.1 灰度直方图 5](#_Toc4381)  [1.2 直方图均衡化 5](#_Toc11523)  [1.3 直方图匹配 6](#_Toc24893)  [2.灰度化与对比度增强 7](#_Toc11708)  [2.1 灰度化 7](#_Toc18920)  [2.2 线性变换 8](#_Toc30940)  [2.3 对数变换 8](#_Toc2116)  [3.图像缩放与旋转 9](#_Toc20789)  [3.1 图像缩放 9](#_Toc27927)  [3.2 图像旋转 10](#_Toc24771)  [4.图像加噪与滤波 12](#_Toc28020)  [4.1 椒盐噪声 12](#_Toc5939)  [4.2 高斯噪声 13](#_Toc21939)  [4.3 泊松噪声 13](#_Toc9579)  [4.4 空域均值滤波 14](#_Toc6473)  [4.5 空域高斯滤波 15](#_Toc18066)  [4.6 空域中值滤波 15](#_Toc20024)  [4.7 空域双边滤波 16](#_Toc18994)  [4.8 频域低通滤波 17](#_Toc6449)  [4.9 频域高通滤波 18](#_Toc12377)  [5.边缘提取 19](#_Toc14243)  [5.1 Sobel算子 19](#_Toc28655)  [5.2 robert算子 20](#_Toc32577)  [5.3prewitt算子 21](#_Toc155)  [5.4拉普拉斯算子 21](#_Toc9932)  [6.目标提取 22](#_Toc19691)  [6.1目标提取 22](#_Toc2599)  [7.特征提取 23](#_Toc5787)  [7.1 lBP特征提取 23](#_Toc27638)  [7.2 hog特征提取 23](#_Toc26157)  [8.模型预测 24](#_Toc2400)  [8.1模型训练与应用 24](#_Toc10291)  [（三）算法代码及功能注释 25](#_Toc14501)  [1灰度直方图、直方图均衡化和直方图匹配(规定化) 25](#_Toc28155)  [1.1 灰度直方图 25](#_Toc20907)  [1.2 直方图均衡化 25](#_Toc13431)  [1.3 直方图匹配 26](#_Toc22448)  [2.灰度化与对比度增强 27](#_Toc27549)  [2.1 灰度化 28](#_Toc21949)  [2.2 线性变换 28](#_Toc2453)  [2.3 对数变换 29](#_Toc11158)  [2.4 指数变换 29](#_Toc4196)  [3.图像缩放与旋转 30](#_Toc14134)  [3.1 图像缩放 30](#_Toc25088)  [3.2 图像旋转 31](#_Toc26389)  [4.图像加噪与滤波 33](#_Toc24043)  [4.1 椒盐噪声 33](#_Toc25556)  [4.2 高斯噪声 33](#_Toc23757)  [4.3 泊松噪声 34](#_Toc13305)  [4.4 空域均值滤波 34](#_Toc413)  [4.5 空域高斯滤波 35](#_Toc4838)  [4.6 空域中值滤波 35](#_Toc21853)  [4.7 空域双边滤波 36](#_Toc15411)  [4.8 频域低通滤波 37](#_Toc2423)  [4.9 频域高通滤波 38](#_Toc14084)  [5.边缘提取 39](#_Toc24124)  [5.1 Sobel算子 39](#_Toc18730)  [5.2 robert算子 40](#_Toc22031)  [5.3prewitt算子 40](#_Toc14334)  [5.4拉普拉斯算子 41](#_Toc25417)  [6.目标提取 42](#_Toc27990)  [6.1目标提取 42](#_Toc28118)  [7.特征提取 43](#_Toc2322)  [7.1 lbp特征提取 43](#_Toc14103)  [7.2 hog特征提取 44](#_Toc27658)  [8.模型预测 46](#_Toc5530)  [8.1 模型训练与应用 46](#_Toc28514)  [9.GUI界面设计 50](#_Toc3988)  [9.1 GUI界面设计代码 50](#_Toc21393)  [（四）运行结果 71](#_Toc21071)  [（五）实验结论及收获建议 85](#_Toc3237)  **（一）程序流程**  1、用 Matlab 设计实现图形化界面，调用后台函数完成设计，函数尽量使用自己编写的函数。设计完  成后，点击 GUI图形界面上的菜单或者按钮，进行必要的交互式操作后，最终能显示运行结果。  2、要求至少实现以下功能:   1. 对于打开的图像可以显示其灰度直方图，并实现直方图均衡化和直方图匹配(规定化)。 2. 读取图像并灰度化，然后实现该灰度图像的对比度增强，要求实现线性变换和非线性变换(包括对数变换和指数变换)。 3. 实现图像的缩放变换、旋转变换等。 4. 图像加噪(用输入参数控制不同噪声)，然后分别使用空域和频域进行滤波处理。 5. 采用 robert 算子，prewitt 算子，sobel 算子，拉普拉斯算子对图像进行边缘提取。 6. 读入图像，应用所学的知识从图像中提取出目标。 7. 对原始图像和所提取的目标分别进行特征提取(LBP和HOG)   (8) 加分项：应用传统机器学习或者深度学习方法，训练智能模型实现图像的分类。 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **（二）算法原理与实现流程** 1灰度直方图、直方图均衡化和直方图匹配(规定化)1.1 灰度直方图 1.1.1 灰度直方图实现原理  灰度直方图是灰度级的函数，是图像处理和计算机视觉中的一种工具，它用于统计图像中每个灰度级出现的频率。  1.1.2灰度直方图实现流程  图像预处理：如果输入图像是彩色的，则需要将其转换为灰度图像。这通常是通过将每个像素的RGB值转换为一个代表亮度的单一灰度值来完成的。转换公式可以是简单的平均值计算，或者是加权求和，如Y = 0.299R + 0.587G + 0.114B（这是考虑到人眼对不同颜色敏感度不同的标准转换公式）。  定义灰度范围：确定图像中可能存在的灰度来存储每个灰度级别的像素数量。  遍历图像像素：遍历图像中的每一个像素，读取其灰度值，并根据该灰度值递增相应的直方图数组元素。例如，如果一个像素的灰度值为128，那么就增加直方图数组中索引为128的位置的计数值。  归一化：有时为了比较不同大小的图像或者突出分布特性，会对直方图进行归一化。这意味着将每个直方图条目的值除以像素总数，得到的是每个灰度级的概率密度而不是绝对计数。 1.2 直方图均衡化 1.2.1 直方图均衡化实现原理  直方图均衡化（Histogram Equalization, HE）是一种用于图像增强的点处理技术，它通过重新分配图像的灰度值来扩展对比度。其目的是将原始图像的灰度分布转换为均匀分布，从而使得图像中的细节更加清晰。  1.2.2直方图均衡化实现流程  累积分布函数（CDF）：首先计算图像中每个灰度级别的累积分布函数（Cumulative Distribution Function, CDF）。对于一个具有L个可能灰度级别的图像，灰度级别从0到L-1，第i个灰度级别的累积分布函数定义为所有小于或等于i的灰度级别的概率之和。其中 p(j) 是灰度级别j的概率密度，即该灰度级别出现的次数除以总像素数:  灰度映射：根据CDF计算出新的灰度映射规则。新灰度值v可以通过以下公式计算,这里i是原始灰度值，L是灰度级的数量（例如对于8位图像，L=256）v是应用了直方图均衡化之后的新灰度值。这个公式确保了输出图像的灰度分布尽可能接近均匀分布:  映射应用：将上述得到的灰度映射应用于原图像的每一个像素上，生成一个新的图像，其中每个像素的灰度值都是按照上述规则计算出来的。这样就实现了图像的直方图均衡化。 1.3 直方图匹配 **1.3.1直方图匹配实现原理**  直方图匹配（Histogram Matching），也被称为直方图规定化（Histogram Specification），是一种图像处理技术，它使得一个图像的灰度分布与另一个参考图像的灰度分布相匹配。这种技术可以用于对比度调整、图像融合、纹理合成等应用中。  **1.3.2直方图匹配实现流程**  计算原始图像和参考图像的直方图：分别计算待处理图像（源图像）和参考图像的灰度直方图。  计算累积分布函数（CDF）：对于两个图像，分别计算它们的累积分布函数（Cumulative Distribution Function, CDF）。CDF表示的是小于或等于某一灰度级别的像素数占总像素数的比例，其中psource(j)和ptarget(j)分别是源图像和目标图像在灰度级别j的概率密度：  创建应用灰度映射表：创建一个从源图像灰度级别到目标图像灰度级别的映射表。使用上述创建的映射表对源图像进行变换。对于源图像中的每个像素，根据其灰度值查找映射表，并将该像素的灰度值替换为映射表中对应的值。 2.灰度化与对比度增强2.1 灰度化 **2.1.1 灰度化实现原理**  是指将彩色图像转换为灰度图像的过程，在实现流程中展示三种方式。  **2.2.2 灰度化实现流程**  A.平均值法  平均值法是最简单的方法，取RGB三个通道值的平均值作为灰度值：   1. 加权求和法（Luminosity Method）   加权求和法是我们常用的方法，是根据人眼对不同颜色的感知权重来计算灰度值。由于人类视觉系统对绿色更为敏感，而对蓝色相对不那么敏感，因此可以使用加权求和的方式计算灰度值，我们本项目实现的代码就是基于此方法：   1. 最大值法   还有些情况下，可能会采用最大值法，即选择RGB三个通道中的最大值作为灰度值。这种方法较少用，因为它忽略了颜色信息之间的差异：   1. 最小值法   最小值法则相反，它选择RGB三个通道中的最小值作为灰度值。这种方法同样很少使用，因为它倾向于保留阴影部分，而不强调颜色间的区别： 2.2 线性变换 **2.2.1 线性变换实现原理**  线性变换是数字图像处理中的一种基本操作，它指的是对图像的像素值进行线性运算。这类变换可以用来调整图像的对比度、亮度，或者在更复杂的情况下用于图像的几何变换如旋转、缩放和平移等。线性变换的特点是满足叠加原理。  **2.2.2 线性变换实现流程**  在本项目中，我们实现的是点处理（灰度变换），公式如下：  其中 r 是原始图像中的像素强度值，s 是变换后的输出像素强度值，L 是比例因子，参数控制图像对比度；如果大于1，则增加对比度；如果小于1，则降低对比度。而 b 是偏移量，用来调整图像的整体亮度；正值增加亮度，而负值则降低亮度。  裁剪操作：线性变换后，可能会有像素值超出标准的 [0, 255] 范围。为了确保结果图像的有效性，需要将这些值裁剪到合法范围内。所有低于 0 的值被设置为 0，高于 255 的值被设置为 255。 2.3 对数变换 **2.3.1 对数变换实现原理**  对数变换属于数字图像处理领域中的一种非线性点处理技术，其核心在于应用对数函数以调整图像的像素强度值。该变换在扩展图像的暗部细节方面尤为有效，因为它能够有效地压缩高动态范围图像的亮度范围，从而使原本处于低亮度区域的细节变得更加清晰可见。  **2.3.2 对数变换实现流程**  对数变换的公式如下：  其中r是输入图像中的像素强度值，s 是输出图像中对应的像素强度值，c 是一个常数，用于调整变换后的强度范围。  **2.4 指数变换**  **2.3.1 指数变换实现原理**  指数变换的作用是扩展图像的高灰度级、压缩低灰度级。虽然幂次变换也有这个功能，但是图像经过指数变换后对比度更高，高灰度级也被扩展到了更宽的范围。  **2.3.1 指数变换实现流程**  指数变换的公式如下：  s是输出图像的像素值，r是输入图像的像素值，c是常数，γ是指数，它控制着曲线的形状当γ<1时，该变换被称为伽玛校正或幂律变换，它会拉伸图像的暗区域，使得图像看起来更明亮；而当γ>1时，则会压缩暗区域，使图像变暗。 3.图像缩放与旋转3.1 图像缩放 **3.1.1 图像缩放实现原理**  图像缩放（image scaling）是指对数字图像的大小进行调整的过程。图像缩放是一种非平凡的过程，需要在处理效率以及结果的平滑度（smoothness）和清晰度（sharpness）上做一个权衡。当一个图像的大小增加之后，组成图像的像素的可见度将会变得更高，从而使得图像表现得“软”。相反地，缩小一个图像将会增强它的平滑度和清晰度。  图像的放大和缩小的基本原理就是一种映射，即缩小后或者放大后的图像像素位置向原图的一个映射。简单理解就是，将放大或者缩小后的图像将其的坐标（长宽）拉伸或者压缩到和原图一样大时，其像素坐标点对应在原图上的位置就是其映射位置。  **3.1.2 图像缩放实现流程**  确定缩放需求：决定你想要调整图像的新尺寸，通常通过指定宽度和高度的缩放因子来完成。缩放因子可以是大于1的数（放大）或小于1的数（缩小）。  创建新的图像矩阵：根据计算出的新尺寸创建一个空的图像矩阵，准备存放缩放后的图像数据。  插值方法选择：本项目我们选择双线性插值。考虑四个最邻近像素点，并通过加权平均计算新位置的颜色值。结果更平滑，但在边缘处可能会有些模糊。  给定一个点 P(x,y)，我们想要计算其在图像中的灰度值（或颜色值）。假设这个点位于四个已知像素点Q11(x1,y1)，Q21(x2,y1)，Q12(x1,y2)，Q22(x2,y2)，所形成的矩形区域内，那么我们可以使用以下公式来计算 P点的灰度值 VP ：  其中wx是 P点相对于左侧边界的相对距离，wy是P点相对于上侧边界的相对距离， VQij表示 Qij点的灰度值。  像素映射与插值计算：对于新图像中的每一个像素，确定其在原图中的对应位置（通常是小数坐标）。使用选定的插值方法计算该位置的颜色值，并将此值赋给新图像中的相应像素。  边界处理：在插值过程中，如果计算出的坐标超出了原图边界，则需要采取措施防止访问越界内存区域。常见的做法是复制最近的有效像素值或者用其他方式填充边界外的像素。 3.2 图像旋转 **3.2.1 图像旋转实现原理**  一般图像的旋转是以图像的中心为原点，旋转一定的角度，也就是将图像上的所有像素都旋转一个相同的角度。旋转后图像的的大小一般会改变，即可以把转出显示区域的图像截去，或者扩大图像范围来显示所有的图像。图像的旋转变换也可以用矩阵变换来表示。  图像的旋转类似坐标平面中XOY点的旋转，如下图，点P以坐标原点O为旋转中心，逆时针旋转角度β后得到点Q：  IMG_256  图3.2.1 图像旋转实现原理示意图  设点P0(x0,y0)逆时针旋转θ角后的对应点为P(x,y)。那么，旋转前后点、的坐标分别是：  IMG_256  IMG_256  其矩阵表达式为：  IMG_256  其逆运算为：  IMG_256  **3.2.2 图像旋转实现流程**  确定旋转中心：通常选择图像的中心作为旋转中心，但也可以根据需求选择其他点。  构建旋转矩阵：旋转矩阵用于定义旋转的角度和方向。    应用几何变换：对于图像中的每一个像素 (x,y)，首先将其坐标转换为相对于旋转中心的坐标系（即减去旋转中心的坐标），然后应用旋转矩阵，最后再将结果转换回原始坐标系（即加上旋转中心的坐标）。    插值计算：由于旋转后的坐标通常是浮点数，并且可能不在整数网格上，因此需要使用插值方法来估算新位置的颜色值。常用的插值方法有最近邻插值、双线性插值和双三次插值等。  处理边界问题：旋转可能会导致部分原图区域超出新的图像边界。解决这个问题的方法包括裁剪、填充黑色或其他颜色、或者复制边缘像素。 4.图像加噪与滤波4.1 椒盐噪声 **4.1.1 椒盐噪声实现原理**  椒盐噪声是由图像传感器，传输信道，解码处理等产生的黑白相间的亮暗点噪声。所谓椒盐，椒就是黑，盐就是白，椒盐噪声就是在图像上随机出现黑色白色的像素。椒盐噪声是一种因为信号脉冲强度引起的噪声，产生该噪声的算法也比较简单。 椒盐噪声往往由图像切割引起，去除脉冲干扰及椒盐噪声最常用的算法是中值滤波。  E:\qq\1344075434\nt_qq\nt_data\Pic\2024-12\Ori\8f44efc2174531a4000c67d317ba4a2b.png  **4.1.2 椒盐噪声实现流程**  确定噪声密度：噪声密度是指图像中被噪声影响的像素比例。通常用两个参数来表示：一个是“盐”（即白色噪声）的比例，另一个是“胡椒”（即黑色噪声）的比例。这两个比例之和不应超过1。  随机选择像素位置：根据设定的噪声密度，在图像中随机选择一定数量的像素作为噪声点。这些像素将被设置为全白（盐）或全黑（胡椒）。  应用噪声：对于选定的像素位置，根据一定的概率将其值设为最大灰度值（对于8位图像来说就是255，代表白色）或者最小灰度值（0，代表黑色）。如果图像不是灰度图而是彩色图，则需要对每个颜色通道分别进行处理。 4.2 高斯噪声 **4.2.1 高斯噪声实现原理**  高斯噪声（Gaussian Noise）是一种常见的随机噪声，其概率密度函数服从高斯分布（也称为正态分布）。在图像处理中，高斯噪声通常用来模拟由于电子设备中的热噪声引起的图像退化。这种噪声的特点是每个像素的值都以原像素值为中心，在一定范围内波动，波动范围遵循高斯分布。  **4.2.2 高斯噪声实现流程**  确定噪声参数：   * 均值（Mean）：表示噪声的平均强度。如果均值为0，则表示噪声的平均效果是对图像没有影响。 * 标准差（Standard Deviation, 简称sigma或σ）：决定了噪声的强度和分布宽度。较大的标准差意味着更强烈的噪声，以及像素值变化范围更大。   生成随机噪声：使用高斯分布随机数生成器创建与图像大小相同的噪声矩阵。该矩阵中的每个元素都是从具有指定均值和标准差的高斯分布中抽取的随机数。  应用噪声到图像：将生成的噪声矩阵加到原始图像上。对于彩色图像，可以将噪声分别加到每个颜色通道上。注意确保结果图像的像素值仍然保持在合法范围内（例如，对于8位灰度图像，像素值应在0-255之间）。  裁剪或归一化：如果有像素值超出了允许的范围（比如小于0或大于255），则需要进行裁剪或者归一化处理，以保证最终输出图像的有效性。 4.3 泊松噪声 **4.3.1 泊松噪声实现原理**  泊松噪声（Poisson Noise）是一种统计噪声，其强度与信号的强度成正比，满足泊松分布。它在光子计数等低光条件下特别常见，因为光子到达探测器的过程可以被建模为泊松过程。泊松噪声的特点是它的方差等于其均值，这与高斯噪声不同，后者具有固定的方差。  **4.3.2 泊松噪声实现流程**  生成泊松噪声：对于图像中的每个像素，使用其灰度值作为参数 λ，从泊松分布中抽取一个随机数。这个随机数将作为该像素的新值，从而引入噪声。  应用噪声到图像：将生成的泊松噪声应用于原始图像，通常通过替换原始像素值来完成。需要注意的是，由于泊松分布只能产生非负整数值，因此对于浮点型图像或者需要保持特定范围的图像，可能需要进行适当的转换和裁剪。  处理极低亮度区域：在非常暗的区域（例如像素值接近0），泊松分布会趋向于产生更多的零值，这可能会导致这些区域完全变黑。根据具体应用场景，你可能需要考虑如何处理这种情况，比如设置一个最小阈值。 4.4 空域均值滤波 **4.4.1 空域均值滤波实现原理**  均值滤波（Mean Filtering）是一种常用的图像处理和信号处理技术，用于减少随机噪声。它的基本原理是对图像或信号的每个点计算一个邻域内的平均值来替代该点的原始值。对于二维图像，这个过程可以看作是将一个小窗口滑过整个图像，并在每个位置上用窗口内像素值的平均值替换中心像素的值。  **4.4.2 空域均值滤波实现流程**  定义滤波窗口大小：选择一个合适的窗口大小，通常是奇数如3x3, 5x5等。窗口大小的选择取决于需要平滑的程度和细节保留的需求。较大的窗口可以去除更多的噪声，但同时也会模糊更多的图像细节。  遍历图像：从图像的一个角开始，按照一定顺序（例如从左到右，从上到下）逐个像素地移动滤波窗口。  计算平均值：对于每一个中心像素，计算其周围由窗口定义的区域内的所有像素的平均灰度值（对于彩色图像，可能需要对每个颜色通道分别计算）。对于一个3x3的窗口，公式如下：  其中I(i,j)表示窗口内位置(i,j)处的像素强度值。  更新中心像素：用计算出的平均值替换原来的中心像素值。  重复操作：直到所有像素都被处理完毕。  边界处理：当滤波窗口靠近图像边缘时，部分窗口可能会超出图像边界。这时有几种处理方法可以选择，比如忽略这些超出边界的像素、复制最近的边界像素值、或者使用周期性边界条件等。 4.5 空域高斯滤波 **4.5.1 空域高斯滤波实现原理**  空域高斯滤波（Gaussian Filtering in Spatial Domain）是一种常见的图像平滑技术，它利用高斯函数来对图像进行加权平均处理。与均值滤波不同的是，高斯滤波器不仅考虑了邻域内像素的空间位置关系，还根据距离中心像素的远近赋予不同的权重，使得离中心越近的像素有更大的影响，而远处的像素影响较小。这种方法能够有效地减少噪声，同时尽量保留图像的边缘信息。  **4.5.2 空域高斯滤波实现流程**  高斯核构造：首先，需要构造一个二维高斯核（Gaussian Kernel），它是基于高斯分布的概率密度函数构建的。高斯核的大小通常是奇数（如3x3, 5x5等），以确保有一个明确的中心点。高斯核中的每个元素是根据以下公式计算得到的：  其中，σ是标准差，x和y是相对于中心点的位置坐标。为了简化计算，通常会预先计算好高斯核，并将其归一化，使得所有元素之和为1。  图像卷积：接着，将这个高斯核应用于图像的每一个像素上，通过卷积操作来计算新像素值。具体步骤如下：   * 遍历图像：从图像的一个角开始，按照一定顺序逐个像素地移动高斯核。 * 应用高斯核：对于每一个中心像素，使用高斯核对周围像素进行加权求和。即，将高斯核中的权重与对应位置的像素强度相乘后求和。 * 更新中心像素：用计算出的新值替换原来的中心像素值。   边界处理：当高斯核靠近图像边缘时，部分核可能会超出图像边界。这时需要采取适当的边界处理方法，比如：忽略超出边界的核区域；复制最近的边界像素值；使用周期性边界条件；或者采用其他填充策略。 4.6 空域中值滤波 **4.6.1 空域中值滤波实现原理**  空域高斯滤波（Gaussian Filtering in Spatial Domain）是一种常见的图像平滑技术，它利用高斯函数来对图像进行加权平均处理。与均值滤波不同的是，高斯滤波器不仅考虑了邻域内像素的空间位置关系，还根据距离中心像素的远近赋予不同的权重，使得离中心越近的像素有更大的影响，而远处的像素影响较小。这种方法能够有效地减少噪声，同时尽量保留图像的边缘信息。  **4.6.2 空域中值滤波实现流程**  定义滤波窗口：首先，定义一个滤波窗口或称为掩模（mask），通常是一个奇数大小的正方形区域，如3x3、5x5等。这个窗口将在整个图像上滑动，覆盖每个像素及其周围的邻域。  遍历图像：从图像的一个角开始，按照一定顺序（例如从左到右，从上到下）逐个像素地移动滤波窗口。对于图像边缘的像素，可能需要采用适当的边界处理方法，如忽略超出边界的像素、复制最近的边界像素值或者使用周期性边界条件等。  排序与选择中值：对于每一个中心像素，执行以下步骤：   * 收集像素值：将窗口内所有像素的灰度值（对于彩色图像可能是每个颜色通道的值）收集起来。 * 排序像素值：对这些像素值进行排序（升序或降序）。 * 选择中值：如果窗口大小为奇数，则直接选取排序后的中间值；如果窗口大小为偶数，则可以取中间两个值的平均值，但更常见的是调整窗口大小为奇数以避免这种情况。   更新中心像素：用计算出的中值替换原来的中心像素值。由于中值滤波器不是简单地对像素值求平均，而是选择了排序后的中间值，因此它可以有效地抑制离群点（即异常高的或低的像素值），这对于减少随机噪声特别有效。  重复操作：继续上述过程直到所有像素都被处理完毕。最终得到的图像是经过中值滤波处理后的版本，理论上具有更少的随机噪声并且较好地保持了边缘信息。 4.7 空域双边滤波 **4.7.1 空域双边滤波实现原理**  双边滤波（Bilateral Filtering）是一种非线性、边缘保持的平滑滤波器，它结合了空间邻近度和像素强度相似性的加权平均来处理图像。与传统的空域滤波器如均值滤波或高斯滤波不同，双边滤波不仅考虑了像素的空间位置，还考虑了像素值之间的差异，从而能够在有效减少噪声的同时很好地保持图像中的边缘和其他重要特征。  **4.7.2 空域双边滤波实现流程**  定义滤波窗口：首先，定义一个滤波窗口或称为掩模（mask），通常是一个奇数大小的正方形区域，如3x3、5x5等。这个窗口将在整个图像上滑动，覆盖每个像素及其周围的邻域。  遍历图像：从图像的一个角开始，按照一定顺序（例如从左到右，从上到下）逐个像素地移动滤波窗口。  计算权重：对于每一个中心像素 p，计算其周围所有像素 q 的权重。双边滤波使用两个独立的高斯函数来计算权重：   * 空间权重：根据像素间的欧几里得距离计算，由高斯核Gs决定。 * 强度权重：根据像素值之间的差异计算，由另一个高斯核Gr决定。   计算公式如下： 4.8 频域低通滤波 **4.8.1 频域低通滤波实现原理**  频域低通滤波（Frequency Domain Low-Pass Filtering）是图像处理中用于平滑图像、减少噪声的一种技术。它通过在频率域内操作，允许低频成分通过而抑制高频成分来实现。  **4.8.2 频域低通滤波实现流程**  图像预处理   * 灰度化：如果输入图像是彩色的，通常需要先将其转换为灰度图像。 * 填充：为了减少边界效应和伪影，可以对图像进行零填充（zero-padding），即在图像周围添加一圈零值像素。这有助于避免快速傅里叶变换（FFT）时由于周期性假设带来的问题。   转换到频率域：使用快速傅里叶变换（FFT）将图像从空间域转换到频率域。这个过程会生成一个复数矩阵，表示原始图像的不同频率分量。  其中 f(x,y) 是原始图像，F(u,v) 是其对应的频率域表示。  中心化：为了便于处理，通常需要将零频率成分移动到频谱的中心位置。这可以通过乘以一个特定的相位因子来实现，例如使用(-1)x+y对原图像进行预处理，然后再做FFT。  应用低通滤波器  选择合适的低通滤波器，并应用到频率域图像上。常用的低通滤波器包括理想低通滤波器（ILPF）、巴特沃斯低通滤波器（BLPF）和高斯低通滤波器（GLPF）。每种滤波器都有自己的传输函数 H(u,v)，用来确定哪些频率应该被保留或抑制:  理想低通滤波器：    巴特沃斯低通滤波器：    高斯低通滤波器:    逆变换：应用完滤波器后，需要通过逆快速傅里叶变换（IFFT）将图像转换回空间域。  去中心化：如果之前进行了中心化处理，则现在需要去除中心化效果。对于之前提到的相位因子，可以通过再次乘以相同的相位因子来完成。  后处理：最后，可能还需要对结果进行一些后处理，如取绝对值、归一化等，确保输出图像的像素值在合理的范围内（例如0到255之间）。 4.9 频域高通滤波 **4.9.1 频域高通滤波实现原理**  频域高通滤波（Frequency Domain High-Pass Filtering）是一种用于增强图像中高频成分的技术，如边缘、线条和其他快速变化的细节，同时抑制低频成分（平滑区域）。这种技术在图像处理中常用于锐化图像或进行边缘检测。与低通滤波相反。  **4.9.2 频域高通滤波实现流程**  与低通滤波相比，使用高通滤波器，保留高频率部分，丢失低频率部分，通过傅里叶变换将时域图像转换到频域图像，并移动到中心位置，进行滤波处理并进行傅里叶反转换。 5.边缘提取5.1 Sobel算子 **5.1.1 Sobel算子实现原理**  Sobel算子是一种广泛应用于图像处理和计算机视觉领域的边缘检测算子。它通过计算图像中每个像素点的梯度来确定边缘的位置，利用了图像中亮度变化的特性。Sobel算子不仅考虑了水平方向上的梯度变化，也考虑了垂直方向上的梯度变化，从而可以更准确地捕捉到边缘信息。  **5.1.2 Sobel算子实现流程**  核定义：Sobel算子使用两个3x3的卷积核（也称为滤波器或掩模），分别用于检测水平方向和垂直方向的边缘。  水平方向（Gx）：    垂直方向（Gy）：    卷积操作：对于图像中的每一个像素，使用上述两个核进行卷积操作，以计算该像素在水平和垂直方向上的梯度分量 Gx和Gy，具体步骤如下：  将核与图像的相应区域（即以当前像素为中心的3x3邻域）相乘。  对所得结果求和，得到该像素位置处的Gx和Gy值。  梯度幅度计算：根据Gx和Gy值可以计算出每个像素点的梯度幅度 M 和方向 θ  梯度幅度M：    梯度方向 θ：    阈值化：为了突出边缘并抑制非边缘区域，通常会对梯度幅度应用一个阈值。低于某个设定阈值的像素被认为是背景或非边缘部分，并被设置为零；高于阈值的像素则被认为是边缘的一部分。  边缘图生成：最后，将所有满足条件的像素标记出来，形成最终的边缘图。这个图像通常只包含二进制值（0或1），表示是否存在边缘。 5.2 robert算子 **5.2.1 robert算子实现原理**  Roberts算子又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。常用来处理具有陡峭的低噪声图像，当图像边缘接近于正45度或负45度时，该算法处理效果更理想。其缺点是对边缘的定位不太准确，提取的边缘线条较粗。  **5.2.2 robert算子实现流程**  Robert算子使用两个2x2的卷积核（也称为滤波器或掩模），分别用于检测水平方向和垂直方向的边缘。  水平方向（Gx）：    垂直方向（Gy）：    后面的操作与Sobel算子的后续操作基本一致，不再过多赘述。 5.3prewitt算子 **5.3.1 prewitt算子实现原理**  Prewitt算子是一种用于边缘检测的图像处理技术，它通过计算图像中每个像素点的梯度来确定边缘的位置。与Sobel算子类似，Prewitt算子使用两个3x3的卷积核（也称为滤波器或掩模）分别检测水平和垂直方向上的边缘变化。然而，Prewitt算子的权重分配相对均匀，这使得它在某些情况下对噪声更敏感。  **5.3.2 prewitt算子实现流程**  核定义：Prewitt算子使用两个3x3的卷积核，分别用于检测水平方向和垂直方向的边缘：  水平方向（Gx）：    垂直方向（Gy）：    后面的操作与前面的两种算子的后续操作基本一致，不再过多赘述。  5.4拉普拉斯算子 **5.4.1 拉普拉斯算子实现原理**  拉普拉斯算子（Laplacian Operator）是一种用于检测图像中边缘和细节的二阶微分算子。它通过计算图像中的灰度值变化率来突出图像中的局部极大值或极小值，从而识别出边缘和其他显著特征。拉普拉斯算子在图像处理中常用于锐化图像、增强边缘以及进行图像分割等任务。  **5.4.2 拉普拉斯算子实现流程**  核定义：拉普拉斯算子通常用一个3x3的小卷积核表示，该核用于计算每个像素点处的二阶导数。最常用的两种形式如下：  标准拉普拉斯算子：    另一种常见形式：    卷积操作：对于图像中的每一个像素，使用上述核进行卷积操作，以计算该像素位置处的拉普拉斯响应。具体步骤如下：将核与图像的相应区域（即以当前像素为中心的3x3邻域）相乘。对所得结果求和，得到该像素位置处的拉普拉斯响应值 L(x,y)。  零交叉检测（Zero-Crossing Detection）：拉普拉斯算子的一个重要特性是它可以用来检测边缘，这依赖于所谓的“零交叉”概念。在理想情况下，边缘对应于图像亮度从亮到暗或从暗到亮的过渡区域，在这些地方，一阶导数达到峰值，而二阶导数则穿过零点。因此，可以通过查找拉普拉斯响应图中的零交叉点来定位边缘。  阈值化：为了进一步确定哪些零交叉点真正代表边缘，通常需要应用一个阈值。只有那些绝对值大于设定阈值的零交叉点才会被标记为边缘点。这样可以减少噪声引起的伪边缘，并确保只保留显著的边缘信息。  边缘图生成：最后，将所有满足条件的像素标记出来，形成最终的边缘图。这个图像通常只包含二进制值（0或1），表示是否存在边缘。 6.目标提取6.1目标提取 **6.1.1 目标提取实现原理**  运用图像乘法运算，提取目标。图像乘法运算指的是将两个图像（或一个图像与一个掩模）对应像素值相乘的操作。这种操作可以用于强调或抑制图像中的某些部分。在目标提取的应用场景下，通常是将原图像与一个设计好的掩模（mask）进行乘法运算，以突出目标区域并抑制背景。  **6.1.2 目标提取实现流程**  将原始图像 I(x,y) 与掩模 M(x,y) 进行逐像素相乘：  其中，O(x,y) 是输出图像，x 和 y 分别表示像素的位置坐标。对于二值掩模，当 M(x,y)=1 时，对应的像素值保持不变；当 M(x,y)=0 时，对应的像素值变为0，即该位置的像素被抑制。 7.特征提取7.1 lBP特征提取 **7.1.1** LBP**目标提取实现原理**  LBP（Local Binary Patterns，局部二值模式）是一种用于纹理分析和描述的简单而有效的图像处理方法。它最初被设计用来进行纹理分类，但后来也被广泛应用于人脸识别、目标检测等领域。LBP通过比较图像中每个像素点与其邻域内的像素点之间的关系来构建特征描述符。  **7.1.2 lbp目标提取实现流程**  预处理：读取并可能调整输入图像大小，将其转换为灰度图，以便于LBP计算。  生成LBP特征：对整个图像或感兴趣的区域应用LBP操作，得到LBP特征图。  分割或检测：可以通过设定阈值直接从LBP特征图中分割出目标，或者训练一个分类器来区分目标和非目标区域。  后处理：对初步的结果进行形态学操作（如膨胀、腐蚀），去除小的连通域，填充孔洞等，优化最终的目标提取结果。  定位和识别：如果有需要，还可以进一步使用机器学习模型或其他技术来精确定位目标并对其进行分类。 7.2 hog特征提取 **7.2.1 hog目标提取实现原理**  HOG（Histogram of Oriented Gradients，方向梯度直方图）特征描述符是一种用于物体检测的计算机视觉技术。它通过计算和统计图像局部区域的梯度方向直方图来描述目标的边缘和形状信息。HOG特征最初是由Navneet Dalal和Bill Triggs在2005年的CVPR会议上提出的，并广泛应用于行人检测领域。  **7.2.2 hog目标提取实现流程**  在实际应用中，HOG特征通常与滑动窗口搜索策略相结合，用以定位图像中的目标位置。具体步骤如下：  设置滑动窗口：定义一个初始大小的窗口，在图像上逐行逐列地移动，每次移动一定的步长。  提取HOG特征：对于每个窗口内的子图像，按照上述过程提取HOG特征。  训练分类器：使用大量的正样本（包含目标的图像）和负样本（不包含目标的图像）来训练一个二分类器，如支持向量机（SVM）。训练好的分类器能够区分目标和非目标。  检测目标：将测试图像通过滑动窗口分割成许多子图像，提取它们的HOG特征，并用训练好的分类器判断哪些窗口对应于目标。  后处理：由于滑动窗口可能会产生重叠的检测结果，因此需要进行非极大值抑制（Non-Maximum Suppression, NMS）等操作来选择最可能的目标位置。 8.模型预测8.1模型训练与应用 使用了TensorFlow和Keras库来构建、训练和评估一个基于ResNet50的卷积神经网络（CNN），用于对CUB\_200\_2011数据集中的鸟类图片进行分类。之后在matlab中调用.h5文件模型进行图形预测。  Figure_1  图8.1.1 模型训练过程中的准确率和损失值随epoch的变化情况 （三）算法代码及功能注释1灰度直方图、直方图均衡化和直方图匹配(规定化)1.1 灰度直方图  |  | | --- | | function my\_imhist(img)  % 输入参数img为灰度图像（二维矩阵形式）  % 灰度级范围，对于常见的8位灰度图像是0到255  grayLevels = 0:255;  % 用于统计每个灰度级像素个数的数组，初始化为全0  histogram = zeros(1, 256);  % 获取图像的尺寸，height表示行数（高度），width表示列数（宽度）  [height, width] = size(img);  % 嵌套循环遍历图像的每一个像素  for row = 1:height  for col = 1:width  % 获取当前像素的灰度值，由于Matlab索引从1开始，这里无需额外处理索引偏移  pixelValue = img(row, col);  % 对应灰度级的像素个数加1  histogram(pixelValue + 1) = histogram(pixelValue + 1) + 1;  end  end  % 绘制直方图，以灰度级为横坐标，对应像素个数为纵坐标  bar(grayLevels, histogram);  % 设置x轴标签  xlabel('灰度值');  % 设置y轴标签  ylabel('像素个数');  % 设置图表标题  title('灰度直方图');  end |  1.2 直方图均衡化  |  | | --- | | function eq\_gray\_img = my\_histeq(gray\_img)  % 获取图像尺寸和灰度级别  [rows, cols] = size(gray\_img);  levels = 256; % 对于8位图像  % 计算直方图  hist = imhist(gray\_img);  % 计算累积分布函数 (CDF)  cdf = cumsum(hist);  % 线性化 CDF 到 [0, levels-1]  cdf\_min = min(cdf(cdf > 0)); % 找到第一个非零值  cdf\_normalized = round((cdf - cdf\_min) / (rows \* cols - cdf\_min) \* (levels - 1));  % 创建查找表 (LUT) 并应用到图像  lut = uint8(1:levels);  lut(:) = cdf\_normalized + 1;  eq\_gray\_img = lut(double(gray\_img) + 1);  % 转换回uint8类型  eq\_gray\_img = uint8(eq\_gray\_img);  end |  1.3 直方图匹配  |  | | --- | | function matched\_image = my\_histmatch(input\_image, target\_image)  % 检查输入图像是否为灰度图，若不是则尝试转换为灰度图（若转换失败则报错）  if size(input\_image, 3) > 1  input\_image = rgb2gray(input\_image);  if isempty(input\_image)  error('无法将输入的原始彩色图像转换为灰度图像，请检查图像数据');  end  end  if size(target\_image, 3) > 1  target\_image = rgb2gray(target\_image);  if isempty(target\_image)  error('无法将输入的目标彩色图像转换为灰度图像，请检查图像数据');  end  end  % 获取输入图像尺寸，并检查尺寸合法性，避免后续出现越界访问  [input\_height, input\_width] = size(input\_image);  [target\_height, target\_width] = size(target\_image);  if input\_height <= 0 || input\_width <= 0 || target\_height <= 0 || target\_width <= 0  error('输入的图像尺寸不合法，请检查图像数据');  end  % 若目标图像与原始图像尺寸不一致，进行等比例缩放使其尺寸适配（这里采用简单的双线性插值缩放方法）  % 实际应用中可根据具体需求选择更合适的缩放算法或进行更复杂的图像预处理  if input\_height ~= target\_height || input\_width ~= target\_width  input\_image = imresize(input\_image, [target\_height, target\_width], 'bilinear');  [input\_height, input\_width] = size(input\_image);  end  % 步骤一：计算原始图像的直方图和累计分布函数（CDF）  input\_hist = zeros(1, 256); % 初始化原始图像直方图数组  for row = 1:input\_height  for col = 1:input\_width  pixel\_value = input\_image(row, col) + 1; % Matlab索引从1开始，灰度值范围0-255需加1对应索引  input\_hist(pixel\_value) = input\_hist(pixel\_value) + 1;  end  end  input\_hist = input\_hist / (input\_height \* input\_width); % 归一化直方图，得到概率分布  input\_cdf = cumsum(input\_hist); % 计算原始图像累计分布函数  % 步骤二：计算目标图像的直方图和累计分布函数（CDF）  target\_hist = zeros(1, 256); % 初始化目标图像直方图数组  for row = 1:target\_height  for col = 1:target\_width  pixel\_value = target\_image(row, col) + 1;  target\_hist(pixel\_value) = target\_hist(pixel\_value) + 1;  end  end  target\_hist = target\_hist / (target\_height \* target\_width); % 归一化直方图  target\_cdf = cumsum(target\_hist); % 计算目标图像累计分布函数  % 步骤三：建立映射关系  mapping = zeros(1, 256); % 用于存储从原始图像灰度值到目标图像灰度值的映射  for i = 1:256  % 找到目标图像CDF中与原始图像当前灰度级的CDF值最接近的位置  [~, idx] = min(abs(target\_cdf - input\_cdf(i)));  mapping(i) = idx - 1; % 由于Matlab索引从1开始，这里要减去1得到实际灰度值（0-255范围）  end  % 步骤四：根据映射关系生成匹配后的图像  matched\_image = zeros(size(input\_image), class(input\_image)); % 初始化匹配后的图像  for row = 1:input\_height  for col = 1:input\_width  pixel\_value = input\_image(row, col) + 1;  matched\_image(row, col) = mapping(pixel\_value);  end  end  matched\_image = uint8(matched\_image); % 将匹配后的图像转换为合适的无符号8位整数类型（常见灰度图像类型）  end |  2.灰度化与对比度增强2.1 灰度化  |  | | --- | | function grayImage = my\_rgb2gray(rgbImage)  % CUSTOM\_RGB2GRAY 将RGB图像转换为灰度图像。  % 输入:  % rgbImage - 输入的RGB图像，尺寸为MxNx3。  % 输出:  % grayImage - 输出的灰度图像，尺寸为MxN。  % 确保输入是一个三维数组  if size(rgbImage, 3) ~= 3  error('输入图像必须是RGB格式');  end  % 定义RGB到灰度的转换系数  weights = [0.2989, 0.5870, 0.1140];  % 将RGB图像的每个通道乘以其对应的权重，然后相加得到灰度图像  grayImage = rgbImage(:,:,1) \* weights(1) + ...  rgbImage(:,:,2) \* weights(2) + ...  rgbImage(:,:,3) \* weights(3);  % 确保输出数据类型与输入一致  grayImage = uint8(grayImage);  end |  2.2 线性变换  |  | | --- | | function transformed\_img = my\_linear\_transform(img, alpha, beta)  % MY\_LINEAR\_TRANSFORM 对图像应用线性变换。  %  % 输入:  % img - 输入灰度图像。  % alpha - 放大系数。  % beta - 偏移量。  %  % 输出:  % transformed\_img - 变换后的图像。  % 将图像转换为 double 类型以确保精度  img\_double = double(img);  % 应用线性变换  transformed\_img = alpha \* img\_double + beta;  % 裁剪超出 [0, 255] 范围的值  transformed\_img(transformed\_img < 0) = 0;  transformed\_img(transformed\_img > 255) = 255;  % 转换回 uint8 类型  transformed\_img = uint8(transformed\_img);  end |  2.3 对数变换  |  | | --- | | function transformed\_img = my\_log\_transform(img)  % MY\_LOG\_TRANSFORM 对图像应用对数变换。  %  % 输入:  % img - 输入灰度图像。  %  % 输出:  % transformed\_img - 变换后的图像。  % 将图像转换为 double 类型以确保精度  img\_double = double(img);  % 计算对数变换常数 c  c = 255 / log(1 + max(img\_double(:)));  % 应用对数变换  transformed\_img = c \* log(1 + img\_double);  % 转换回 uint8 类型  transformed\_img = uint8(transformed\_img);  end |  2.4 指数变换  |  | | --- | | function transformed\_img = my\_exponential\_transform(img, gamma)  % MY\_EXPONENTIAL\_TRANSFORM 对图像应用指数变换。  %  % 输入:  % img - 输入灰度图像。  % gamma - 指数变换的伽马值。  %  % 输出:  % transformed\_img - 变换后的图像。  % 将图像转换为 double 类型以确保精度，并归一化到 [0, 1] 范围  img\_normalized = double(img) / 255;  % 应用指数变换  transformed\_img = 255 \* (img\_normalized .^ gamma);  % 转换回 uint8 类型  transformed\_img = uint8(transformed\_img);  end |  3.图像缩放与旋转3.1 图像缩放  |  | | --- | | function value = bilinear\_interpolation(img, x, y)  % 获取图像尺寸  [m, n] = size(img);  % 向上取整和向下取整  x1 = floor(x); % 向下  x2 = ceil(x); % 向上  y1 = floor(y);  y2 = ceil(y);  % 确保不超出图像边界  x1 = max(1, min(x1, n-1));  x2 = max(1, min(x2, n));  y1 = max(1, min(y1, m-1));  y2 = max(1, min(y2, m));  % 如果点位于图像边界，则直接返回该点的值  if x1 == x2 && y1 == y2  value = img(y1, x1);  return;  end  % 获取四个邻近点的值  Q11 = img(y1, x1);  Q21 = img(y2, x1);  Q12 = img(y1, x2);  Q22 = img(y2, x2);  % 计算权重  wx = x - x1;  wy = y - y1;  % 应用双线性插值公式  value = (Q11 \* (x2-x) \* (y2-y) + ...  Q21 \* (x-x1) \* (y2-y) + ...  Q12 \* (x2-x) \* (y-y1) + ...  Q22 \* (x-x1) \* (y-y1));  end |  3.2 图像旋转  |  | | --- | | function rotatedImage = my\_imrotate(image, angle, varargin)  % MY\_IMROTATE 旋转图像。  % 输入:  % image - 输入的图像。  % angle - 旋转的角度（度数），正为逆时针方向。  % varargin - 可选参数，可以包括插值方法 ('nearest', 'bilinear', 'bicubic') 和是否裁剪 ('crop')。  % 默认设置  method = 'bilinear'; % 默认使用双线性插值  crop = false; % 默认不裁剪  % 解析可变输入参数  if mod(length(varargin), 2) ~= 0  error('参数必须成对出现');  end  for i = 1:2:length(varargin)  switch lower(varargin{i})  case 'method'  method = varargin{i+1};  case 'crop'  crop = true;  otherwise  error(['未知参数: ' varargin{i}]);  end  end  % 获取原始图像尺寸  [rows, cols] = size(image);  % 计算旋转中心  center\_x = (cols + 1) / 2;  center\_y = (rows + 1) / 2;  % 将角度转换为弧度  theta = deg2rad(angle);  % 创建旋转矩阵  cos\_theta = cos(theta);  sin\_theta = sin(theta);  % 计算旋转后的最大尺寸  max\_size = ceil(sqrt(rows^2 + cols^2));  output\_rows = max\_size;  output\_cols = max\_size;  % 初始化旋转后的图像  rotatedImage = zeros(output\_rows, output\_cols, class(image));  % 创建输出图像的坐标网格，并移动到旋转中心  [X\_out, Y\_out] = meshgrid(1:output\_cols, 1:output\_rows);  X\_out\_centered = X\_out - (output\_cols + 1) / 2;  Y\_out\_centered = Y\_out - (output\_rows + 1) / 2;  % 应用旋转矩阵的逆变换，将旋转后的坐标映射回原图坐标  X\_in = X\_out\_centered \* cos\_theta + Y\_out\_centered \* sin\_theta + center\_x;  Y\_in = -X\_out\_centered \* sin\_theta + Y\_out\_centered \* cos\_theta + center\_y;  % 找到映射后的坐标落在原始图像范围内的索引  valid\_idx = (X\_in >= 1) & (X\_in <= cols) & ...  (Y\_in >= 1) & (Y\_in <= rows);  % 根据选择的方法进行插值  switch lower(method)  case 'nearest'  % 最近邻插值逻辑略...  case 'bilinear'  % 双线性插值  % 对有效索引进行插值  X\_valid = X\_in(valid\_idx);  Y\_valid = Y\_in(valid\_idx);  valid\_coords = find(valid\_idx);  % 使用 interp2 进行双线性插值  rotatedImage(valid\_coords) = interp2(1:cols, 1:rows, double(image), X\_valid, Y\_valid, 'linear');  case 'bicubic'  % 双三次插值实现略...  error('双三次插值尚未实现');  otherwise  error('未知的插值方法');  end  % 如果需要裁剪，执行相应的逻辑  if crop  % 找到非零像素的边界框  [bw\_rows, bw\_cols] = find(rotatedImage ~= 0);  if ~isempty(bw\_rows) && ~isempty(bw\_cols)  rotatedImage = rotatedImage(min(bw\_rows):max(bw\_rows), min(bw\_cols):max(bw\_cols));  end  end  % 确保返回的图像与输入图像具有相同的维度  if ndims(image) == 3 && size(image, 3) > 1  rotatedImage = permute(rotatedImage, [1 2 3]); % 确保彩色图像是三维数组  elseif ndims(image) == 2  rotatedImage = rotatedImage(:,:,1); % 确保灰度图像是二维数组  end  end |  4.图像加噪与滤波4.1 椒盐噪声  |  | | --- | | function noisyImage = addSaltAndPepperNoise(image, density)  [height, width] = size(image);  numPixels = height \* width;  numNoisePixels = round(numPixels \* density);  noisyImage = image;  % 生成随机索引，确保每个噪声像素点只被选择一次（更精确的方式）  noisePixelIndices = randperm(numPixels, numNoisePixels);  % 分别统计盐噪声和椒噪声的数量（这里简单按照等概率分配，可调整）  numSaltPixels = floor(numNoisePixels / 2);  numPepperPixels = numNoisePixels - numSaltPixels;  % 生成盐噪声像素索引  saltPixelIndices = noisePixelIndices(1:numSaltPixels);  % 生成椒噪声像素索引  pepperPixelIndices = noisePixelIndices(numSaltPixels + 1:end);  % 将对应索引的像素设置为盐噪声（1）或椒噪声（0）  [ySalt, xSalt] = ind2sub([height, width], saltPixelIndices);  noisyImage(sub2ind(size(image), ySalt, xSalt)) = 1;  [yPepper, xPepper] = ind2sub([height, width], pepperPixelIndices);  noisyImage(sub2ind(size(image), yPepper, xPepper)) = 0;  end |  4.2 高斯噪声  |  | | --- | | function noisyImage = addGaussianNoise(image, mean, variance)  [height, width] = size(image);  noisyImage = zeros(size(image));  % 使用向量化操作添加高斯噪声，效率更高且更符合数学原理  noiseMatrix = mean + variance \* randn(height, width);  % 处理边界情况，确保像素值在合理范围[0, 1]内，采用更平滑的截断方式（避免硬截断）  clippedNoiseMatrix = max(0, min(1, noiseMatrix));  noisyImage = max(0, min(1, image + clippedNoiseMatrix));  end |  4.3 泊松噪声  |  | | --- | | function noisyImage = poissonNoiseImplementation(image)  [height, width] = size(image);  noisyImage = zeros(size(image));  % 对于每个像素，根据其灰度值按照泊松分布生成噪声  for y = 1:height  for x = 1:width  lambda = image(y, x);  % 使用内置的泊松分布随机数生成函数，这里考虑了像素值不能为负的情况  if lambda > 0  noisyImage(y, x) = max(0, poissrnd(lambda));  else  noisyImage(y, x) = 0;  end  end  end  end |  4.4 空域均值滤波  |  | | --- | | function filtered\_image = my\_mean\_filter(input\_image, kernel\_size)  % MY\_MEAN\_FILTER 自定义实现均值滤波  % input\_image 为输入的图像数据（可以是灰度图或者彩色图等，根据实际需求调整处理逻辑）  % kernel\_size 为滤波器尺寸，格式为[行尺寸 列尺寸]，例如[3 3]表示3x3的滤波器  % 获取输入图像的尺寸信息  [image\_rows, image\_cols, ~] = size(input\_image);  % 计算滤波器半尺寸，用于后续图像边界填充及窗口滑动  half\_kernel\_row = floor(kernel\_size(1) / 2);  half\_kernel\_col = floor(kernel\_size(2) / 2);  % 对输入图像进行边界填充，填充方式选择对称填充，以处理图像边缘像素  padded\_image = padarray(input\_image, [half\_kernel\_row half\_kernel\_col], 'symmetric');  % 初始化滤波后的图像，尺寸与原图像一致  filtered\_image = zeros(image\_rows, image\_cols, size(input\_image, 3));  % 循环遍历图像的每个像素（针对彩色图像的每个通道分别处理，如果是灰度图可简化外层循环）  for channel = 1:size(input\_image, 3)  for row = 1:image\_rows  for col = 1:image\_cols  % 提取以当前像素为中心的窗口区域  window = padded\_image(row:row + kernel\_size(1) - 1, col:col + kernel\_size(2) - 1, channel);  % 计算窗口内像素的平均值，作为当前像素滤波后的结果  filtered\_image(row, col, channel) = mean(window(:));  end  end  end  end |  4.5 空域高斯滤波  |  | | --- | | function filtered\_image = my\_gaussian\_filter(input\_image, kernel\_size, sigma)  % MY\_GAUSSIAN\_FILTER 自定义实现高斯滤波  % input\_image 为输入的图像数据  % kernel\_size 为滤波器尺寸，例如[3 3]  % sigma 为高斯分布的标准差，控制滤波的平滑程度  [image\_rows, image\_cols, ~] = size(input\_image);  half\_kernel\_row = floor(kernel\_size(1) / 2);  half\_kernel\_col = floor(kernel\_size(2) / 2);  padded\_image = padarray(input\_image, [half\_kernel\_row half\_kernel\_col], 'symmetric');  filtered\_image = zeros(image\_rows, image\_cols, size(input\_image, 3));  % 生成二维高斯核  [x, y] = meshgrid(-half\_kernel\_col:half\_kernel\_col, -half\_kernel\_row:half\_kernel\_row);  gaussian\_kernel = exp(-(x.^2 + y.^2) / (2 \* sigma^2));  gaussian\_kernel = gaussian\_kernel / sum(gaussian\_kernel(:));  % 针对每个像素（及彩色图像的每个通道）进行滤波操作  for channel = 1:size(input\_image, 3)  for row = 1:image\_rows  for col = 1:image\_cols  window = padded\_image(row:row + kernel\_size(1) - 1, col:col + kernel\_size(2) - 1, channel);  % 将窗口内像素与高斯核进行加权求和，得到滤波后的像素值  filtered\_image(row, col, channel) = sum(sum(window.\* gaussian\_kernel));  end  end  end  end |    4.6 空域中值滤波  |  | | --- | | function filtered\_image = my\_median\_filter(input\_image)  % MY\_MEDIAN\_FILTER 自定义实现中值滤波  % input\_image 为输入的图像数据  [image\_rows, image\_cols, ~] = size(input\_image);  % 这里以3x3滤波器为例，设置边界填充尺寸，可根据实际需求扩展为通用尺寸输入参数  pad\_size = 1;  padded\_image = padarray(input\_image, [pad\_size pad\_size], 'symmetric');  filtered\_image = zeros(image\_rows, image\_cols, size(input\_image, 3));  % 遍历图像像素（及各通道）进行中值滤波  for channel = 1:size(input\_image, 3)  for row = 1:image\_rows  for col = 1:image\_cols  window = padded\_image(row:row + 2 \* pad\_size, col:col + 2 \* pad\_size, channel);  window\_vec = window(:);  sorted\_vec = sort(window\_vec);  % 取排序后向量的中间值作为中值滤波结果  filtered\_image(row, col, channel) = sorted\_vec(floor(length(sorted\_vec) / 2 + 1));  end  end  end  end |  4.7 空域双边滤波  |  | | --- | | function filtered\_image = my\_bilateral\_filter(input\_image, sigma\_d, sigma\_r)  % MY\_BILATERAL\_FILTER 自定义实现双边滤波  % input\_image 为输入的图像数据  % sigma\_d 为空间域标准差，影响空间距离对滤波权重的作用  % sigma\_r 为值域标准差，影响像素值差异对滤波权重的作用  [image\_rows, image\_cols, ~] = size(input\_image);  % 根据空间域标准差大致确定边界填充尺寸，实际应用可进一步优化此计算方式  pad\_size = floor(3 \* sigma\_d);  padded\_image = padarray(input\_image, [pad\_size pad\_size], 'symmetric');  filtered\_image = zeros(image\_rows, image\_cols, size(input\_image, 3));  % 遍历图像像素（及各通道）进行双边滤波操作  for channel = 1:size(input\_image, 3)  for row = 1:image\_rows  for col = 1:image\_cols  window = padded\_image(row:row + 2 \* pad\_size, col:col + 2 \* pad\_size, channel);  center\_value = input\_image(row, col, channel);  spatial\_weights = zeros(2 \* pad\_size + 1, 2 \* pad\_size + 1);  range\_weights = zeros(2 \* pad\_size + 1, 2 \* pad\_size + 1);  combined\_weights = zeros(2 \* pad\_size + 1, 2 \* pad\_size + 1);  sum\_weights = 0;  sum\_weighted\_values = 0;  [x, y] = meshgrid(-pad\_size:pad\_size, -pad\_size:pad\_size);  spatial\_weights = exp(-(x.^2 + y.^2) / (2 \* sigma\_d^2));  % 计算空间权重、值域权重，并结合得到综合权重，进而计算滤波后的像素值  for i = 1:size(window, 1)  for j = 1:size(window, 2)  range\_weights(i, j) = exp(-(window(i, j) - center\_value)^2 / (2 \* sigma\_r^2));  combined\_weights(i, j) = spatial\_weights(i, j) \* range\_weights(i, j);  sum\_weights = sum\_weights + combined\_weights(i, j);  sum\_weighted\_values = sum\_weighted\_values + combined\_weights(i, j) \* window(i, j);  end  end  filtered\_image(row, col, channel) = sum\_weighted\_values / sum\_weights;  end  end  end  end |  4.8 频域低通滤波  |  | | --- | | function H = butterworthLowpassFilter(M, N, cutoffFreq, n)  % BUTTERWORTHLOWPASSFILTER 生成巴特沃斯低通滤波器的传递函数  % M, N 分别为图像频谱的行数和列数  % cutoffFreq 为截止频率  % n 为滤波器阶数  [U, V] = meshgrid(-fix(N/2):ceil(N/2)-1, -fix(M/2):ceil(M/2)-1);  D = sqrt(U.^2 + V.^2);  H = 1./ (1 + (D / cutoffFreq).^(2\*n));  end  function H = idealLowpassFilter(M, N, cutoffFreq)  % IDEALLOWPASSFILTER 生成理想低通滤波器的传递函数  % M, N 分别为图像频谱的行数和列数  % cutoffFreq 为截止频率  [U, V] = meshgrid(-fix(N/2):ceil(N/2)-1, -fix(M/2):ceil(M/2)-1);  D = sqrt(U.^2 + V.^2);  H = double(D <= cutoffFreq);  end  function H = gaussianLowpassFilter(M, N, cutoffFreq)  % GAUSSIANLOWPASSFILTER 生成高斯低通滤波器的传递函数  % M, N 分别为图像频谱的行数和列数  % cutoffFreq 为截止频率  [U, V] = meshgrid(-fix(N/2):ceil(N/2)-1, -fix(M/2):ceil(M/2)-1);  D = sqrt(U.^2 + V.^2);  H = exp(-(D.^2) / (2 \* cutoffFreq^2));  end |  4.9 频域高通滤波  |  | | --- | | function noisyImage = poissonNoiseImplementation(image)  [height, width] = size(image);  noisyImage = zeros(size(image));  % 对于每个像素，根据其灰度值按照泊松分布生成噪声  for y = 1:height  for x = 1:width  lambda = image(y, x);  % 使用内置的泊松分布随机数生成函数，这里考虑了像素值不能为负的情况  if lambda > 0  noisyImage(y, x) = max(0, poissrnd(lambda));  else  noisyImage(y, x) = 0;  end  end  end  end  % 定义理想高通滤波器  function H = idealHighpassFilter(M, N, D0)  [U, V] = meshgrid(0:N-1, 0:M-1);  U = mod(U, N) - floor(N/2);  V = mod(V, M) - floor(M/2);  D = sqrt(U.^2 + V.^2);  H = double(D >= D0);  end  % 定义高斯高通滤波器  function H = gaussianHighpassFilter(M, N, D0)  [U, V] = meshgrid(0:N-1, 0:M-1);  U = mod(U, N) - floor(N/2);  V = mod(V, M) - floor(M/2);  D = sqrt(U.^2 + V.^2);  H = 1 - exp(-(D.^2) / (2\*(D0^2)));  end |  5.边缘提取5.1 Sobel算子 表格 17 Sobel代码   |  | | --- | | function edge\_img = my\_sobel\_edge\_detection(gray\_img)  % 获取输入灰度图像的尺寸  [height, width] = size(gray\_img);  % 初始化用于存储水平和垂直方向卷积结果的矩阵  gx = zeros(height, width);  gy = zeros(height, width);  % 定义Sobel算子卷积核（水平方向和垂直方向）  sobelKernelX = [-1 -2 -1; 0 0 0; 1 2 1];  sobelKernelY = [-1 0 1; -2 0 2; -1 0 1];  % 对图像进行边界扩充，以便处理边缘像素（采用复制边界的方式扩充）  padded\_gray\_img = padarray(gray\_img, [1, 1], 'replicate');  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（水平方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  gx(row, col) = sobelKernelX(1, 1) \* padded\_gray\_img(row, col) + sobelKernelX(1, 2) \* padded\_gray\_img(row, col + 1) + sobelKernelX(1, 3) \* padded\_gray\_img(row, col + 2) +...  sobelKernelX(2, 1) \* padded\_gray\_img(row + 1, col) + sobelKernelX(2, 2) \* padded\_gray\_img(row + 1, col + 1) + sobelKernelX(2, 3) \* padded\_gray\_img(row + 1, col + 2) +...  sobelKernelX(3, 1) \* padded\_gray\_img(row + 2, col) + sobelKernelX(3, 2) \* padded\_gray\_img(row + 2, col + 1) + sobelKernelX(3, 3) \* padded\_gray\_img(row + 2, col + 2);  end  end  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（垂直方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  gy(row, col) = sobelKernelY(1, 1) \* padded\_gray\_img(row, col) + sobelKernelY(1, 2) \* padded\_gray\_img(row, col + 1) + sobelKernelY(1, 3) \* padded\_gray\_img(row, col + 2) +...  sobelKernelY(2, 1) \* padded\_gray\_img(row + 1, col) + sobelKernelY(2, 2) \* padded\_gray\_img(row + 1, col + 1) + sobelKernelY(2, 3) \* padded\_gray\_img(row + 1, col + 2) +...  sobelKernelY(3, 1) \* padded\_gray\_img(row + 2, col) + sobelKernelY(3, 2) \* padded\_gray\_img(row + 2, col + 1) + sobelKernelY(3, 3) \* padded\_gray\_img(row + 2, col + 2);  end  end  % 计算边缘强度，按照公式sqrt(gx.^2 + gy.^2)  edge\_img = sqrt(gx.^2 + gy.^2);  end |  5.2 robert算子  |  | | --- | | function edge\_img = my\_roberts\_edge\_detection(gray\_img)  % 获取输入灰度图像的尺寸  [height, width] = size(gray\_img);  % 初始化用于存储水平和垂直方向卷积结果的矩阵  g1 = zeros(height, width);  g2 = zeros(height, width);  % 定义Roberts算子卷积核（水平方向和垂直方向）  robertsKernelX = [-1 0; 0 1];  robertsKernelY = [0 -1; 1 0];  % 对图像进行边界扩充，以便处理边缘像素（采用复制边界的方式扩充）  padded\_gray\_img = padarray(gray\_img, [1, 1], 'replicate');  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（水平方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  g1(row, col) = robertsKernelX(1, 1) \* padded\_gray\_img(row, col) + robertsKernelX(1, 2) \* padded\_gray\_img(row, col + 1) +...  robertsKernelX(2, 1) \* padded\_gray\_img(row + 1, col) + robertsKernelX(2, 2) \* padded\_gray\_img(row + 1, col + 1);  end  end  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（垂直方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  g2(row, col) = robertsKernelY(1, 1) \* padded\_gray\_img(row, col) + robertsKernelY(1, 2) \* padded\_gray\_img(row, col + 1) +...  robertsKernelY(2, 1) \* padded\_gray\_img(row + 1, col) + robertsKernelY(2, 2) \* padded\_gray\_img(row + 1, col + 1);  end  end  % 计算边缘强度，按照公式sqrt(g1.^2 + g2.^2)  edge\_img = sqrt(g1.^2 + g2.^2);  end |  5.3prewitt算子  |  | | --- | | function edge\_img = my\_prewitt\_edge\_detection(gray\_img)  % 获取输入灰度图像的尺寸  [height, width] = size(gray\_img);  % 初始化用于存储水平和垂直方向卷积结果的矩阵  gx = zeros(height, width);  gy = zeros(height, width);  % 定义Prewitt算子卷积核（水平方向和垂直方向）  prewittKernelX = [-1 -1 -1; 0 0 0; 1 1 1];  prewittKernelY = [-1 0 1; -1 0 1; -1 0 1];  % 对图像进行边界扩充，以便处理边缘像素（采用复制边界的方式扩充）  padded\_gray\_img = padarray(gray\_img, [1, 1], 'replicate');  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（水平方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  gx(row, col) = prewittKernelX(1, 1) \* padded\_gray\_img(row, col) + prewittKernelX(1, 2) \* padded\_gray\_img(row, col + 1) + prewittKernelX(1, 3) \* padded\_gray\_img(row, col + 2) +...  prewittKernelX(2, 1) \* padded\_gray\_img(row + 1, col) + prewittKernelX(2, 2) \* padded\_gray\_img(row + 1, col + 1) + prewittKernelX(2, 3) \* padded\_gray\_img(row + 1, col + 2) +...  prewittKernelX(3, 1) \* padded\_gray\_img(row + 2, col) + prewittKernelX(3, 2) \* padded\_gray\_img(row + 2, col + 1) + prewittKernelX(3, 3) \* padded\_gray\_img(row + 2, col + 2);  end  end  % 循环遍历图像（除去扩充的边界部分），进行卷积操作（垂直方向）  for row = 1:height  for col = 1:width  % 对应卷积核与图像像素的相乘累加操作，实现卷积  gy(row, col) = prewittKernelY(1, 1) \* padded\_gray\_img(row, col) + prewittKernelY(1, 2) \* padded\_gray\_img(row, col + 1) + prewittKernelY(1, 3) \* padded\_gray\_img(row, col + 2) +...  prewittKernelY(2, 1) \* padded\_gray\_img(row + 1, col) + prewittKernelY(2, 2) \* padded\_gray\_img(row + 1, col + 1) + prewittKernelY(2, 3) \* padded\_gray\_img(row + 1, col + 2) +...  prewittKernelY(3, 1) \* padded\_gray\_img(row + 2, col) + prewittKernelY(3, 2) \* padded\_gray\_img(row + 2, col + 1) + prewittKernelY(3, 3) \* padded\_gray\_img(row + 2, col + 2);  end  end  % 计算边缘强度，按照公式sqrt(gx.^2 + gy.^2)  edge\_img = sqrt(gx.^2 + gy.^2);  end |  5.4拉普拉斯算子 表格 20 拉普拉斯代码   |  | | --- | | function edge\_img = my\_laplacian\_edge\_detection(gray\_img)  % MY\_LAPLACIAN\_EDGE\_DETECTION 使用拉普拉斯算子进行边缘检测。  %  % 输入:  % gray\_img - 灰度输入图像。  %  % 输出:  % edge\_img - 边缘检测后的二值图像。  % 将图像转换为 double 类型以确保精度，并归一化到 [0, 1] 范围  gray\_img = im2double(gray\_img);  % 定义拉普拉斯算子卷积核（常用形式）  laplacianKernel = [0 -1 0; -1 4 -1; 0 -1 0];  % 对图像进行边界扩充，以便处理边缘像素（采用复制边界的方式扩充）  padded\_gray\_img = padarray(gray\_img, [1, 1], 'replicate');  % 使用 conv2 函数进行卷积操作，提高效率  conv\_result = conv2(padded\_gray\_img, laplacianKernel, 'valid');  % 进行二值化处理，这里使用 Otsu 方法自动选择阈值  level = graythresh(conv\_result);  edge\_img = imbinarize(conv\_result, level);  % 可选步骤：应用形态学操作去除小对象或填补空洞  % edge\_img = bwareaopen(edge\_img, 50); % 移除面积小于50个像素的对象  % edge\_img = imfill(edge\_img, 'holes'); % 填补封闭区域内的孔洞  % 确保输出图像的数据类型与输入一致  edge\_img = cast(edge\_img, class(gray\_img)); |  6.目标提取6.1目标提取 注：该要求实现写在GUI界面设计内，故无函数形式   |  | | --- | | % 检查是否已经加载了原图和背景图  if ~isfield(handles, 'imageData') || isempty(handles.imageData) || ...  ~isfield(handles, 'backgroundImage') || isempty(handles.backgroundImage)  warndlg('请先加载原图和背景图', '缺少图像');  return;  end  % 将图像转换为double类型  original\_img = im2double(handles.imageData);  background\_img = im2double(handles.backgroundImage);  % 确保背景图像是灰度图像  if size(background\_img, 3) == 3  backgroundGray = my\_rgb2gray(background\_img); % 转换为灰度图像  else  backgroundGray = background\_img; % 如果已经是灰度图像，则直接使用  end  % 将背景图像调整到与原图相同的大小  if ~isequal(size(original\_img(:,:,1)), size(backgroundGray))  backgroundGray = imresize(backgroundGray, size(original\_img(:,:,1)));  end  % 将背景灰度图像归一化到[0, 1]范围，以便作为掩码使用  backgroundMask = mat2gray(backgroundGray);  % 使用掩码从原图中提取目标区域（图像乘法）  if size(original\_img, 3) == 3 % 如果是RGB图像  targetImage = bsxfun(@times, original\_img, cast(backgroundMask, class(original\_img))); % 应用掩码到每个颜色通道并保持原始图像的数据类型  else % 如果是灰度图像  targetImage = bsxfun(@times, original\_img, cast(backgroundMask, class(original\_img))); % 应用掩码到灰度图像并保持原始图像的数据类型  end |  7.特征提取7.1 lbp特征提取  |  | | --- | | function lbp\_img = computeLBPImage(gray\_img)  % 默认参数：半径为1，邻居点数为8（3x3邻域）  radius = 1;  neighbors = 8;  % 获取图像尺寸  [rows, cols] = size(gray\_img);  % 初始化LBP图像，填充NaN以处理边界情况  lbp\_img = NaN(rows, cols);  % 遍历图像中的每个像素（除了边界）  for i = radius + 1:rows - radius  for j = radius + 1:cols - radius  centerPixel = gray\_img(i, j);  code = 0;  for n = 0:(neighbors-1)  % 计算邻居的位置  theta = 2 \* pi / neighbors \* n;  x = i + round(radius \* cos(theta));  y = j - round(radius \* sin(theta));  % 比较中心像素与邻居像素  if gray\_img(x, y) >= centerPixel  code = bitset(code, neighbors - n);  end  end  lbp\_img(i, j) = code;  end  end  % 处理边界（可选），这里简单地用0填充  lbp\_img(isnan(lbp\_img)) = 0; |  7.2 hog特征提取  |  | | --- | | function hog\_features = extractHOGFeaturesManual(img)  [h, w] = size(img);  % 确保图像为 double 类型并归一化到 [0, 1]  img = im2double(img); % 将图像转换为 double 类型  img = sqrt(img); % 伽马校正  fy = [-1 0 1]; % 定义竖直模板  fx = fy'; % 定义水平模板  Iy = imfilter(img, fy, 'replicate'); % 竖直边缘  Ix = imfilter(img, fx, 'replicate'); % 水平边缘  Ied = sqrt(Ix.^2 + Iy.^2); % 边缘强度  Iphase = Iy ./ Ix;  step = 16; % step\*step个像素作为一个单元  orient = 9; % 方向直方图的方向个数  jiao = 360 / orient; % 每个方向包含的角度数  Cell = cell(1, 1);  ii = 1;  jj = 1;  for i = 1:step:h - step + 1  ii = 1;  for j = 1:step:w - step + 1  tmpx = Ix(i:i + step - 1, j:j + step - 1);  tmped = Ied(i:i + step - 1, j:j + step - 1);  tmped = tmped / sum(sum(tmped)); % 局部边缘强度归一化  tmpphase = Iphase(i:i + step - 1, j:j + step - 1);  Hist = zeros(1, orient);  for p = 1:step  for q = 1:step  if isnan(tmpphase(p, q)) % 0/0会得到nan，如果像素是nan，重设为0  tmpphase(p, q) = 0;  end  ang = atan(tmpphase(p, q));  ang = mod(ang \* 180 / pi, 360); % 全部变正，-90变270  if tmpx(p, q) < 0  if ang < 90 % 如果是第一象限  ang = ang + 180; % 移到第三象限  end  if ang > 270 % 如果是第四象限  ang = ang - 180; % 移到第二象限  end  end  ang = ang + 0.0000001;  Hist(ceil(ang / jiao)) = Hist(ceil(ang / jiao)) + tmped(p, q); % 使用边缘强度加权  end  end  Hist = Hist / sum(Hist); % 方向直方图归一化  Cell{ii, jj} = Hist;  ii = ii + 1;  end  jj = jj + 1;  end  [m, n] = size(Cell);  feature = cell(1, (m - 1) \* (n - 1));  for i = 1:m - 1  for j = 1:n - 1  f = [];  f = [f Cell{i, j}(:)' Cell{i, j + 1}(:)' Cell{i + 1, j}(:)' Cell{i + 1, j + 1}(:)'];  feature{(i - 1) \* (n - 1) + j} = f;  end  end  l = length(feature);  hog\_features = [];  for i = 1:l  hog\_features = [hog\_features; feature{i}(:)'];  end |  8.模型预测8.1 模型训练与应用  |  | | --- | | import tensorflow as tf  from tensorflow.keras import layers, models  from tensorflow.keras.applications.resnet50 import ResNet50  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib  matplotlib.use('TkAgg')  import matplotlib.pyplot as plt  import numpy as np  import glob  import os  from pathlib import Path  # 设置随机种子以保证结果可复现  np.random.seed(2024)  # 获取所有图片路径  imgs\_path = glob.glob(r'D:\CUB\_200\_2011\images\\*\\*.jpg')  # 提取标签名称（使用更稳健的路径处理）  all\_labels\_name = [Path(imgs\_p).parent.name for imgs\_p in imgs\_path]  # 获取唯一的标签名，并创建标签到索引的映射  label\_names = np.unique(all\_labels\_name)  label\_to\_index = dict((name, i) for i, name in enumerate(label\_names))  index\_to\_label = dict((v, k) for k, v in label\_to\_index.items())  # 将标签名称转换为索引  all\_labels = [label\_to\_index.get(name) for name in all\_labels\_name]  # 打乱数据集  random\_index = np.random.permutation(len(imgs\_path))  imgs\_path = np.array(imgs\_path)[random\_index]  all\_labels = np.array(all\_labels)[random\_index]  # 划分训练集和测试集  train\_count = int(len(imgs\_path) \* 0.8)  train\_path = imgs\_path[:train\_count]  train\_labels = all\_labels[:train\_count]  test\_path = imgs\_path[train\_count:]  test\_labels = all\_labels[train\_count:]  # 构建数据集  train\_ds = tf.data.Dataset.from\_tensor\_slices((train\_path, train\_labels))  test\_ds = tf.data.Dataset.from\_tensor\_slices((test\_path, test\_labels))  def load\_img(path, label):  image = tf.io.read\_file(path)  image = tf.image.decode\_jpeg(image, channels=3)  image = tf.image.resize(image, [256, 256])  image = tf.cast(image, tf.float32)  image = image / 255.0  return image, label  AUTOTUNE = tf.data.experimental.AUTOTUNE  BATCH\_SIZE = 16 # 增大批次大小  # 加载和预处理图像  train\_ds = train\_ds.map(load\_img, num\_parallel\_calls=AUTOTUNE).cache() # 添加缓存  test\_ds = test\_ds.map(load\_img, num\_parallel\_calls=AUTOTUNE).cache()  # 配置数据集以提高性能  train\_ds = train\_ds.repeat().shuffle(300).batch(BATCH\_SIZE).prefetch(buffer\_size=AUTOTUNE)  test\_ds = test\_ds.batch(BATCH\_SIZE).prefetch(buffer\_size=AUTOTUNE)  # 加载预训练的ResNet50模型，不包括顶层的全连接层  base\_model = ResNet50(include\_top=False, weights='imagenet', input\_shape=(256, 256, 3))  # 冻结基础模型的层，不让它们在训练过程中更新  base\_model.trainable = False  # 构建新的模型，在ResNet50的基础上添加自定义的顶层  model = models.Sequential([  base\_model,  layers.GlobalAveragePooling2D(),  layers.Dense(512, activation='relu'), # 精简了全连接层神经元数量  layers.BatchNormalization(),  layers.Dense(200, activation='softmax') # 假设有200个类别  ])  # 编译模型，调整了学习率  model.compile(optimizer=tf.keras.optimizers.Adam(0.001),  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # 打印模型结构  model.summary()  # 训练模型  train\_steps\_per\_epoch = len(train\_path) // BATCH\_SIZE  test\_steps\_per\_epoch = len(test\_path) // BATCH\_SIZE  history = model.fit(  train\_ds,  epochs=15,  steps\_per\_epoch=train\_steps\_per\_epoch,  validation\_data=test\_ds,  validation\_steps=test\_steps\_per\_epoch  )  # 绘制训练与验证的准确率和损失值  plt.figure(figsize=(12, 4))  plt.subplot(1, 2, 1)  plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')  plt.plot(history.epoch, history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Accuracy over Epochs')  plt.legend()  plt.subplot(1, 2, 2)  plt.plot(history.epoch, history.history['loss'], label='Train Loss')  plt.plot(history.epoch, history.history['val\_loss'], label='Validation Loss')  plt.title('Loss over Epochs')  plt.legend()  plt.show()  # 保存模型（修改后的保存方式）  filepath = r'C:\Users\22756\Desktop\Digital-Image-Processing\model\_birds\_resnet.h5'  model.save(filepath)  print("模型训练完成并已保存") | | % 定义一个结构体来管理相关数据和操作  classdef BirdClassifier  properties  dataMap % 用于存储类别编号和对应鸟类名称的映射关系  net % 加载的深度学习网络模型  end  methods  % 构造函数，用于初始化数据映射和加载网络模型  function obj = BirdClassifier()  % 初始化类别编号和鸟类名称的映射关系  data\_str = '0: ''Acadian\_Flycatcher'', 1: ''American\_Crow'', 2: ''American\_Goldfinch'', 3: ''American\_Pipit'', 4: ''American\_Redstart'', 5: ''American\_Three\_toed\_Woodpecker'', 6: ''Anna\_Hummingbird'', 7: ''Artic\_Tern'', 8: ''Baird\_Sparrow'', 9: ''Baltimore\_Oriole'', 10: ''Bank\_Swallow'', 11: ''Barn\_Swallow'', 12: ''Bay\_breasted\_Warbler'', 13: ''Belted\_Kingfisher'', 14: ''Bewick\_Wren'', 15: ''Black\_Tern'', 16: ''Black\_and\_white\_Warbler'', 17: ''Black\_billed\_Cuckoo'', 18: ''Black\_capped\_Vireo'', 19: ''Black\_footed\_Albatross'', 20: ''Black\_throated\_Blue\_Warbler'', 21: ''Black\_throated\_Sparrow'', 22: ''Blue\_Grosbeak'', 23: ''Blue\_Jay'', 24: ''Blue\_headed\_Vireo'', 25: ''Blue\_winged\_Warbler'', 26: ''Boat\_tailed\_Grackle'', 27: ''Bobolink'', 28: ''Bohemian\_Waxwing'', 29: ''Brandt\_Cormorant'', 30: ''Brewer\_Blackbird'', 31: ''Brewer\_Sparrow'', 32: ''Bronzed\_Cowbird'', 33: ''Brown\_Creeper'', 34: ''Brown\_Pelican'', 35: ''Brown\_Thrasher'', 36: ''Cactus\_Wren'', 37: ''California\_Gull'', 38: ''Canada\_Warbler'', 39: ''Cape\_Glossy\_Starling'', 40: ''Cape\_May\_Warbler'', 41: ''Cardinal'', 42: ''Carolina\_Wren'', 43: ''Caspian\_Tern'', 44: ''Cedar\_Waxwing'', 45: ''Cerulean\_Warbler'', 46: ''Chestnut\_sided\_Warbler'', 47: ''Chipping\_Sparrow'', 48: ''Chuck\_will\_Widow'', 49: ''Clark\_Nutcracker'', 50: ''Clay\_colored\_Sparrow'', 51: ''Cliff\_Swallow'', 52: ''Common\_Raven'', 53: ''Common\_Tern'', 54: ''Common\_Yellowthroat'', 55: ''Crested\_Auklet'', 56: ''Dark\_eyed\_Junco'', 57: ''Downy\_Woodpecker'', 58: ''Eared\_Grebe'', 59: ''Eastern\_Towhee'', 60: ''Elegant\_Tern'', 61: ''European\_Goldfinch'', 62: ''Evening\_Grosbeak'', 63: ''Field\_Sparrow'', 64: ''Fish\_Crow'', 65: ''Florida\_Jay'', 66: ''Forsters\_Tern'', 67: ''Fox\_Sparrow'', 68: ''Frigatebird'', 69: ''Gadwall'', 70: ''Geococcyx'', 71: ''Glaucous\_winged\_Gull'', 72: ''Golden\_winged\_Warbler'', 73: ''Grasshopper\_Sparrow'', 74: ''Gray\_Catbird'', 75: ''Gray\_Kingbird'', 76: ''Gray\_crowned\_Rosy\_Finch'', 77: ''Great\_Crested\_Flycatcher'', 78: ''Great\_Grey\_Shrike'', 79: ''Green\_Jay'', 80: ''Green\_Kingfisher'', 81: ''Green\_Violetear'', 82: ''Green\_tailed\_Towhee'', 83: ''Groove\_billed\_Ani'', 84: ''Harris\_Sparrow'', 85: ''Heermann\_Gull'', 86: ''Henslow\_Sparrow'', 87: ''Herring\_Gull'', 88: ''Hooded\_Merganser'', 89: ''Hooded\_Oriole'', 90: ''Hooded\_Warbler'', 91: ''Horned\_Grebe'', 92: ''Horned\_Lark'', 93: ''Horned\_Puffin'', 94: ''House\_Sparrow'', 95: ''House\_Wren'', 96: ''Indigo\_Bunting'', 97: ''Ivory\_Gull'', 98: ''Kentucky\_Warbler'', 99: ''Laysan\_Albatross'', 100: ''Lazuli\_Bunting'', 101: ''Le\_Conte\_Sparrow'', 102: ''Least\_Auklet'', 103: ''Least\_Flycatcher'', 104: ''Least\_Tern'', 105: ''Lincoln\_Sparrow'', 106: ''Loggerhead\_Shrike'', 107: ''Long\_tailed\_Jaeger'', 108: ''Louisiana\_Waterthrush'', 109: ''Magnolia\_Warbler'', 110: ''Mallard'', 111: ''Mangrove\_Cuckoo'', 112: ''Marsh\_Wren'', 113: ''Mockingbird'', 114: ''Mourning\_Warbler'', 115: ''Myrtle\_Warbler'', 116: ''Nashville\_Warbler'', 117: ''Nelson\_Sharp\_tailed\_Sparrow'', 118: ''Nighthawk'', 119: ''Northern\_Flicker'', 120: ''Northern\_Fulmar'', 121: ''Northern\_Waterthrush'', 122: ''Olive\_sided\_Flycatcher'', 123: ''Orange\_crowned\_Warbler'', 124: ''Orchard\_Oriole'', 125: ''Ovenbird'', 126: ''Pacific\_Loon'', 127: ''Painted\_Bunting'', 128: ''Palm\_Warbler'', 129: ''Parakeet\_Auklet'', 130: ''Pelagic\_Cormorant'', 131: ''Philadelphia\_Vireo'', 132: ''Pied\_Kingfisher'', 133: ''Pied\_billed\_Grebe'', 134: ''Pigeon\_Guillemot'', 135: ''Pileated\_Woodpecker'', 136: ''Pine\_Grosbeak'', 137: ''Pine\_Warbler'', 138: ''Pomarine\_Jaeger'', 139: ''Prairie\_Warbler'', 140: ''Prothonotary\_Warbler'', 141: ''Purple\_Finch'', 142: ''Red\_bellied\_Woodpecker'', 143: ''Red\_breasted\_Merganser'', 144: ''Red\_cockaded\_Woodpecker'', 145: ''Red\_eyed\_Vireo'', 146: ''Red\_faced\_Cormorant'', 147: ''Red\_headed\_Woodpecker'', 148: ''Red\_legged\_Kittiwake'', 149: ''Red\_winged\_Blackbird'', 150: ''Rhinoceros\_Auklet'', 151: ''Ring\_billed\_Gull'', 152: ''Ringed\_Kingfisher'', 153: ''Rock\_Wren'', 154: ''Rose\_breasted\_Grosbeak'', 155: ''Ruby\_throated\_Hummingbird'', 156: ''Rufous\_Hummingbird'', 157: ''Rusty\_Blackbird'', 158: ''Sage\_Thrasher'', 159: ''Savannah\_Sparrow'', 160: ''Sayornis'', 161: ''Scarlet\_Tanager'', 162: ''Scissor\_tailed\_Flycatcher'', 163: ''Scott\_Oriole'', 164: ''Seaside\_Sparrow'', 165: ''Shiny\_Cowbird'', 166: ''Slaty\_backed\_Gull'', 167: ''Song\_Sparrow'', 168: ''Sooty\_Albatross'', 169: ''Spotted\_Catbird'', 170: ''Summer\_Tanager'', 171: ''Swainson\_Warbler'', 172: ''Tennessee\_Warbler'', 173: ''Tree\_Sparrow'', 174: ''Tree\_Swallow'', 175: ''Tropical\_Kingbird'', 176: ''Vermilion\_Flycatcher'', 177: ''Vesper\_Sparrow'', 178: ''Warbling\_Vireo'', 179: ''Western\_Grebe'', 180: ''Western\_Gull'', 181: ''Western\_Meadowlark'', 182: ''Western\_Wood\_Pewee'', 183: ''Whip\_poor\_Will'', 184: ''White\_Pelican'', 185: ''White\_breasted\_Kingfisher'', 186: ''White\_breasted\_Nuthatch'', 187: ''White\_crowned\_Sparrow'', 188: ''White\_eyed\_Vireo'', 189: ''White\_necked\_Raven'', 190: ''White\_throated\_Sparrow'', 191: ''Wilson\_Warbler'', 192: ''Winter\_Wren'', 193: ''Worm\_eating\_Warbler'', 194: ''Yellow\_Warbler'', 195: ''Yellow\_bellied\_Flycatcher'', 196: ''Yellow\_billed\_Cuckoo'', 197: ''Yellow\_breasted\_Chat'', 198: ''Yellow\_headed\_Blackbird'', 199: ''Yellow\_throated\_Vireo''';  pattern = '(\d+): ''([^'']+)''';  matches = regexp(data\_str, pattern, 'tokens');  obj.dataMap = containers.Map();  for i = 1:length(matches)  key = matches{i}{1}; % 转换键为数字  value = matches{i}{2}; % 获取值  obj.dataMap(key) = value; % 将键值对插入 Map  end  % 加载深度学习网络模型  obj.net = importKerasNetwork("C:\Users\22756\Desktop\Digital-Image-Processing\model\_birds\_resnet.h5");  end  % 预测鸟类类别函数，接收图像作为输入，返回预测的鸟类类别名称  function type = predictBirdType(obj, img)  targetSize = [256, 256];  img = imresize(img, targetSize);  output = predict(obj.net,img);  predictedClassIndex = find(max(output) == output, 1);  predictedstr = char(num2str(predictedClassIndex));  type = obj.dataMap(predictedstr);  disp('预测的类别索引：');  disp(predictedClassIndex);  disp('预测的类别：');  disp(type);  end  end  end | |  9.GUI界面设计9.1 GUI界面设计代码  |  | | --- | | function varargout = birdimageprocessing(varargin)  % BIRDIMAGEPROCESSING MATLAB code for birdimageprocessing.fig  % BIRDIMAGEPROCESSING, by itself, creates a new BIRDIMAGEPROCESSING or raises the existing  % singleton\*.  %  % H = BIRDIMAGEPROCESSING returns the handle to a new BIRDIMAGEPROCESSING or the handle to  % the existing singleton\*.  %  % BIRDIMAGEPROCESSING('CALLBACK',hObject,eventData,handles,...) calls the local  % function named CALLBACK in BIRDIMAGEPROCESSING.M with the given input arguments.  %  % BIRDIMAGEPROCESSING('Property','Value',...) creates a new BIRDIMAGEPROCESSING or raises the  % existing singleton\*. Starting from the left, property value pairs are  % applied to the GUI before birdimageprocessing\_OpeningFcn gets called. An  % unrecognized property name or invalid value makes property application  % stop. All inputs are passed to birdimageprocessing\_OpeningFcn via varargin.  %  % \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one  % instance to run (singleton)".  %  % See also: GUIDE, GUIDATA, GUIHANDLES  % Edit the above text to modify the response to help birdimageprocessing  % Last Modified by GUIDE v2.5 24-Dec-2024 15:20:02  % Begin initialization code - DO NOT EDIT  gui\_Singleton = 1;  gui\_State = struct('gui\_Name', mfilename, ...  'gui\_Singleton', gui\_Singleton, ...  'gui\_OpeningFcn', @birdimageprocessing\_OpeningFcn, ...  'gui\_OutputFcn', @birdimageprocessing\_OutputFcn, ...  'gui\_LayoutFcn', [] , ...  'gui\_Callback', []);  if nargin && ischar(varargin{1})  gui\_State.gui\_Callback = str2func(varargin{1});  end  if nargout  [varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});  else  gui\_mainfcn(gui\_State, varargin{:});  end  % End initialization code - DO NOT EDIT  % --- Executes just before birdimageprocessing is made visible.  function birdimageprocessing\_OpeningFcn(hObject, eventdata, handles, varargin)  % This function has no output args, see OutputFcn.  % hObject handle to figure  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % varargin command line arguments to birdimageprocessing (see VARARGIN)  % Choose default command line output for birdimageprocessing  handles.output = hObject;  % Update handles structure  guidata(hObject, handles);  % UIWAIT makes birdimageprocessing wait for user response (see UIRESUME)  % uiwait(handles.figure1);  % --- Outputs from this function are returned to the command line.  function varargout = birdimageprocessing\_OutputFcn(hObject, eventdata, handles)  % varargout cell array for returning output args (see VARARGOUT);  % hObject handle to figure  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % Get default command line output from handles structure  varargout{1} = handles.output;  % --- Executes on button press in pushbutton1.  % --- Executes on button press in pushbutton1.  % --- Executes on button press in pushbutton1.  function pushbutton1\_Callback(hObject, eventdata, handles)  % 打开文件选择对话框让用户选择一个图像文件  [filename, pathname] = uigetfile({'\*.jpg; \*.jpeg; \*.png; \*.bmp','Image Files (\*.jpg, \*.jpeg, \*.png, \*.bmp)'; ...  '\*.\*', 'All Files (\*.\*)'}, 'Select an Image');  if isequal(filename,0) || isequal(pathname,0)  % 如果用户取消了文件选择，就退出函数  return;  end  % 构建完整的文件路径  fullpath = fullfile(pathname, filename);  % 读取图像文件  img = imread(fullpath);  % 显示图像  % 假设你有一个 axes 或 image 控件用于显示图像，其标签为 'axes1'  axes(handles.axes1); % 设置当前坐标轴为要显示图像的坐标轴  imshow(img); % 在坐标轴上显示图像  drawnow; % 更新图形窗口  % 更新编辑框以显示所选图像的文件名  set(handles.edit1, 'String', filename);  % 将图像数据存储在 handles 结构体中以便其他回调函数访问  handles.imageData = img; % 添加这行来保存图像数据  handles.imagePath = fullpath;  guidata(hObject, handles); % 确保更新 handles 结构体  function edit1\_Callback(hObject, eventdata, handles)  % hObject handle to edit1 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % Hints: get(hObject,'String') returns contents of edit1 as text  % str2double(get(hObject,'String')) returns contents of edit1 as a double  % --- Executes during object creation, after setting all properties.  function edit1\_CreateFcn(hObject, eventdata, handles)  % hObject handle to edit1 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles empty - handles not created until after all CreateFcns called  % Hint: edit controls usually have a white background on Windows.  % See ISPC and COMPUTER.  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  set(hObject,'BackgroundColor','white');  end  % --- Executes on button press in pushbuttonHistogram.  function pushbuttonHistogram\_Callback(hObject, eventdata, handles)  % 确保有一个打开的图像  if ~isfield(handles, 'imageData') || isempty(handles.imageData)  warndlg('请先选择一张图像', '无图像');  return;  end  % 获取图像数据  img = handles.imageData;  % 如果图像是彩色的，转换为灰度图像  if size(img, 3) > 1  img = my\_rgb2gray(img);  end  % 在axes2上显示直方图  axes(handles.axes2);  my\_imhist(img);  title('灰度直方图');  % --- Executes on button press in pushbuttonEqualize.  function pushbuttonEqualize\_Callback(hObject, eventdata, handles)  % 确保有一个打开的图像  if ~isfield(handles, 'imageData') || isempty(handles.imageData)  warndlg('请先选择一张图像', '无图像');  return;  end  % 获取原始图像数据  original\_img = handles.imageData;  % 如果图像是彩色的，转换为灰度图像副本进行处理  if size(original\_img, 3) > 1  gray\_img = my\_rgb2gray(original\_img);  else  gray\_img = original\_img; % 如果已经是灰度图像，则直接使用  end  % 执行直方图均衡化  eq\_gray\_img = histeq(gray\_img);  % 显示均衡化后的直方图在axes3中  axes(handles.axes3);  my\_imhist(eq\_gray\_img);  title('均衡化后的灰度直方图');  % 更新handles结构体中的图像数据，以便后续操作使用  handles.equalizedImageData = eq\_gray\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbuttonMatch.  function pushbuttonMatch\_Callback(hObject, eventdata, handles)  % 确保有一个均衡化后的图像  if ~isfield(handles, 'equalizedImageData') || isempty(handles.equalizedImageData)  warndlg('请先执行直方图均衡化', '无均衡化图像');  return;  end  % 使用均衡化后的图像作为源图像  src\_img = handles.equalizedImageData;  % 如果用户已经选择了参考图像，则直接使用它  if isfield(handles, 'referenceImageData') && ~isempty(handles.referenceImageData)  ref\_img = handles.referenceImageData;  else  % 否则，打开参考图像  [ref\_filename, ref\_pathname] = uigetfile({'\*.jpg; \*.jpeg; \*.png; \*.bmp','Image Files (\*.jpg, \*.jpeg, \*.png, \*.bmp)'; ...  '\*.\*', 'All Files (\*.\*)'}, 'Select Reference Image');  if isequal(ref\_filename,0) || isequal(ref\_pathname,0)  % 如果用户取消了文件选择，就退出函数  return;  end  ref\_fullpath = fullfile(ref\_pathname, ref\_filename);  ref\_img = imread(ref\_fullpath);  % 如果参考图像是彩色的，转换为灰度图像  if size(ref\_img, 3) > 1  ref\_img = my\_rgb2gray(ref\_img);  end  % 保存参考图像到handles结构中，以便之后可以直接使用  handles.referenceImageData = ref\_img;  guidata(hObject, handles);  end  % 执行直方图匹配  matched\_img = my\_histmatch(src\_img, ref\_img);  % 显示匹配后的直方图在axes3中  axes(handles.axes3);  my\_imhist(matched\_img);  title('匹配后的灰度直方图');  % 更新handles结构体中的图像数据  handles.matchedImageData = matched\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton5.  function pushbutton5\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton5 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 确保有一个打开的图像  if ~isfield(handles, 'imageData') || isempty(handles.imageData)  warndlg('请先选择一张图像', '无图像');  return;  end  % 获取原始图像数据  img = handles.imageData;  % 如果图像是彩色的，转换为灰度图像副本进行处理  if size(img, 3) > 1  gray\_img = my\_rgb2gray(img);  else  gray\_img = img; % 如果已经是灰度图像，则直接使用  end  % 在axes4上显示灰度图像  axes(handles.axes4);  imshow(gray\_img);  title('灰度图像');  % 更新handles结构体中的图像数据，以便后续操作使用  handles.grayImageData = gray\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton6.  function pushbutton6\_Callback(hObject, eventdata, handles)  % 确保有一个灰度化的图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = handles.grayImageData; % 不需要显式转换为double类型，由my\_linear\_transform处理  % 定义线性变换参数（这里假设简单增强对比度）  alpha = 1.2; % 放大系数  beta = -20; % 偏移量  % 应用线性变换  linear\_transformed\_img = my\_linear\_transform(gray\_img, alpha, beta);  % 在axes5上显示线性变换后的图像  axes(handles.axes5);  imshow(linear\_transformed\_img);  title('线性变换后的图像');  % 更新handles结构体中的图像数据  handles.linearTransformedImageData = linear\_transformed\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton7.  function pushbutton7\_Callback(hObject, eventdata, handles)  % 确保有一个灰度化的图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = handles.grayImageData;  % 应用对数变换  log\_transformed\_img = my\_log\_transform(gray\_img);  % 在axes5上显示对数变换后的图像  axes(handles.axes5);  imshow(log\_transformed\_img);  title('对数变换后的图像');  % 更新handles结构体中的图像数据  handles.logTransformedImageData = log\_transformed\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton8.  function pushbutton8\_Callback(hObject, eventdata, handles)  % 确保有一个灰度化的图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = handles.grayImageData;  % 定义指数变换参数  gamma = 0.4; % 指数变换的伽马值  % 应用指数变换  exponential\_transformed\_img = my\_exponential\_transform(gray\_img, gamma);  % 在axes5上显示指数变换后的图像  axes(handles.axes5);  imshow(exponential\_transformed\_img);  title('指数变换后的图像');  % 更新handles结构体中的图像数据  handles.exponentialTransformedImageData = exponential\_transformed\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton9.  function pushbutton9\_Callback(hObject, eventdata, handles)  % 确保有一个灰度化的图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = double(handles.grayImageData); % 转换为double类型以便于计算  % 提示用户输入缩放比例  prompt = {'请输入宽度缩放因子:', '请输入高度缩放因子:'};  dlg\_title = '输入缩放参数';  num\_lines = 1;  def\_input = {'2', '1.5'};  answer = inputdlg(prompt, dlg\_title, num\_lines, def\_input);  if isempty(answer) || any(cellfun(@isempty, answer))  return; % 用户取消了对话框或未填写所有字段  end  scale\_factor\_x = str2double(answer{1});  scale\_factor\_y = str2double(answer{2});  % 输入验证  if isnan(scale\_factor\_x) || isnan(scale\_factor\_y) || ...  scale\_factor\_x <= 0 || scale\_factor\_y <= 0  warndlg('请输入有效的正数作为缩放因子', '无效输入');  return;  end  % 获取输入图像的尺寸  [m, n] = size(gray\_img);  % 计算放大后的图像尺寸，采用四舍五入  new\_m = round(m \* scale\_factor\_y);  new\_n = round(n \* scale\_factor\_x);  % 创建一个新的空白图像  scaled\_img = zeros(new\_m, new\_n);  % 计算每个新像素的值（双线性插值）  for i = 1:new\_m  for j = 1:new\_n  % 计算在原图像中的坐标  x = (j - 0.5) / scale\_factor\_x - 0.5;  y = (i - 0.5) / scale\_factor\_y - 0.5;  % 向上取整和向下取整  x1 = floor(x); % 向下  x2 = ceil(x); % 向上  y1 = floor(y);  y2 = ceil(y);  % 确保不超出图像边界  x1 = max(1, min(x1, n-1));  x2 = max(1, min(x2, n-1));  y1 = max(1, min(y1, m-1));  y2 = max(1, min(y2, m-1));  % 双线性插值计算  if x2 == x1 && y2 == y1  value = gray\_img(y1+1, x1+1);  else  Q11 = gray\_img(y1+1, x1+1);  Q21 = gray\_img(y2+1, x1+1);  Q12 = gray\_img(y1+1, x2+1);  Q22 = gray\_img(y2+1, x2+1);  value = (Q11 \* (x2-x) \* (y2-y) + ...  Q21 \* (x-x1) \* (y2-y) + ...  Q12 \* (x2-x) \* (y-y1) + ...  Q22 \* (x-x1) \* (y-y1));  end  scaled\_img(i, j) = value;  end  end  % 将图像转换回uint8类型  scaled\_img = uint8(scaled\_img);  % 在axes5上显示缩放后的图像  axes(handles.axes5);  cla; % 清除坐标轴以确保新图像正确显示  imshow(scaled\_img);  title(['缩放因子: ', num2str(scale\_factor\_x), 'x', num2str(scale\_factor\_y)]);  % 更新handles结构体中的图像数据  handles.scaledImageData = scaled\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton10.  function pushbutton10\_Callback(hObject, eventdata, handles)  % 确保有一个灰度化的图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = handles.grayImageData;  % 提示用户输入旋转角度  angle = inputdlg('请输入旋转角度（顺时针方向，单位：度）:', '输入旋转角度', 1, {'0'});  if isempty(angle)  return; % 用户取消了对话框  end  angle\_degrees = str2double(angle{1});  if isnan(angle\_degrees)  warndlg('请输入有效的数字作为旋转角度', '无效输入');  return;  end  % 应用旋转变换  rotated\_img = my\_imrotate(gray\_img, -angle\_degrees, 'method', 'bilinear', 'crop', true);  % 在axes5上显示旋转变换后的图像  axes(handles.axes5);  imshow(rotated\_img);  title(['旋转角度: ', num2str(angle\_degrees), '度']);  % 更新handles结构体中的图像数据  handles.rotatedImageData = rotated\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton11.  function pushbutton11\_Callback(hObject, eventdata, handles)  % 确保有一个打开的灰度图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = im2double(handles.grayImageData);  % 使用列表对话框让用户选择噪声类型  options = {'椒盐噪声', '高斯噪声', '泊松噪声'};  noiseType = listdlg('PromptString', '请选择噪声类型：', ...  'ListString', options, ...  'Name', '噪声类型选择');  if isempty(noiseType)  return; % 用户取消了对话框  end  % 定义每个噪声类型的参数和默认值  switch options{noiseType}  case '椒盐噪声'  prompt = {'请输入密度 (0-1)：'};  dlgtitle = '椒盐噪声参数';  definput = {'0.05'};  rangeHint = '密度范围为0到1之间';  case '高斯噪声'  prompt = {'请输入均值：', '请输入方差 (0-1)：'};  dlgtitle = '高斯噪声参数';  definput = {'0', '0.01'};  rangeHint = '均值可以是任意实数，方差范围为0到1之间';  case '泊松噪声'  % 泊松噪声没有额外参数  noisy\_img = imnoise(gray\_img, 'poisson');  otherwise  error('未知的噪声类型');  end  % 如果不是泊松噪声，则获取用户输入的参数  if ~strcmp(options{noiseType}, '泊松噪声')  userInputs = inputdlg(prompt, dlgtitle, 1, definput);  if isempty(userInputs)  return; % 用户取消了对话框  end  % 将输入转换为数值并应用噪声  params = cellfun(@str2double, userInputs, 'UniformOutput', false);  if any(cellfun(@isnan, params))  warndlg('请输入有效的数字参数', '无效输入');  return;  end  try  switch options{noiseType}  case '椒盐噪声'  density = params{1};  if density < 0 || density > 1  warndlg('密度应在0到1之间', '参数超出范围');  return;  end  noisy\_img = imnoise(gray\_img, 'salt & pepper', density);  case '高斯噪声'  mean\_val = params{1};  variance = params{2};  if variance < 0 || variance > 1  warndlg('方差应在0到1之间', '参数超出范围');  return;  end  noisy\_img = imnoise(gray\_img, 'gaussian', mean\_val, variance);  end  catch ME  warndlg(['发生错误: ' ME.message], '添加噪声失败');  return;  end  end  % 显示加噪后的图像  axes(handles.axes5);  imshow(noisy\_img);  title(['加噪后的图像 (' options{noiseType} ')']);  % 更新handles结构体中的图像数据  handles.noisyImageData = noisy\_img;  guidata(hObject, handles);  % 提示参数范围  msgbox(rangeHint, '参数范围提示');  % --- Executes on button press in pushbutton12.  function pushbutton12\_Callback(hObject, eventdata, handles)  % 确保有一个加噪后的图像  if ~isfield(handles, 'noisyImageData') || isempty(handles.noisyImageData)  warndlg('请先添加噪声', '无噪声图像');  return;  end  % 获取加噪后的图像数据  noisy\_img = handles.noisyImageData;  % 使用列表对话框让用户选择滤波器类型  options = {'均值滤波', '高斯滤波', '中值滤波', '双边滤波', '自定义滤波'};  filterType = listdlg('PromptString', '请选择空域滤波器类型：',...  'ListString', options,...  'Name', '空域滤波器类型选择');  if isempty(filterType)  return; % 用户取消了对话框  end  filtered\_img = [];  % 应用所选的空域滤波器  switch options{filterType}  case '均值滤波'  filtered\_img = my\_mean\_filter(noisy\_img, [3 3]);  case '高斯滤波'  filtered\_img = my\_gaussian\_filter(noisy\_img, [3 3], 0.5);  case '中值滤波'  filtered\_img = my\_median\_filter(noisy\_img);  case '双边滤波'  filtered\_img = my\_bilateral\_filter(noisy\_img, 1, 0.1); % 示例参数，可调整  case '自定义滤波'  prompt = {'请输入滤波器尺寸:', '请输入标准差:'};  dlg\_title = '自定义高斯滤波器参数';  num\_lines = 1;  def\_input = {'5', '1'};  answer = inputdlg(prompt, dlg\_title, num\_lines, def\_input);  if isempty(answer) || any(cellfun(@isempty, answer))  return; % 用户取消了对话框或未填写所有字段  end  size\_val = str2double(answer{1});  sigma\_val = str2double(answer{2});  if isnan(size\_val) || isnan(sigma\_val) || size\_val <= 0 || sigma\_val <= 0  warndlg('请输入有效的正数作为滤波器尺寸和标准差', '无效输入');  return;  end  filtered\_img = my\_gaussian\_filter(noisy\_img, [size\_val size\_val], sigma\_val);  otherwise  error('未知的滤波器类型');  end  % 显示滤波后的图像  axes(handles.axes6);  imshow(filtered\_img);  title('空域滤波后的图像');  % 更新handles结构体中的图像数据  handles.filteredSpatialData = filtered\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton13.  function pushbutton13\_Callback(hObject, eventdata, handles)  % 确保有一个加噪后的图像  if ~isfield(handles, 'noisyImageData') || isempty(handles.noisyImageData)  warndlg('请先添加噪声', '无噪声图像');  return;  end  % 获取加噪后的图像数据  noisy\_img = im2double(handles.noisyImageData);  % 计算图像的二维快速傅里叶变换  F = fftshift(fft2(noisy\_img));  % 获取频谱尺寸  [M, N] = size(F);  % 使用列表对话框让用户选择频域滤波器类型  options = {'理想低通滤波', '巴特沃斯低通滤波', '高斯低通滤波'};  filterType = listdlg('PromptString', '请选择频域滤波器类型：',...  'ListString', options,...  'Name', '频域滤波器类型选择');  if isempty(filterType)  return; % 用户取消了对话框  end  % 设置截止频率（这里可以考虑后续做成可输入的形式来提高灵活性）  cutoffFreq = 30;  % 滤波器阶数（针对巴特沃斯低通滤波，同样可考虑做成可调整形式）  n = 2;  % 根据用户选择调用相应的滤波函数创建滤波器传递函数H  switch options{filterType}  case '理想低通滤波'  H = idealLowpassFilter(M, N, cutoffFreq);  case '巴特沃斯低通滤波'  H = butterworthLowpassFilter(M, N, cutoffFreq, n);  case '高斯低通滤波'  H = gaussianLowpassFilter(M, N, cutoffFreq);  otherwise  error('未知的频域滤波器类型');  end  % 应用滤波器  G = F.\* H;  % 逆傅里叶变换回到空间域  filtered\_img = real(ifft2(ifftshift(G)));  % 显示滤波后的图像  axes(handles.axes6);  imshow(filtered\_img, []);  title('频域滤波后的图像');  % 更新handles结构体中的图像数据  handles.filteredFrequencyData = filtered\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton14.  function pushbutton14\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton14 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 检查是否存在灰度图像数据，若不存在则弹出提示并返回  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = im2double(handles.grayImageData);  % 调用自定义的Roberts算子边缘检测函数进行边缘检测  edge\_img = my\_roberts\_edge\_detection(gray\_img);  % 显示边缘检测后的图像  axes(handles.axes6);  imshow(edge\_img, []);  title('Roberts算子边缘检测后的图像');  % 更新handles结构体中的图像数据  handles.edgeDetectedImageData = edge\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton15.  function pushbutton15\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton15 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 确保有一个打开的灰度图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = im2double(handles.grayImageData);  % 调用自定义的Prewitt算子边缘检测函数进行边缘检测  edge\_img = my\_prewitt\_edge\_detection(gray\_img);  % 显示边缘检测后的图像  axes(handles.axes6);  imshow(edge\_img);  title('Prewitt算子边缘检测后的图像');  % 更新handles结构体中的图像数据  handles.edgeDetectedImageData = edge\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton16.  function pushbutton16\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton16 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 确保有一个打开的灰度图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = im2double(handles.grayImageData);  % 调用自定义的Sobel算子边缘检测函数进行边缘检测  edge\_img = my\_sobel\_edge\_detection(gray\_img);  % 显示边缘检测后的图像  axes(handles.axes6);  imshow(edge\_img);  title('Sobel算子边缘检测后的图像');  % 更新handles结构体中的图像数据  handles.edgeDetectedImageData = edge\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton17.  function pushbutton17\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton17 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 确保有一个打开的灰度图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  % 获取灰度图像数据  gray\_img = im2double(handles.grayImageData);  % 调用自定义的拉普拉斯算子边缘检测函数进行边缘检测  edge\_img = my\_laplacian\_edge\_detection(gray\_img);  % 显示边缘检测后的图像  axes(handles.axes6);  imshow(edge\_img);  title('拉普拉斯算子边缘检测后的图像');  % 更新handles结构体中的图像数据  handles.edgeDetectedImageData = edge\_img;  guidata(hObject, handles);  % --- Executes on button press in pushbutton18.  function pushbutton18\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton18 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  [filename, pathname] = uigetfile({'\*.jpg;\*.jpeg;\*.png;\*.bmp','All Image Files'; ...  '\*.\*','All Files'}, 'Select a Background Image');  if isequal(filename, 0) || isequal(pathname, 0)  % 用户取消了文件选择  return;  end  % 构建完整的文件路径  fullpath = fullfile(pathname, filename);  % 读取背景图像  background\_img = imread(fullpath);  % 将背景图像存储到handles结构体中  handles.backgroundImage = background\_img;  guidata(hObject, handles); % 更新handles结构体  % 显示背景图像名  set(handles.edit2, 'String', filename);  % 在axes3中显示背景图像  axes(handles.axes3);  imshow(background\_img);  title('背景图像');  function edit2\_Callback(hObject, eventdata, handles)  % hObject handle to edit2 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % Hints: get(hObject,'String') returns contents of edit2 as text  % str2double(get(hObject,'String')) returns contents of edit2 as a double  % --- Executes during object creation, after setting all properties.  function edit2\_CreateFcn(hObject, eventdata, handles)  % hObject handle to edit2 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles empty - handles not created until after all CreateFcns called  % Hint: edit controls usually have a white background on Windows.  % See ISPC and COMPUTER.  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  set(hObject,'BackgroundColor','white');  end  % --- Executes on button press in pushbutton19.  function pushbutton19\_Callback(hObject, eventdata, handles)  % 检查是否已经加载了原图和背景图  if ~isfield(handles, 'imageData') || isempty(handles.imageData) || ...  ~isfield(handles, 'backgroundImage') || isempty(handles.backgroundImage)  warndlg('请先加载原图和背景图', '缺少图像');  return;  end  % 将图像转换为double类型  original\_img = im2double(handles.imageData);  background\_img = im2double(handles.backgroundImage);  % 确保背景图像是灰度图像  if size(background\_img, 3) == 3  backgroundGray = my\_rgb2gray(background\_img); % 转换为灰度图像  else  backgroundGray = background\_img; % 如果已经是灰度图像，则直接使用  end  % 将背景图像调整到与原图相同的大小  if ~isequal(size(original\_img(:,:,1)), size(backgroundGray))  backgroundGray = imresize(backgroundGray, size(original\_img(:,:,1)));  end  % 将背景灰度图像归一化到[0, 1]范围，以便作为掩码使用  backgroundMask = mat2gray(backgroundGray);  % 使用掩码从原图中提取目标区域（图像乘法）  if size(original\_img, 3) == 3 % 如果是RGB图像  targetImage = bsxfun(@times, original\_img, cast(backgroundMask, class(original\_img))); % 应用掩码到每个颜色通道并保持原始图像的数据类型  else % 如果是灰度图像  targetImage = bsxfun(@times, original\_img, cast(backgroundMask, class(original\_img))); % 应用掩码到灰度图像并保持原始图像的数据类型  end  % 显示结果  axes(handles.axes7);  imshow(targetImage);  title('提取的目标区域');  % 更新handles结构体中的目标数据  handles.targetImage = targetImage;  guidata(hObject, handles);  function pushbutton20\_Callback(hObject, eventdata, handles)  % 检查是否有灰度图像  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  gray\_img = im2double(handles.grayImageData);  % 提取并显示LBP特征图像  lbp\_img = computeLBPImage(gray\_img); % 假设有一个computeLBPImage函数来生成LBP图像  axes(handles.axes8);  imshow(lbp\_img, []);  title('原始图像的LBP特征');  % 打印LBP特征信息  disp('原始图像的LBP特征:');  disp(computeLBPHistogram(lbp\_img)); % 假设有一个函数来计算LBP直方图作为特征  % 更新handles结构体中的特征数据  handles.lbpFeaturesOriginal = computeLBPHistogram(lbp\_img);  guidata(hObject, handles);  function pushbutton21\_Callback(hObject, eventdata, handles)  % 检查是否有目标区域图像  if ~isfield(handles, 'targetImage') || isempty(handles.targetImage)  warndlg('请先提取目标区域', '无目标区域');  return;  end  target\_gray\_img = my\_rgb2gray(im2double(handles.targetImage));  % 提取并显示目标区域的LBP特征图像  lbp\_img\_target = computeLBPImage(target\_gray\_img);  axes(handles.axes8);  imshow(lbp\_img\_target, []);  title('目标区域的LBP特征');  % 打印LBP特征信息  disp('目标区域的LBP特征:');  disp(computeLBPHistogram(lbp\_img\_target));  % 更新handles结构体中的特征数据  handles.lbpFeaturesTarget = computeLBPHistogram(lbp\_img\_target);  guidata(hObject, handles);  % --- Executes on button press in pushbutton22.  function pushbutton22\_Callback(hObject, eventdata, handles)  if ~isfield(handles, 'grayImageData') || isempty(handles.grayImageData)  warndlg('请先将图像灰度化', '无灰度图像');  return;  end  gray\_img = im2double(handles.grayImageData);  % 使用原始的HOG特征提取逻辑  hog\_features = extractHOGFeaturesManual(gray\_img);  % 显示原始图像  axes(handles.axes8);  imshow(gray\_img);  title('原始图像');  % 创建一个新的figure窗口来展示HOG特征的mesh图  figure;  mesh(hog\_features);  title('原始图像的HOG特征');  xlabel('Block Index');  ylabel('Feature Vector Elements');  zlabel('Magnitude');  % 打印HOG特征信息  disp('原始图像的HOG特征:');  disp(hog\_features);  % 更新handles结构体中的特征数据  handles.hogFeaturesOriginal = hog\_features;  guidata(hObject, handles);  % --- Executes on button press in pushbutton23.  function pushbutton23\_Callback(hObject, eventdata, handles)  if ~isfield(handles, 'targetImage') || isempty(handles.targetImage)  warndlg('请先提取目标区域', '无目标区域');  return;  end  target\_gray\_img = rgb2gray(im2double(handles.targetImage)); % 假设targetImage可能是RGB  % 使用原始的HOG特征提取逻辑  hog\_features = extractHOGFeaturesManual(target\_gray\_img);  % 显示目标区域的灰度图像  axes(handles.axes8);  imshow(target\_gray\_img);  title('目标区域 (灰度图像)');  % 创建一个新的figure窗口来展示HOG特征的mesh图  figure;  mesh(hog\_features);  title('目标区域的HOG特征');  xlabel('Block Index');  ylabel('Feature Vector Elements');  zlabel('Magnitude');  % 打印HOG特征信息  disp('目标区域的HOG特征:');  disp(hog\_features);  % 更新handles结构体中的特征数据  handles.hogFeaturesTarget = hog\_features;  guidata(hObject, handles);  % --- Executes on button press in pushbutton25.  function pushbutton25\_Callback(hObject, eventdata, handles)  % hObject handle to pushbutton25 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % 获取之前存储的图像数据  img = handles.imageData;  % 初始化 BirdClassifier 类的对象  birdClassifier = BirdClassifier();  % 使用 BirdClassifier 对象进行预测  predictionResult = birdClassifier.predictBirdType(img);  % 将预测结果显示在 edit4 编辑框中  set(handles.edit4, 'String', predictionResult);  % 更新图形界面以反映更改  guidata(hObject, handles);  function edit4\_Callback(hObject, eventdata, handles)  % hObject handle to edit4 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles structure with handles and user data (see GUIDATA)  % Hints: get(hObject,'String') returns contents of edit4 as text  % str2double(get(hObject,'String')) returns contents of edit4 as a double  % --- Executes during object creation, after setting all properties.  function edit4\_CreateFcn(hObject, eventdata, handles)  % hObject handle to edit4 (see GCBO)  % eventdata reserved - to be defined in a future version of MATLAB  % handles empty - handles not created until after all CreateFcns called  % Hint: edit controls usually have a white background on Windows.  % See ISPC and COMPUTER.  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  set(hObject,'BackgroundColor','white');  end |  （四）运行结果   图4.1 GUI界面    图4.2 导入图像并运用模型预测    图4.3 灰度直方图与直方图均衡化    图4.4 直方图匹配    图4.5 灰度化图片    图4.6 图像缩放    图4.7 图像旋转    图4.8 线性增强    图4.9 对数增强    图4.10 指数增强    图4.11 椒盐噪声（参数0.05）    图4.12 高斯噪声（参数0 0.01）    图4.13 泊松噪声    图4.14 低通滤波    图4.15 高通滤波    图4.16 空域均值滤波    图4.17 空域高斯滤波    图4.18 空域中值滤波    图4.19 空域双边滤波    图4.20 Roberts算子边缘检测    图4.21 Prewitt算子边缘检测    图4.22 Sobel算子边缘检测    图4.23拉普拉斯算子边缘检测    图4.24 导入背景得出目标    图4.25 原图像LBP特征提取    图4.26 提取图像LBP特征提取    图4.27 原图像HOG特征提取    图4.28 提取图像HOG特征提取 （五）实验结论及收获建议 在本次大作业中，我通过广泛检索网络博客、深入咨询人工智能大模型、积极与杰出同学交流，深入挖掘课本内容，自主编写了大部分图像处理函数。在此过程中，我不仅掌握了运用MATLAB这一强大工具实现图形用户界面（GUI）的方法，还对图像处理的理论和实践有了更深入的理解。  在进行目标提取、局部二值模式（LBP）与方向梯度直方图（HOG）特征提取以及大型模型训练调用的过程中，我遭遇了诸多挑战。这些挑战包括但不限于对知识点的不熟悉、算法的选择、参数的调整以及计算资源的限制。通过网络学习和优秀同学的指导，我勉强达到了相关要求。这一过程让我深刻认识到，反复尝试是获得更佳结果的必要途径。  Github链接：https://github.com/19220110wuxiuquan/Digital-Image-Processing |