

**QATAR UNIVERSITY**

**COLLEGE OF ENGINERING**

**CSE DEPARTMENT**

**CMPE 485 FUND OF DIGITAL IMAGE PROCESSING**

**CRN\_25187\_L51**

**Dr. Sumaya Almadeed**

**SPRING 2023**

**Project 1: Preprocessing & ML**

|  |  |  |
| --- | --- | --- |
| **STUDENT NAME** | **STUDENT ID** | **Work percentage** |
| Mallak Abunimeh | 201903622 | 50% |
| Nirvana Aladl | 201903622 | 50% |

**Preprocessing code:**

This code performs a series of image processing steps to preprocess a dataset of tomato images. The steps are:

1. Set the paths for the original images dataset folder (raw\_data\_dir) and the destination folder for the preprocessed dataset (preprocessed\_data\_dir).
2. Check if the preprocessed\_data\_dir folder exists, and create it if it doesn't.
3. Create a cell array destination\_folders with the preprocessed\_data\_dir as the only element. This is used to specify the destination folder for saving the preprocessed images.
4. Use the dir function to retrieve a structure array of all files with the extension ".jpg" in the raw\_data\_dir folder. The variable file\_path will store this information.
5. Assign labels to each image based on whether the image file name contains the substring 'diseased' or not. Images with filenames containing 'diseased' are labeled as 1, and others are labeled as 0.
6. Initialize empty matrices features and labels\_list to store the feature vectors and labels, respectively.
7. Iterate over each image file in the dataset. For each image:

* Create the full file path for the current image.
* Read the image using imread.
* Check if the image has a colormap. If it does, remove the colormap using ind2rgb.
* Convert the image from RGB color space to HSV color space using rgb2hsv.
* Apply a Gaussian filter to the original image using imgaussfilt.
* Convert the original image to grayscale using rgb2gray.
* Apply histogram equalization to the grayscale image using histeq.
* Display the original image, HSV image, hue channel, saturation channel, and value channel.
* Apply thresholding on the saturation channel to create a binary mask.
* Apply morphological operations (dilation and erosion) to the binary mask.
* Perform blob detection on the eroded mask using regionprops to extract properties of connected components in the binary image.
* If no objects were detected, continue to the next iteration.
* Display the original image with rectangles around the detected objects.
* Save the image with the detected objects in the preprocessed data directory.
* Construct a feature vector for the current image by concatenating the bounding box coordinates of the first detected blob with the mean values of the hue, saturation, and value channels.
* Append the feature vector to the features matrix and the corresponding label to the labels\_list.

1. Shuffle the data by generating random permutation indices and rearranging the features and labels accordingly.
2. Normalize the features using the normalize function.
3. Divide the data into training and testing sets based on a specified ratio.
4. Save the preprocessed data, including the training and testing features and labels, into a .mat file in the preprocessed data directory.
5. Display a message indicating that the preprocessing is complete.

Overall, this code applies various image processing techniques such as color space conversion, filtering, thresholding, and morphological operations to preprocess a dataset of tomato images. It extracts relevant features from the images and prepares the data for further analysis or machine learning tasks.

**The code:**

% original images dataset folder path

raw\_data\_dir = '\\quad\users\CTXUSRFR\ma1903622\Desktop\dataset\tomato\tomato'; %This line specifies the path to the folder where the original images dataset is located.

% destination to the preprocessed dataset folder

preprocessed\_data\_dir = '\\quad\users\CTXUSRFR\ma1903622\Desktop\dataset\tomato\_updated'; %This line specifies the path to the destination folder where the preprocessed dataset will be stored.

% Check if the preprocessed\_data\_dir exists, and create it if it doesn't

if ~isfolder(preprocessed\_data\_dir)

mkdir(preprocessed\_data\_dir);

end

% destination folder path

destination\_folders = {preprocessed\_data\_dir}; %This line creates a cell array destination\_folders with the preprocessed\_data\_dir as the only element. This is used to specify the destination folder for saving the preprocessed images.

% creating a full file path with the specified type

file\_path = dir(fullfile(raw\_data\_dir, '\*.jpg')); %This line uses the dir function to retrieve a structure array of all files with the extension .jpg in the raw\_data\_dir folder.

% create labels for each image

%This code assigns labels to each image based on whether the image file name contains the substring 'diseased' or not. Images with filenames containing 'diseased' are labeled as 1, and others are labeled as 0.

labels = zeros(numel(file\_path), 1);

for i = 1:numel(file\_path)

if contains(file\_path(i).name, 'diseased')

labels(i) = 1;

else

labels(i) = 0;

end

end

% initialize the features and labels matrices

%These lines initialize empty matrices features and labels\_list to store the feature vectors and labels, respectively.

features = [];

labels\_list = [];

% looping to preprocess each image in the dataset

%These lines iterate over each image file in the dataset. It creates the full file path for the current image, reads the image using imread, and stores it in the variable image.

for i = 1:numel(file\_path)

% creating a full file path to the image file in the current iteration

filename = fullfile(raw\_data\_dir, file\_path(i).name);

fprintf('Reading the files %d of %d: %s\n', i, numel(file\_path), filename);

% reading the image

image = imread(filename);

% check if the image has a colormap

%This code checks if the image has a colormap by checking the number of dimensions (ndims(image)). If the image has two dimensions, it means it has a colormap, and the colormap is removed using ind2rgb.

if ndims(image) == 2

% remove the colormap

image = ind2rgb(image, gray(256));

end

% convert the image to hsv

%This line converts the image from RGB color space to HSV color space using the rgb2hsv function.

hsv\_image = rgb2hsv(image);

% applying the Gaussian filter

%This line applies a Gaussian filter to the original image (image) using a standard deviation of 3. The filtered image is stored in the variable img\_filtered\_gaussian.

img\_filtered\_gaussian = imgaussfilt(image, 3);

% converting to grayscale

%This line converts the original image (image) to grayscale using the rgb2gray function. The grayscale image is stored in the variable img\_gray

img\_gray = rgb2gray(image);

% applying histogram equalization

%This line applies histogram equalization to the grayscale image (img\_gray) using the histeq function. Histogram equalization enhances the contrast of the image by redistributing the pixel intensities. The equalized image is stored in the variable img\_eq.

img\_eq = histeq(img\_gray);

% display the original image

figure, imshow(image); title('Original Image');

% display the HSV image

figure, imshow(hsv\_image); title('HSV Image');

% extracting the hue, saturation, and value channels

%These lines extract the individual channels of the HSV image. The hue, saturation, and value channels are stored in the variables h, s, and v, respectively.

h = hsv\_image(:, :, 1); % Hue channel

s = hsv\_image(:, :, 2); % Saturation channel

v = hsv\_image(:, :, 3); % Value (intensity) channel

% display the hue, saturation, and value channels

figure, imshow(h); title('Hue Channel');

figure, imshow(s); title('Saturation Channel');

figure, imshow(v); title('Value (Intensity) Channel');

% applying thresholding on the saturation channel

%This code applies thresholding on the saturation channel (s). It calculates a threshold value as 0.0999 times the maximum value in s using max(s(:)). Then, it creates a binary mask where pixel values in s greater than the threshold are set to 1, and others are set to 0. The resulting mask is displayed in a new figure window titled "Thresholded Image".

threshold = max(s(:)) .\* 0.0999;

mask = s > threshold;

figure, imshow(mask); title('Thresholded Image');

%In this part of the code, morphological operations, specifically dilation and erosion, are applied to the binary image mask. These operations are commonly used for preprocessing and enhancing certain features in images.

% applying morphological operations (dilation and erosion)

se = strel('square', 2); %A structuring element (se) is created using the strel function. The 'square' argument specifies that the structuring element should be a square shape, and the 2 specifies its size (2x2).

dilated\_mask = imdilate(mask, se); %The imdilate function is used to perform the dilation operation on the binary image mask using the structuring element se. Dilation expands the boundaries of the white regions (foreground) in the image, which can help fill in gaps and connect nearby regions.

eroded\_mask = imerode(dilated\_mask, se); %The imerode function is used to perform the erosion operation on the dilated mask (dilated\_mask) using the same structuring element se. Erosion, on the other hand, shrinks the boundaries of the white regions, which can help remove noise and refine the shape of the detected objects.

figure, imshow(eroded\_mask); title('Morphologically Processed Image'); %This line displays the resulting morphologically processed image (eroded\_mask) using the imshow function. It shows the binary image after both dilation and erosion operations have been applied. The title of the figure is set to "Morphologically Processed Image".

% performing blob detection

%This line performs blob detection on the eroded mask (eroded\_mask) using the regionprops function. It extracts properties of connected components in the binary image, such as bounding boxes. The detected blobs are stored in the blob\_analysis structure.

blob\_analysis = regionprops(eroded\_mask, 'BoundingBox');

% check if any objects were detected

%This code checks if any objects were detected by the blob analysis. If the blob\_analysis structure is empty, it means no objects were detected. In such a case, the code continues to the next iteration of the loop, skipping further processing for the current image.

if isempty(blob\_analysis)

% handle the case where no objects were detected

continue; % move to the next image

end

% create a new figure to display the detected objects

figure('Name', 'Detected Objects');

imshow(image);

hold on;

% These lines create a new figure with the name "Detected Objects" and display the original image (image) in the figure. It then loops over each detected blob in the blob\_analysis structure and draws rectangles around them using the bounding box information. The rectangles are drawn with a green color and a line width of 2.

% draw rectangles around the detected objects

for j = 1:numel(blob\_analysis)

bbox = blob\_analysis(j).BoundingBox;

rectangle('Position', bbox, 'EdgeColor', 'g', 'LineWidth', 2);

end

% save the image with the detected objects

%This code extracts the name and extension of the current image file being processed. It uses the saveas function to save the current figure (with the detected objects) as an image file in the preprocessed data directory (preprocessed\_data\_dir). The image is saved with the same name and extension as the original image.

[~, name, ext] = fileparts(file\_path(i).name);

saveas(gcf, fullfile(preprocessed\_data\_dir, [name, ext]));

% collect the feature vector and corresponding label

%This line constructs a feature vector for the current image. It concatenates the bounding box coordinates of the first detected blob (blob\_analysis(1).BoundingBox) with the mean values of the hue (h), saturation (s), and value (v) channels. The feature vector represents the characteristics of the detected objects in the image.

features\_vector = [blob\_analysis(1).BoundingBox, mean(h(:)), mean(s(:)), mean(v(:))];

% concatenate the feature vector to the features matrix

%These lines append the feature vector (features\_vector) to the features matrix and the corresponding label (labels(i)) to the labels\_list. This step collects the features and labels for all the images in the dataset.

features = [features; features\_vector];

labels\_list = [labels\_list; labels(i)];

end

%This part: The code then continues to shuffle the data, normalize the features, divide the data into training and testing sets, and finally saves the preprocessed data into a .mat file.

% shuffle the data

shuffled\_indices = randperm(size(features, 1));

features = features(shuffled\_indices, :);

labels\_list = labels\_list(shuffled\_indices, :);

% normalize the features

features = normalize(features);

% divide the data into training and testing sets

train\_ratio = 0.7;

num\_train\_samples = round(train\_ratio \* size(features, 1));

train\_features = features(1:num\_train\_samples, :);

train\_labels = labels\_list(1:num\_train\_samples);

test\_features = features(num\_train\_samples+1:end, :);

test\_labels = labels\_list(num\_train\_samples+1:end);

% save the preprocessed data

save(fullfile(preprocessed\_data\_dir, 'preprocessed\_data.mat'), 'train\_features', 'train\_labels', 'test\_features', 'test\_labels');

disp('Preprocessing complete.');

**ML part:**

**train.m code:**

The train.m code allows the user to select a binary image, performs feature extraction on the image, and stores the extracted features along with a corresponding class label in a feature database file. If the database file already exists, it loads the existing database, appends the new features, and saves the updated database. If the database file doesn't exist, it creates a new database with the current features and saves it.

**train.m code:**

clc; %Clears the command window to remove any previous output or text.

clear all; %Clears all variables from the workspace, removing any stored data.

close all; %Closes all open figures or windows.

%% Taking an Image

[fname, path]=uigetfile('.jpg','Open an Image as input for training'); %Opens a file dialog box for the user to select a JPEG image file. It returns the file name (fname) and the path to the file (path).

fname=strcat(path, fname); %Concatenates the path and fname variables to form the complete file path of the selected image.

im=imread(fname); %Reads the image file specified by fname and stores it in the variable im.

im=im2bw(im); %Converts the image to binary (black and white) format using a thresholding operation.

imshow(im); %Displays the image in a figure window.

title('Input Image'); %Sets the title of the figure window to "Input Image".

c=input('Enter the Class Number from (1-4)'); %Prompts the user to enter a class number from 1 to 4. The input value is stored in the variable c.

%% Feature Extraction

F=FeatureStatistical(im); %Calls a function named FeatureStatistical to extract statistical features from the binary image im. The resulting features are stored in the variable F.

try %Starts a try-catch block to handle potential errors.

load db; %Loads a file named "db.mat" that contains a database of previously stored features and class labels.

F=[F c]; %Appends the class label c to the feature vector F.

db=[db; F]; %Concatenates the new feature vector F with the existing feature database db as a new row.

save db.mat db %Saves the updated feature database db to a file named "db.mat".

catch % Starts the catch block, which executes if an error occurs within the try block.

db=[F c]; %Creates a new feature database db with the current feature vector F and class label c.

save db.mat db %Saves the new feature database db to a file named "db.mat".

end

**test.m code:**

The test.m code allows the user to select a binary test image, extracts statistical features from it, compares the features with a database of training images, finds the closest match, and displays the detected class label of the test image using a message box.

**test.m code:**

%% Test Image

clc; %Clears the command window to remove any previous output or text.

clear all; %Clears all variables from the workspace, removing any stored data.

close all; %Closes all open figures or windows.

[fname, path]=uigetfile('.jpg','provide an Image for testing'); %Opens a file dialog box for the user to select a JPEG image file for testing. It returns the file name (fname) and the path to the file (path).

fname=strcat(path, fname); %Concatenates the path and fname variables to form the complete file path of the selected image.

im=imread(fname); %Reads the image file specified by fname and stores it in the variable im.

im=im2bw(im); %Converts the image to binary (black and white) format using a thresholding operation.

imshow(im); %Displays the image in a figure window.

title('Test Image'); %Sets the title of the figure window to "Test Image".

%% Find the class the test image belongs (related to finding the class of the test image)

Ftest=FeatureStatistical(im); %Calls a function named FeatureStatistical to extract statistical features from the binary test image im. The resulting features are stored in the variable Ftest.

%% Compare with the feature of training image in the database

load db.mat %Loads the feature database file "db.mat" that contains the training image features and class labels.

Ftrain=db(:,1:2); %Extracts the feature vectors of the training images from the loaded database and stores them in the variable Ftrain.

Ctrain=db(:,3); %Extracts the corresponding class labels of the training images from the loaded database and stores them in the variable Ctrain.

for (i=1:size(Ftrain,1)); %Starts a loop that iterates over the rows of Ftrain (number of training images).

dist(i,:)=sum(abs(Ftrain(i,:)-Ftest)); %Calculates the Manhattan distance (sum of absolute differences) between each training image feature vector and the test image feature vector. The distances are stored in the variable dist.

end %Ends the loop.

m=find(dist==min(dist),1); %Finds the index of the minimum distance in the dist vector, indicating the index of the closest match in the training feature vectors.

det\_class=Ctrain(m); %Retrieves the class label corresponding to the closest match from the Ctrain vector and stores it in the variable det\_class.

msgbox(strcat('Detected Class=',num2str(det\_class))); %Displays a message box showing the detected class label of the test image.

**FeatureStatistical.m function code (For feature extraction):**

The FeatureStatistical function takes an image im as input, converts it to double precision, calculates the mean and standard deviation of the pixel intensities in the image, and returns a feature vector F containing these statistical features.

This function is used to extract statistical features from images. During training, the FeatureStatistical function is called on a set of training images to calculate the mean and standard deviation of each image. These values are then stored along with their corresponding class labels in a feature database. During testing, the function is also used to extract the same statistical features from a test image. These features are compared with the features stored in the database to determine the closest match and classify the test image based on the nearest neighbor approach.

**FeatureStatistical.m function code:**

function [F]=FeatureStatistical(im) %This line declares the function FeatureStatistical that takes one input parameter im and returns one output variable F.

% Summary of this function goes here

im=double(im); %Converts the input image im from its original data type to double precision. This allows for more accurate statistical calculations.

m=mean(mean(im)); %Calculates the mean value of all the pixel intensities in the input image im. The function mean is applied twice, first to calculate the mean of each row, and then to calculate the mean of the resulting row means. The mean value is stored in the variable m.

s=std(std(im)); %Calculates the standard deviation of all the pixel intensities in the input image im. The function std is applied twice, first to calculate the standard deviation of each row, and then to calculate the standard deviation of the resulting row standard deviations. The standard deviation value is stored in the variable s.

F=[m s]; %Creates a feature vector F containing the mean (m) and standard deviation (s) values calculated from the input image. The values are concatenated into a 1x2 vector.

end %Ends the function.