```
classdef app1 < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure        matlab.ui.Figure
        Label_5         matlab.ui.control.Label
        Label_4         matlab.ui.control.Label
        Label_3         matlab.ui.control.Label
        Label           matlab.ui.control.Label
        Button_4        matlab.ui.control.Button
        Label_2         matlab.ui.control.Label
        V10Label        matlab.ui.control.Label
        KNNButton       matlab.ui.control.Button
        Button_3        matlab.ui.control.Button
        Button_2        matlab.ui.control.Button
        Button          matlab.ui.control.Button
        TextArea_Output  matlab.ui.control.TextArea
    end
    % Callbacks that handle component events
    methods (Access = private)
        % Button pushed function: Button
        function ButtonPushed(app, event)
    % 选择输入和输出文件夹路径
    inputFolder = uigetdir('选择输入文件夹路径');
    outputFolder = uigetdir('选择输出文件夹路径');
    app.TextArea_Output.Value = "开始处理图像";
    drawnow;
    % 获取输入文件夹中的所有 BMP 文件
    fileList = dir(fullfile(inputFolder, '*.bmp'));
    % 扩展原图像的高斯滤波，对应文档的方案 2
    sigma = 1;     % sigma 赋值
    N = 1;          % 大小是（2N+1）×（2N+1）
    N_row = 2*N+1;
    gausFilter = fspecial('gaussian',[N_row N_row],sigma); % MATLAB 自带
高斯模板滤波
    for i = 1:numel(fileList)
        % 读取原始图像
        filename = fileList(i).name;
        originimg = imread(fullfile(inputFolder, filename));
        originimg = im2gray(originimg);
        [ori_row, ori_col] = size(originimg);
        % 应用高斯滤波
        blur = imfilter(originimg, gausFilter, 'conv');
        % 求高斯模板 H
        H = zeros(N_row, N_row);
        for ai = 1:N_row
            for aj = 1:N_row
                fenzi = double((ai - N - 1)^2 + (aj - N - 1)^2);
                H(ai, aj) = exp(-fenzi / (2 * sigma * sigma)) / (2 * pi *
sigma);
            end
        end
        H = H / sum(H(:)); % 归一化
        desimg = zeros(ori_row, ori_col); % 滤波后图像
        midimg = zeros(ori_row + 2 * N, ori_col + 2 * N); % 中间图像
        for ai = 1:ori_row % 原图像赋值给中间图像，四周边缘设置为0
            for aj = 1:ori_col
                midimg(ai + N, aj + N) = originimg(ai, aj);
            end
        end
        for ai = N + 1:ori_row + N
            for aj = N + 1:ori_col + N
                temp_row = ai - N;
                temp_col = aj - N;
                temp = 0;
                for bi = 1:N_row
                    for bj = 1:N_row
                        temp = temp + (midimg(temp_row + bi - 1, temp_col
+ bj - 1) * H(bi, bj));
                    end
                end
                desimg(temp_row, temp_col) = temp;
            end
        end
        desimg = uint8(desimg);
        % 直方图均衡化
        I1 = histeq(blur, 2);
        % 二值化
        thresh2 = graythresh(desimg); % 针对灰度图自动确定二值化阈值
        I2 = imbinarize(desimg,0.75);
        % 去除小的像素聚类
        I5 = bwareaopen(I2, 65);
        % 保存处理后的图像
        [~, filenameWithoutExtension, ~] = fileparts(filename);
        outputFilename = fullfile(outputFolder,
[filenameWithoutExtension '.bmp']);
        imwrite(I5, outputFilename);
       processedImageMsg = sprintf('Processed image saved: %s\n',
outputFilename);
app.TextArea_Output.Value = [app.TextArea_Output.Value;
{processedImageMsg}]; % 在现有输出的基础上追加输出
    end
    app.TextArea_Output.Value = app.TextArea_Output.Value + "图像处理完
成。";
    drawnow; % 立即刷新图形，以显示每张图像处理的进度
    fprintf('Processing completed.\n');
        end
        % Button pushed function: Button_2
        function Button_2Pushed(app, event)
            % 设置输入和输出文件夹路径
inputFolder = uigetdir('Select the input folder');
outputFolder = uigetdir('Select the output folder');
app.TextArea_Output.Value = "开始训练样本切块处理...";
drawnow;
% 创建输出文件夹
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end
% 获取输入文件夹中的所有 BMP 图像文件
fileList = dir(fullfile(inputFolder, '*.bmp'));
% 循环处理每个输入图像文件
for i = 1:numel(fileList)
    % 读取输入的二值 BMP 图像
    filename = fullfile(inputFolder, fileList(i).name);
    inputImg = imread(filename);
    % 创建当前图像的输出文件夹
    imageOutputFolder = fullfile(outputFolder, fileList(i).name(1:end-
4));
    if ~exist(imageOutputFolder, 'dir')
        mkdir(imageOutputFolder);
    end
    % 连通区域标记矩阵
    labeledImg = zeros(size(inputImg));
    % 当前连通区域的标记值
    label = 1;
    % 循环遍历图像的每个像素
    for row = 1:size(inputImg, 1)
        for col = 1:size(inputImg, 2)
            % 如果当前像素是前景像素且未被标记
            if inputImg(row, col) == 1 && labeledImg(row, col) == 0
                % 使用深度优先搜索进行连通区域标记
                labeledImg = dfs(inputImg, labeledImg, row, col, label);
                label = label + 1;
            end
        end
    end
    % 提取字符区域并保存为 28x28 像素大小的 BMP 文件
    for l = 1:label-1
        % 找到当前连通区域的所有像素坐标
        [rows, cols] = find(labeledImg == l);
        % 计算字符区域的边界框
        minRow = min(rows);
        maxRow = max(rows);
        minCol = min(cols);
        maxCol = max(cols);
        % 截取字符区域
        charImg = inputImg(minRow:maxRow, minCol:maxCol);
        % 检查字符区域是否大于等于 10x10 像素
        if size(charImg, 1) >= 10 && size(charImg, 2) >= 10
            % 调整图像大小为 28x28 像素
            charImgResized = imresize(charImg, [28, 28]);
            % 生成保存路径和文件名
            [~, filenameWithoutExtension, ~] = fileparts(filename);
            charFilename = fullfile(imageOutputFolder,
[filenameWithoutExtension, '_', num2str(l), '.bmp']);
            % 保存字符区域为单独的 BMP 文件
            imwrite(charImgResized, charFilename);
        end
    end
end
% 深度优先搜索函数
function labeledImg = dfs(inputImg, labeledImg, row, col, label)
    % 标记当前像素
    labeledImg(row, col) = label;
    % 定义上、下、左、右四个方向的相对坐标
    directions = [-1, 0; 1, 0; 0, -1; 0, 1];
    % 循环遍历四个方向
    for d = 1:size(directions, 1)
        % 计算相邻像素的坐标
        newRow = row + directions(d, 1);
        newCol = col + directions(d, 2);
        % 检查相邻像素是否在图像范围内且是前景像素且未被标记
        if newRow >= 1 && newRow <= size(inputImg, 1) && newCol >= 1 &&
newCol <= size(inputImg, 2) && inputImg(newRow, newCol) == 1 &&
labeledImg(newRow, newCol) == 0
            % 递归调用深度优先搜索
            labeledImg = dfs(inputImg, labeledImg, newRow, newCol,
label);
        end
    end
end
app.TextArea_Output.Value = "训练样本切块处理完成...";
        end
        % Button pushed function: Button_3
        function Button_3Pushed(app, event)
            % 设置输入和输出文件夹路径
sourceFolder = uigetdir('Select the source folder');
destinationFolder = uigetdir('Select the destination folder');
app.TextArea_Output.Value = "开始合并...";
drawnow;
% 调用函数将所有子文件夹中的文件提取到目标文件夹
extractFilesFromSubfolders(sourceFolder, destinationFolder);
% 递归遍历文件夹并将所有文件提取到目标文件夹
function extractFilesFromSubfolders(sourceFolder, destinationFolder)
    % 获取源文件夹中的所有文件和子文件夹
    fileList = dir(sourceFolder);
    % 遍历源文件夹中的所有文件和子文件夹
    for i = 1:numel(fileList)
        % 忽略特殊文件夹（'.'和'..'）
        if strcmp(fileList(i).name, '.') || strcmp(fileList(i).name,
'..')
            continue;
        end
                % 构建当前文件或子文件夹的完整路径
        itemPath = fullfile(sourceFolder, fileList(i).name);
        % 检查是否为文件
        if fileList(i).isdir
            % 如果是子文件夹，则递归调用本函数继续处理子文件夹
            extractFilesFromSubfolders(itemPath, destinationFolder);
        else
            % 如果是文件，则将文件复制到目标文件夹中
            [~, filename, fileExt] = fileparts(fileList(i).name);
            destinationFile = fullfile(destinationFolder, [filename,
fileExt]);
            copyfile(itemPath, destinationFile);
        end
    end
end
app.TextArea_Output.Value = "合并完成...";
        end
        % Button pushed function: KNNButton
        function KNNButtonPushed(app, event)
            % 指定图像文件夹路径
imageFolderPath = uigetdir('Select the source folder');
app.TextArea_Output.Value = "开始训练KNN模型...";
drawnow;
% 获取文件夹中的所有图像文件
imageFiles = dir(fullfile(imageFolderPath, '**', '*.bmp'));
numImages = numel(imageFiles);
% 创建存储图像和标签的变量
images = cell(numImages, 1);
labels = zeros(numImages, 1);
% 遍历图像文件，提取图像和标签
for i = 1:numImages
    % 获取图像文件名和标签
    imageFilename = imageFiles(i).name;
    [~, name, ~] = fileparts(imageFilename);
    % 提取标签
    underscoreIndex = strfind(name, '_');
    label = str2double(name(1:underscoreIndex-1));
    % 读取图像
    image = imread(fullfile(imageFolderPath, imageFilename));
    % 存储图像和标签
    images{i} = image;
    labels(i) = label;
end
% 将数据随机划分为训练图像和测试图像
rng(42); % 设置随机数种子，以确保结果可重复
trainRatio = 0.8; % 80% 作为训练图像，20% 作为测试图像
numTrainImages = round(numImages * trainRatio);
trainIndices = randperm(numImages, numTrainImages);
trainImages = images(trainIndices);
trainLabels = labels(trainIndices);
testIndices = setdiff(1:numImages, trainIndices);
testImages = images(testIndices);
testLabels = labels(testIndices);
% 提取训练图像特征
% 这里使用的是图像的像素值作为特征
trainFeatures = cell(numTrainImages, 1);
for i = 1:numTrainImages
    binaryImage = trainImages{i};
    imageFeatures = double(binaryImage(:));  % 将特征转换为数值类型
    trainFeatures{i} = imageFeatures';
end
% 步骤6：将训练特征矩阵转换为训练数据
trainX = cell2mat(trainFeatures);
trainY = trainLabels;
% 步骤7：使用K折交叉验证进行多次迭代
numFolds = 3;  % 设置K折交叉验证的折数
numLabels = 10;  % 标签的数量
accuracyMatrix = zeros(numFolds, numLabels);  % 存储准确率的矩阵
```

```
for fold = 1:numFolds
    % 划分训练集和验证集
    cv = cvpartition(numTrainImages, 'KFold', numFolds);
    trainIdx = training(cv, fold);
    validationIdx = test(cv, fold);
    trainX_fold = trainX(trainIdx, :);
    trainY_fold = trainY(trainIdx);
    validationX_fold = trainX(validationIdx, :);
    validationY_fold = trainY(validationIdx);
        % 训练 KNN 分类器
    knnModel = fitcknn(trainX_fold, trainY_fold, 'NumNeighbors', 10);
        % 使用验证集进行预测
    predictedLabels = predict(knnModel, validationX_fold);
    % 计算准确率
    for label = 0:numLabels-1
        idx = validationY_fold == label;
        accuracy = sum(predictedLabels(idx) == validationY_fold(idx)) /
sum(idx);
        accuracyMatrix(fold, label+1) = accuracy;
    end
        % 输出每次迭代后的准确率矩阵
     disp(['迭代 ', num2str(fold), ' 的准确率矩阵：']);
    disp(accuracyMatrix);
     % 输出每次迭代后的准确率矩阵
    %  outputMsg = ['迭代 ', num2str(fold), ' 的准确率矩阵：' + newline];
    % outputMsg = [outputMsg, mat2str(accuracyMatrix) + newline];
    % app.TextArea_Output.Value = string([app.TextArea_Output.Value;
outputMsg]);
    % 计算混淆矩阵
    confusionMatrix = confusionmat(validationY_fold, predictedLabels);
    % 将混淆矩阵中的数量转换为准确率
    confusionMatrix = confusionMatrix ./ sum(confusionMatrix, 2);
    % 输出混淆矩阵
    disp(['迭代 ', num2str(fold), ' 的混淆矩阵：']);
    disp(confusionMatrix);
end
% 生成混淆矩阵的文件名，包含迭代次数
confusionMatrixFilename = sprintf('confusion_matrix_iteration_%d.xlsx',
fold);
% 使用 xlswrite 函数将混淆矩阵写入 Excel 文件
xlswrite(confusionMatrixFilename, confusionMatrix, 'ConfusionMatrix');
app.TextArea_Output.Value='训练模型的混淆矩阵已写入 Excel 文
件:confusion_matrix_iteration_%d.xlsx';
app.TextArea_Output.Value='训练好的 KNN 模型已保存为 trained_knn_model.mat
文件,训练模型的混淆矩阵已写入 Excel 文
件:confusion_matrix_iteration_%d.xlsx';
        end
        % Value changed function: TextArea_Output
        function TextArea_OutputValueChanged(app, event)
            value = app.TextArea_Output.Value;
        end
        % Button pushed function: Button_4
        function Button_4Pushed(app, event)
            % 加载训练好的 KNN 模型
load('trained_knn_model.mat');
% 提示用户选择一张手写数字的图片
[imageFilename, imageFolderPath] = uigetfile('*.bmp', 'Select the hand-
written digit image');
% 读取选择的图片
testImage = imread(fullfile(imageFolderPath, imageFilename));
% 将图像转为灰度图像（如果是彩色图像的话）
if size(testImage, 3) > 1
    grayImage = rgb2gray(testImage);
else
    grayImage = testImage;
end
% 高斯滤波
filteredImage = imgaussfilt(grayImage, 1); % 这里的 1 是高斯滤波的标准差，
可以根据需要调整
```

```
% 二值化处理
threshold = graythresh(filteredImage); % 使用 Otsu 阈值确定二值化阈值
binaryImage = imbinarize(filteredImage, threshold);
% 去除小的像素聚类
    binaryImage = bwareaopen(binaryImage, 5);
% 调整图片大小为 28x28 像素
resizedImage = imresize(binaryImage, [28, 28]);
subplot(1, 3, 1);
imshow(testImage);
title('手写数字图片');
% 子图 1: 处理后的二值化图片
subplot(1, 3, 2);
imshow(binaryImage);
title('处理后的二值化图片');
% 子图 2: 调整后的图片
subplot(1, 3, 3);
imshow(resizedImage);
title('调整后的图片');
% 提取测试图片特征
testFeatures = double(binaryImage(:)');
% 使用加载的 KNN 模型进行预测
[predictedLabel, ~] = predict(knnModel, testFeatures);
% 显示预测结果
app.TextArea_Output.Value=(['预测结果: ', num2str(predictedLabel)]);
        end
    end
    % Component initialization
    methods (Access = private)
        % Create UIFigure and components
        function createComponents(app)
            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';
            % Create TextArea_Output
            app.TextArea_Output = uitextarea(app.UIFigure);
            app.TextArea_Output.ValueChangedFcn = createCallbackFcn(app,
@TextArea_OutputValueChanged, true);
            app.TextArea_Output.Position = [83 21 531 299];
            % Create Button
            app.Button = uibutton(app.UIFigure, 'push');
            app.Button.ButtonPushedFcn = createCallbackFcn(app,
@ButtonPushed, true);
            app.Button.Position = [31 397 111 23];
            app.Button.Text = '训练样本降噪处理';
            % Create Button_2
            app.Button_2 = uibutton(app.UIFigure, 'push');
            app.Button_2.ButtonPushedFcn = createCallbackFcn(app,
@Button_2Pushed, true);
            app.Button_2.Position = [189 397 111 23];
            app.Button_2.Text = '训练样本切块处理';
            % Create Button_3
            app.Button_3 = uibutton(app.UIFigure, 'push');
            app.Button_3.ButtonPushedFcn = createCallbackFcn(app,
@Button_3Pushed, true);
            app.Button_3.Position = [351 397 100 23];
            app.Button_3.Text = '合并样本';
            % Create KNNButton
            app.KNNButton = uibutton(app.UIFigure, 'push');
            app.KNNButton.ButtonPushedFcn = createCallbackFcn(app,
@KNNButtonPushed, true);
            app.KNNButton.Position = [514 397 100 23];
            app.KNNButton.Text = 'KNN 模型训练';
            % Create V10Label
            app.V10Label = uilabel(app.UIFigure);
            app.V10Label.FontSize = 18;
            app.V10Label.Position = [148 434 353 23];
            app.V10Label.Text = '噪声手写数字图片中手写数字识别系统 V1.0';
            % Create Label_2
            app.Label_2 = uilabel(app.UIFigure);
            app.Label_2.HorizontalAlignment = 'right';
            app.Label_2.FontSize = 18;
            app.Label_2.Position = [11 259 59 23];
            app.Label_2.Text = '消息框';
            % Create Button_4
            app.Button_4 = uibutton(app.UIFigure, 'push');
            app.Button_4.ButtonPushedFcn = createCallbackFcn(app, 
@Button_4Pushed, true);
            app.Button_4.Position = [224 335 205 46];
            app.Button_4.Text = '预测手写数字';
            % Create Label
            app.Label = uilabel(app.UIFigure);
            app.Label.Position = [438 347 125 22];
            app.Label.Text = '需要有训练好的模型！';
            % Create Label_3
            app.Label_3 = uilabel(app.UIFigure);
            app.Label_3.Position = [148 397 32 22];
            app.Label_3.Text = '----->';
            % Create Label_4
            app.Label_4 = uilabel(app.UIFigure);
            app.Label_4.Position = [308 397 36 22];
            app.Label_4.Text = '------>';
            % Create Label_5
            app.Label_5 = uilabel(app.UIFigure);
            app.Label_5.Position = [467 397 36 22];
            app.Label_5.Text = '------>';
            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end
    % App creation and deletion
    methods (Access = public)
        % Construct app
        function app = app1
            % Create UIFigure and components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```