SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

IT WORKSHOP SCILAB / MATLAB LABORATORY

Course Code: INT319

Semester: VII

Lab Manual

2024

SHANMUGHA ARTS, SCIENCE, TECHNOLOGY AND RESEARCH ACADEMY
(SASTRA Deemed to be) University
Tirumalaisamudram, Thanjavur-613 401
School of Computing

Course Objective:

This course will help the learner to develop applications using machine learning techniques such as supervised and unsupervised learning for classification and clustering.

Course Learning Outcomes:

Upon successful completion of this course, the learner will be able to

- Understand MATLAB commands, toolbox and functions
- Illustrate the image enhancement techniques
- Analyze the dataset using various plotting methods
- Demonstrate the use of image segmentation techniques
- Understand and apply the classification methods
- Develop applications for real time problem solving

List of Experiments:

1. Implementation of Histogram Mapping and Equalization.
2. Implementation of image smoothening Filters.
3. Implementation of image sharpening filters.
4. Visualize dataset using plotting methods.
5. Implementation of image morphology techniques.
6. Implementation of color image processing.
7. Implementation of thresholding methods in medical image segmentation.
8. Implementation of clustering methods in medical image segmentation.
9. Implementation of edge detection methods in object identification.
10. Computation of statistical features such as mean, standard Deviation, correlation coefficient of the given Image
11. Develop a GUI for fruit/vegetable classification.
12. Implementation of deep learning techniques for image classification.

Exercise No. 1 Histogram Mapping and Equalization

Implementation of Histogram Mapping and Equalization.

Objectives

To find out the histogram of an image and perform histogram equalization and histogram specification.

Concept

Histogram equalization is a technique used in image processing to enhance the contrast of an image by redistributing the intensity levels of pixels.

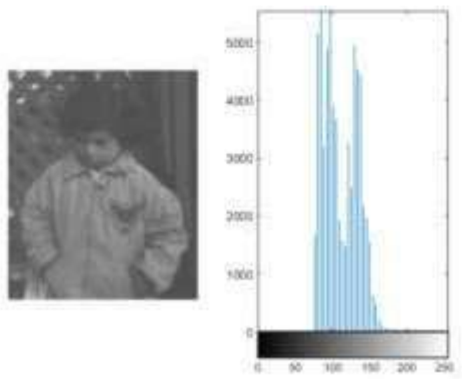
Histogram Equalization Process:

- Calculate the cumulative distribution function (CDF) of the histogram.
- Normalize the CDF to scale it to the range [0, 255] (for 8-bit images).
- Map each pixel intensity in the original image to its corresponding value in the normalized CDF.

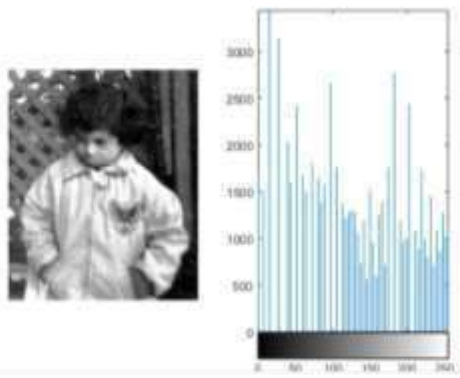
Procedure

- Read an image into the workspace
  ```
  I = imread('pout.tif')
  ```
- Display the image and its histogram.
  ```
  figure

  subplot(1,2,1)

  imshow(I)

  subplot(1,2,2)

  imhist(I,64)
  ```
- Adjust the contrast using histogram equalization
  ```
  J = histeq(I);
  ```
- Display the contrast-adjusted image and its new histogram.
  ```
  figure subplot(1,2,1)

  imshow(J)

  subplot(1,2,2)

  imhist(J,64)
  ```

Input:



Output:



Exercise No. 2   Image Smoothening

Implementation of image smoothening Filters.

Objectives

To apply Low pass filtering (smoothing) to remove high spatial frequency noise from a digital image and improve the quality and clarity of an image.

Concept

Image noise filters are essential in image processing to improve image quality by reducing unwanted noise while preserving important details. There are several types of noise filters commonly used, each suited to different types of noise and applications

1. Smoothing Filters:

Gaussian Blur Filter: Smooths the image by averaging pixel values within a neighborhood defined by a Gaussian kernel.

Median Filter: Replaces each pixel value with the median value in its neighborhood.

2. Frequency Domain Filters:

Wiener Filter: Minimize the mean square error between the original image and the filtered image in the presence of additive noise.

Procedure

- Read an image into the workspace using imread( )
- Display the image using imshow( )
- Apply the following filter to remove noise from image
- Median filter using medfilt2() to remove salt and pepper noise from image

```
I = imread('eight.tif');
figure, imshow(I)
```

- Add salt and pepper noise.

```
J = imnoise(I,'salt & pepper',0.02);
```

- Use a median filter to filter out the noise.

```
K = medfilt2(J);
```

- Display results, side-by-side.

```
imshowpair(J,K,'montage')
```

- Mean filter using the fspecial() and imfilter() function

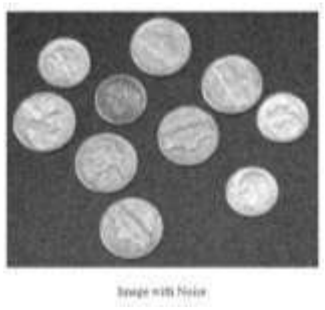
```
I = imread('cameraman.tif'); subplot(2,2,1);

imshow(I); title('Original Image');
```

```
H = fspecial('motion',20,45); MotionBlur = imfilter(I,H,'replicate'); subplot(2,2,2);
 imshow(MotionBlur);title('Motion Blurred Image');


  H = fspecial('disk',10);
  blurred = imfilter(I,H,'replicate'); subplot(2,2,3);
  imshow(blurred); title('Blurred Image');


  H = fspecial('unsharp');
  sharpened = imfilter(I,H,'replicate'); subplot(2,2,4);
   imshow(sharpened); title('Sharpened Image');
```

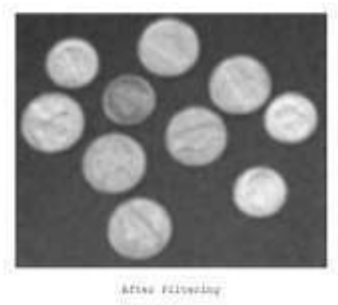
- Gaussian smoothing filters to images using imgaussfilt ()

```
 %Gaussian filter using MATLAB built_in function
  %Read an Image
   Img = imread('coins.png');
   A = imnoise(Img,'Gaussian',0.04,0.003);
   %Image with noise figure,imshow(A);


     H = fspecial('Gaussian',[9 9],1.76); GaussF = imfilter(A,H); figure,imshow(GaussF);
```

Sample Input



Output:



Exercise No. 3   Image Sharpening Filters

Implementation of image sharpening filters.

Objectives

To apply sharpening filtering to highlight fine details of the small objects on the image.

Concept

Image sharpening filters are used to enhance the clarity and detail of images by emphasizing edges and fine details.

Laplacian Filter: Enhances edges by highlighting rapid intensity changes in the image.

High Boost Filtering: Enhances edges while minimizing noise amplification.

Procedure

- Read an image into the workspace and display it.

```
a = imread('hestain.png'); imshow(a)
title('Original Image');
```

- Sharpen the image using the imsharpen function and display it.

```
b = imsharpen(a);
figure, imshow(b)
title('Sharpened Image');
```

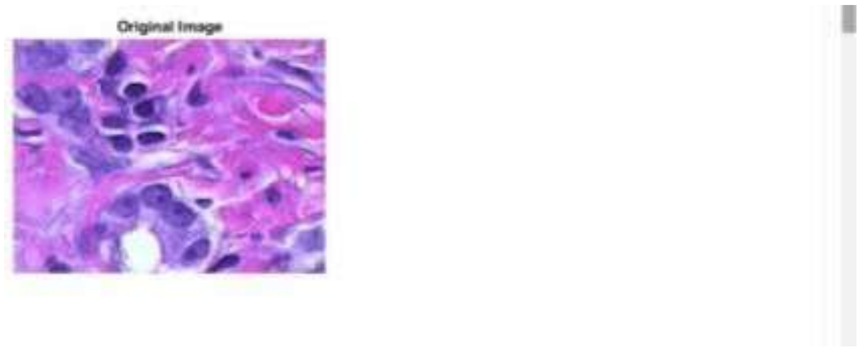
- Control the Amount of Sharpening at the Edges. Read an image into the workspace and display it.

```
a = imread('rice.png');
imshow(a)
title('Original Image');
```

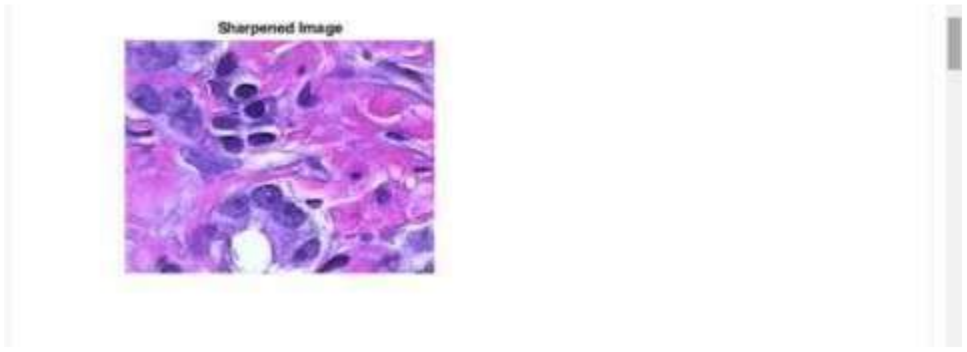
- Sharpen image, specifying the radius and amount parameters.

```
b = imsharpen(a,'Radius',2,'Amount',1);
figure, imshow(b) title('Sharpened Image');
```

Sample Input



Output:



Exercise No. 4 Data Visualization

Visualize dataset using plotting methods.

Objectives

To visualize experimental data in a clear and understandable manner using plots, charts, and graphs.

Concept

Various types of plots you can create to visualize different aspects of your data.

Line Plot: Visualize the relationship between two variables with a continuous line

Scatter Plot: Display the relationship between two variables using points.

Bar Plot: Compare different categories of data by displaying bars.

Histogram: Display the distribution of data over specified bins.

Pie Chart: Show the proportion of parts to the whole.

Procedure

Scatter Plots

The scatter function draws a scatter plot of x and y values.

- load patients Height Weight Systolic

```
scatter(Height,Weight)
xlabel('Height') ylabel('Weight')
```

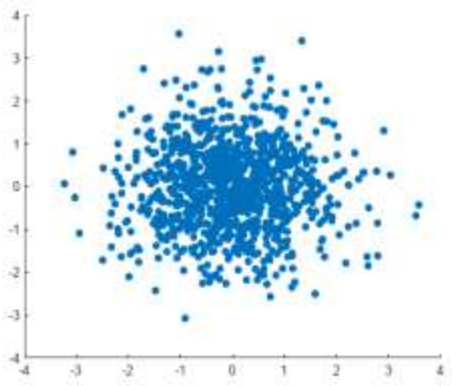
Box Plots

- Load the sample data.
  ```
  load carsmall
  ```
- Create a box plot of the miles per gallon (MPG) measurements. Add a titleand label the axes.
  ```
  boxplot(MPG)
  xlabel('All Vehicles')
  ylabel('Miles per Gallon (MPG)')
  title('Miles per Gallon for All Vehicles')
  ```

Sample Input

Scatter Plot: x = randn(1000,1);
y = randn(1000,1)

Output:



Exercise No. 5   Image Morphology Operations

Implementation of image morphology techniques.

Objectives

To analyze and represent the shapes and structures within an image. This includes detecting boundaries, corners, and regions of interest based on their geometric properties.

Concept

Erosion: Erosion is used to erode away the boundaries of foreground objects (usually white pixels in a binary image) by using a structuring element.

Dilation: Dilation expands the boundaries of foreground objects and fills in gaps between them using a structuring element.

Opening: Opening is a combination of erosion followed by dilation. It removes small objects (noise) from the foreground of an image.

Closing: Closing is a combination of dilation followed by erosion. It is used to close small holes or gaps in the foreground of an image.

Procedure

- Read an image into the workspace using imread( )
- Display the image using imshow( )
- Select the morphology functions

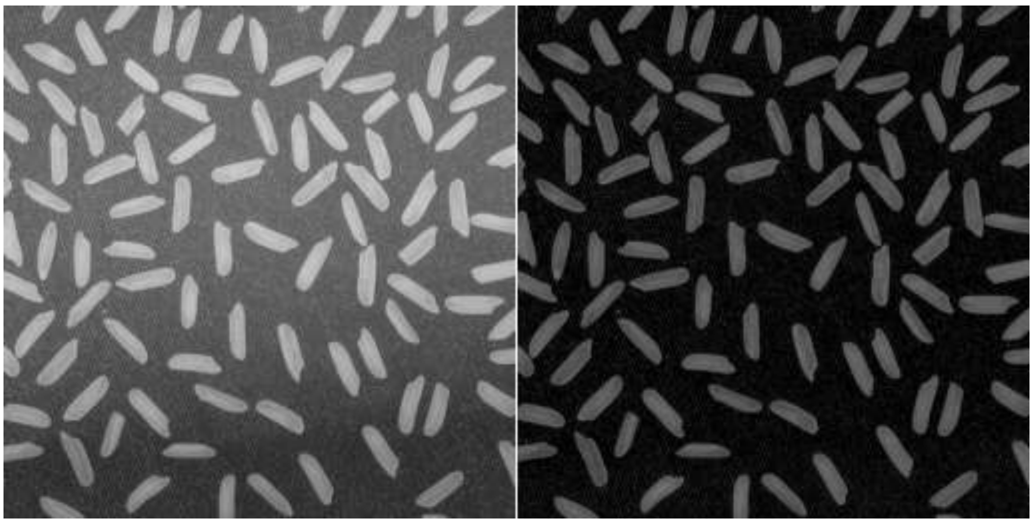
  imopen ()- Perform morphological opening imclose () - Perform morphological closing

  imerode ()- Perform the erosion operation Imdilate () - Dilates the image

- Apply morphology functions on the images

```
# Importing the image
  I = imread("D:\Rice.jpg"); subplot(2, 3, 1), imshow(I);
  title("Original image");


   % Dilated Image
    se = strel("line", 7, 7); dilate = imdilate(I, se); subplot(2, 3, 2), imshow(dilate);
    title("Dilated image");


    % Eroded image
     erode = imerode(I, se); subplot(2, 3, 3), imshow(erode); title("Eroded image");
    % Opened image open = imopen(I, se); subplot(2, 3, 4), imshow(open);
    title("Opened image");
    % Closed image close = imclose(I, se); subplot(2, 3, 5), imshow(close);
     title("Closed image");
```

Sample Input and Output



Exercise No. 6 Color image Processing

Implementation of color image processing.

Objectives

To convert images between various color representations, such as RGB, grayscale, and HSV

Concept

Color conversion involves transforming images from one color representation to another. This process is essential for various image processing tasks, such as adjusting colors, enhancing images, or extracting specific features based on color information.

RGB to Grayscale Conversion: To convert an RGB color image to grayscale, you can use the rgb2gray function.

RGB to HSV Conversion: To convert an RGB color image to HSV (Hue, Saturation, Value) color space, use the rgb2hsv function.

Grayscale to RGB Conversion: To convert a grayscale image to RGB format

Procedure

- Read an image into the workspace using imread( )

  I = imread('lenna.png');

- Count the rows and columns in the image r = size(I, 1);

  c = size(I, 2);

- creating zero matrices R = zeros(r, c, 3);

  G = zeros(r, c, 3);

  B = zeros(r, c, 3);

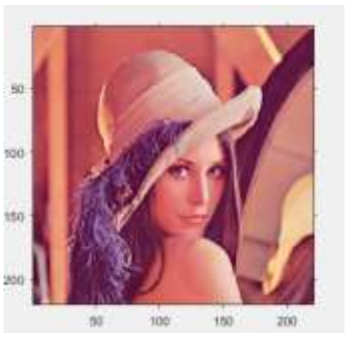
- storing the corresponding color plane

```
% red plane
 R(:, :, 1) = I(:, :, 1);
% green plane
G(:, :, 2) = I(:, :, 2);
% blue plane
B(:, :, 3) = I(:, :, 3);
```

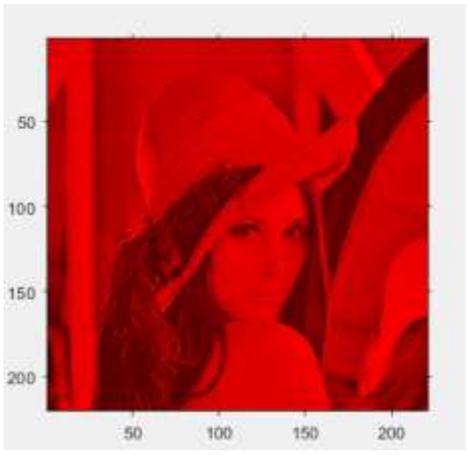
- displaying the images

```
figure, imshow(uint8(R));
figure, imshow(uint8(G));
figure, imshow(uint8(B)); (J,64)
```

Sample Input



Output:



Exercise No. 7 Image Segmentation using Thresholding

Implementation of thresholding methods in medical image segmentation.

Objectives

To segment an image into different regions and Separate the region of interest from the background.

Concept

Image segmentation using thresholding is a common technique to partition an image into distinct regions. Thresholding can be used to segment objects from the background based on the intensity values of the pixels.

Global Thresholding: Use a fixed threshold value to segment the image.

Otsu's Method: Use Otsu's method to automatically determine an optimal threshold value.

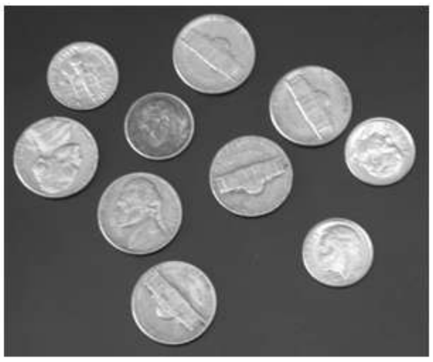
Adaptive thresholding is a technique used to segment an image in cases where lighting conditions vary across the image.

Procedure

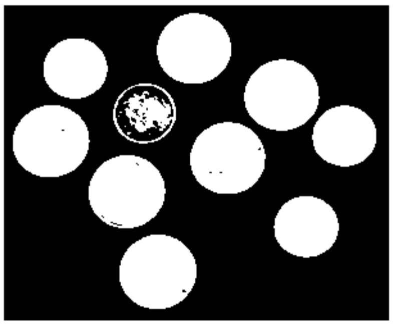
- Read image and display it
  ```
   I = imread('sample.png');
  imshow(I)
  ```
- Calculate a single threshold value for the image.
  ```
   level = multithresh(I);
  ```
- Segment the image into two regions using imquantize , specifying the threshold level returned by multithresh .

  ```
  seg_I = imquantize(I,level);
  
  figure imshow(seg_I,[])
  ```

Input:



Output:



Exercise No. 8   Image Segmentation using Clustering

Implementation of clustering methods in medical image segmentation.

Objectives

To segment an image into different regions and Separate the region of interest from the background based on certain features or characteristics.

Concept

Clustering is a way to separate groups of objects. K-means clustering treats each object as having a location in space. It finds partitions such that objects within each cluster are as close to each other as possible, and as far from objects in other clusters as possible.

Procedure

- Read Image

Read in hestain.png, which is an image of tissue stained with hemotoxylin and eosin (H&E). This staining method helps pathologists distinguish between tissue types that are stained blue-purple and pink.

he = imread("hestain.png"); imshow(he), title('H&E image'); text(size(he,2),size(he,1)+15, ...

"Image courtesy of Alan Partin, Johns Hopkins University", ... "FontSize",7,"HorizontalAlignment","right");

- Classify Colors in RBG Color Space Using K-Means Clustering numColors = 3;

L = imsegkmeans(he,numColors); B = labeloverlay(he,L); imshow(B)

title("Labeled Image RGB")

- Convert Image from RGB Color Space to L*a*b* Color Space lab_he = rgb2lab(he);
- Classify Colors in a*b* Space Using K-Means Clustering ab = lab_he(:,:,2:3);

ab = im2single(ab);

pixel_labels = imsegkmeans(ab,numColors,"NumAttempts",3) B2 = labeloverlay(he,pixel_labels);

imshow(B2)

title("Labeled Image a*b*")

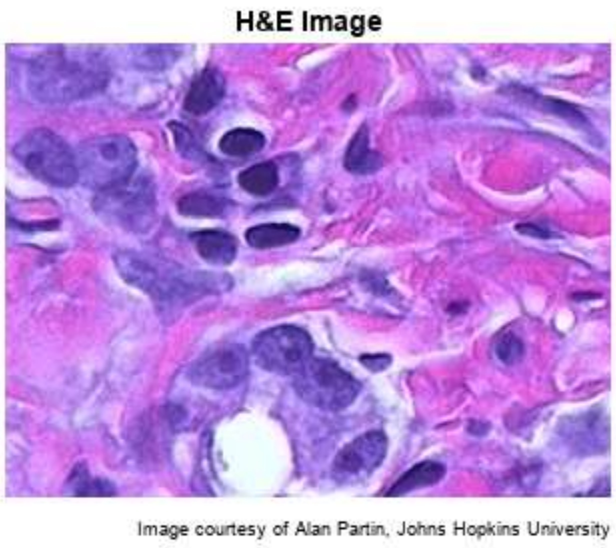
- Create Images that Segment H&E Image by Color

mask1 = pixel_labels == 1; cluster1 = he.*uint8(mask1); imshow(cluster1) title("Objects in Cluster 1");

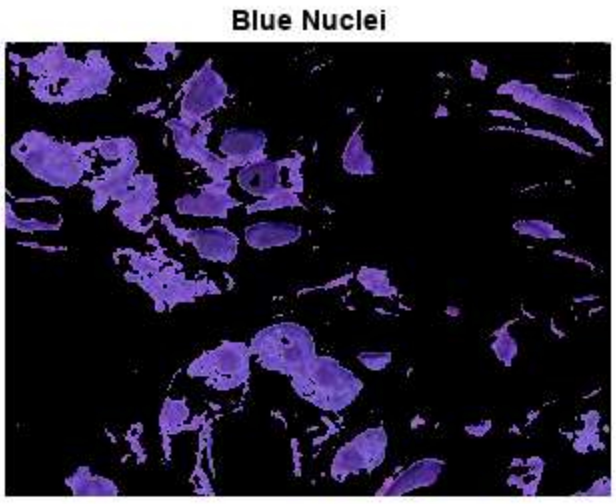
- Segment Nuclei L = lab_he(:,:,1);

```
L_blue = L.*double(mask3); L_blue = rescale(L_blue);
idx_light_blue = imbinarize(nonzeros(L_blue));
```

Input:



Output:



Exercise No. 9 Image Edge Detection

Implementation of edge detection methods in object identification.

Objectives

To find objects boundaries using different edge detection filters such as methods such as Prewitte, Sobel, and Robert .

Concept

Edge detection is a fundamental task in image processing and computer vision, aimed at identifying boundaries within images where abrupt changes in intensity occur. These boundaries often correspond to object boundaries or significant features in the image.

Gradient-Based Methods:

Sobel Operator: Uses discrete differentiation masks to calculate the gradient magnitude and direction. It emphasizes edges in both horizontal and vertical directions.

Prewitt Operator: Similar to Sobel but with slightly different masks, also used for detecting edges in horizontal and vertical directions.

Roberts Cross Operator: Uses a pair of 2x2 convolution masks to detect edges at 45-degree angles.

Procedure

- Read Image

a = imread('D:\Rice.jpg');

- Perform image color conversion and display it b = rgb2gray(a);

subplot(2,2,1); imshow(a);

title('Original Image');

- Apply the edge detection filters such as Prewitte, Sobel, Robert and display the images

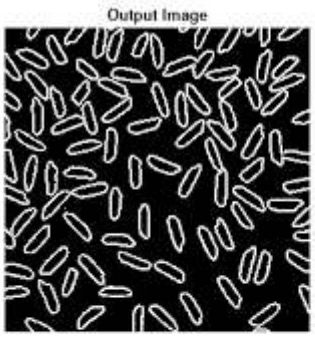
c1 = edge(b,'sobel');

subplot(2,2,2); imshow(c1); title('Sobel Operator'); c2 = edge(b,'prewitt'); subplot(2,2,3); imshow(c2);

title('Prewitt Operator'); c3 = edge(b,'roberts'); subplot(2,2,4); imshow(c3); title('Roberts Operator');

Input:

Output:



Exercise No. 10   Statistical Features Extraction

Computation of statistical features such as mean, standard Deviation, correlation coefficient of the given Image

Objectives

To extract various properties and characteristics of an image using statistical methods. These features can be used for image analysis tasks such as classification, segmentation, object recognition, and more.

Concept

Quantify Image Characteristics:

Texture: Capture the texture patterns and granularity of an image using measures like contrast, entropy, and energy.

Intensity: Analyze the distribution of pixel intensities with features like mean, variance, skewness, and kurtosis.

GLCM: GLCM (Gray-Level Co-occurrence Matrix) image features are statistical measures derived from the GLCM, which quantifies the joint occurrence of pairs of pixel intensity values at specified offsets in an image. GLCM-based features are widely used in texture analysis and can provide valuable information about the spatial distribution of pixel intensities within an image.

Procedure

- Reading an Image

```
a = imread('D:\Sample.jpg');
```

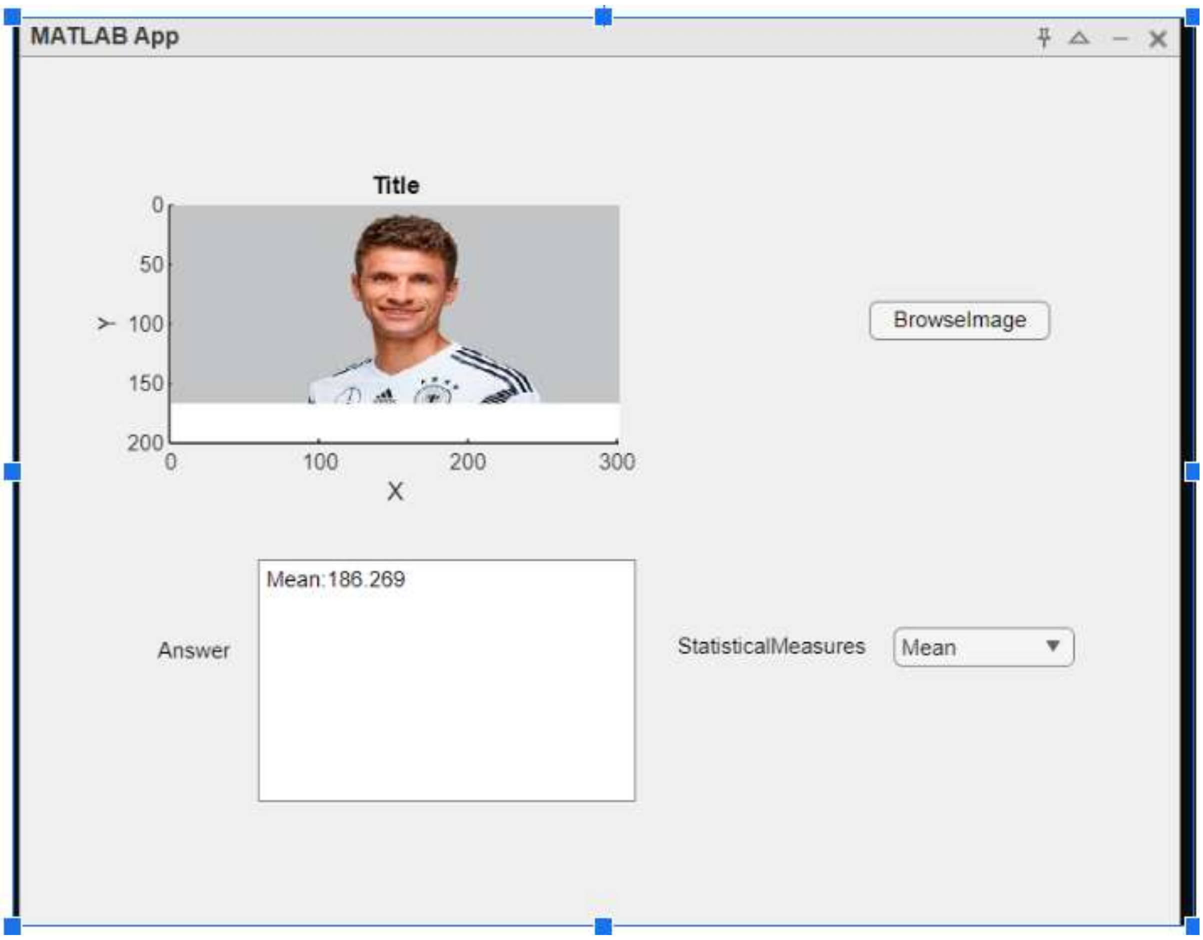
- Display an Image

To display image, use the imshow function. imshow(A)

- Compute statistical features such as mean, standard deviation and correlation coefficient from an image using the standard statistics of an image using

the mean2, std2, and corr2 functions.

Input & Output:



Mean:186.269

Exercise No. 11 GUI for fruit/vegetable classification

Develop a GUI for fruit/vegetable classification.

Objectives

To develop GUI for fruit/vegetable classification using statistical features and artificial neural network.

Concept

Graphical user interfaces (GUIs), also known as apps, provide point-and-click control of your software applications. You can share apps both for use within MATLAB and also as standalone desktop or web apps.

You can choose from the following three ways to create an app in MATLAB:

Convert a script into a simple app: Choose this option when you want to share a script with students or colleagues and allow them to modify variables using interactive controls.

Create an app interactively: Choose this option when you want to create a more sophisticated app using a drag-and-drop environment to build the user interface.

Create an app programmatically: Choose this option when you want to create an app's user interface by writing the code yourself.

Procedure

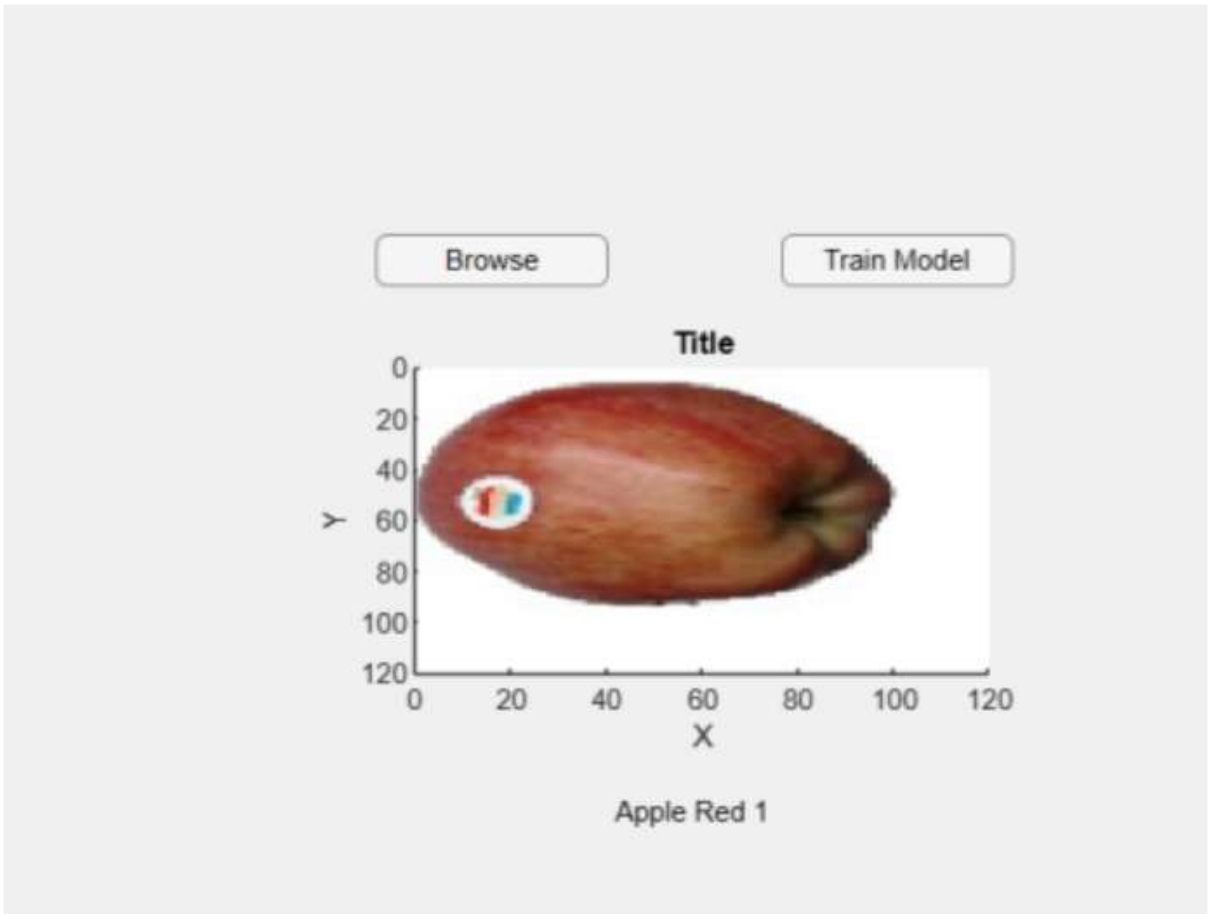
- Design the GUI layout: Create a layout with buttons and axes for loading images, displaying results, and training the ANN.

- Load and preprocess the image: Add functionality to load images and preprocess them before feeding them to the ANN.

- Train the ANN: Add functionality to train the ANN with a dataset.

- Classify the image: Add functionality to classify the loaded image using the trained ANN.

Input & Output:



Apple Red 1

Exercise No. 12   Image Classification using Deep Learning Techniques

Implementation of deep learning techniques for image classification.

Objectives

To automatically categorize images into predefined classes using deep neural networks such as convolutional neural networks (CNNs).

Concept

Deep Learning Image Classification:

- Feature Extraction:

Automatically learn and extract features from images without manual intervention. This includes low-level features like edges and textures, as well as high-level features like shapes and patterns.

- Pattern Recognition:

Recognize and classify complex patterns in images that may be difficult to capture with traditional machine learning techniques.

- Scalability:

Handle large datasets and generalize well to new, unseen images.

- Accuracy:

Achieve high classification accuracy by effectively learning from data and minimizing prediction errors.

- Automation:

Reduce the need for manual feature engineering and domain-specific expertise.

Procedure

Create Simple Image Classification Network

- Load image data.

- Load the digit sample data as an image datastore. The imageDatastore function automatically labels the images based on folder names.

digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ... 'nndatasets','DigitDataset');

imds = imageDatastore(digitDatasetPath, ... 'IncludeSubfolders',true, ... 'LabelSource','foldernames');

- Divide the data into training and validation data sets, so that each category in the training set contains 750 images, and the validation set contains the remaining images from each label. splitEachLabel splits the image datastore into two new datastores for training and validation.

numTrainFiles = 750;

[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

- Define Network Architecture

Define the convolutional neural network architecture. Specify the size of the images in the input layer of the network and the number of classes in the fully connected layer before the classification layer. Each image is 28-by-28-by-1 pixels and there are 10 classes.

inputSize = [28 28 1];

numClasses = 10;

layers = [imageInputLayer(inputSize) convolution2dLayer(5,20) batchNormalizationLayer reluLayer fullyConnectedLayer(numClasses) softmaxLayer classificationLayer];

For more information about deep learning layers, see List of Deep Learning Layers.

- Train Network

Specify the training options and train the network.

By default, trainNetwork uses a GPU if one is available, otherwise, it uses a CPU. Training on a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For information on supported devices, see GPU Support by Release (Parallel Computing Toolbox). You can also specify the execution environment by using the 'ExecutionEnvironment' name-value pair argument of trainingOptions.

options = trainingOptions('sgdm', ... 'MaxEpochs',4, ... 'ValidationData',imdsValidation, ... 'ValidationFrequency',30, ...

'Verbose',false, ... 'Plots','training-progress');

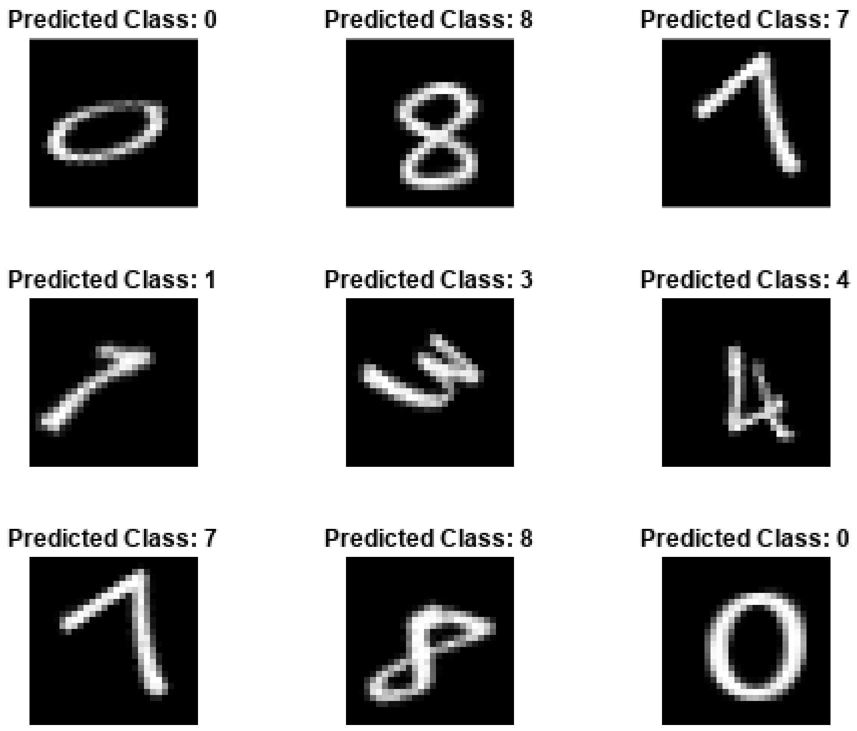
net = trainNetwork(imdsTrain,layers,options);

- Test Network

Classify the validation data and calculate the classification accuracy. YPred = classify(net,imdsValidation);

YValidation = imdsValidation.Labels; accuracy = mean(YPred == YValidation) accuracy = 0.9892

Input & Output:


Predicted Class: 0
Predicted Class: 8
Predicted Class: 7
Predicted Class: 1
Predicted Class: 3
Predicted Class: 4
Predicted Class: 7
Predicted Class: 8
Predicted Class: 0