MATLAB Installation steps.

1. Create MathWorks account using college email id([ch.en.u4\*\*\*\*@ch.students.amrita.edu](mailto:ch.en.u4****@ch.students.amrita.edu))
2. Associate your MathWorks account to the license.
3. We can download MATLAB and related products for use on personal devices using the AMRITA VISHWA VIDYAPEETHAM MATLAB PORTAL.( <https://in.mathworks.com/academia/tah-portal/amrita-vishwa-vidyapeetham-1061757.html>
4. Download the Software. (Latest Version Available)
5. Install the software.
6. Activate the Software.

**Step 1: Create MathWorks Account.**

Go to <https://www.mathworks.com/accesslogin/createProfile.do>

NOTE: • Remember your Email and Password information for later use

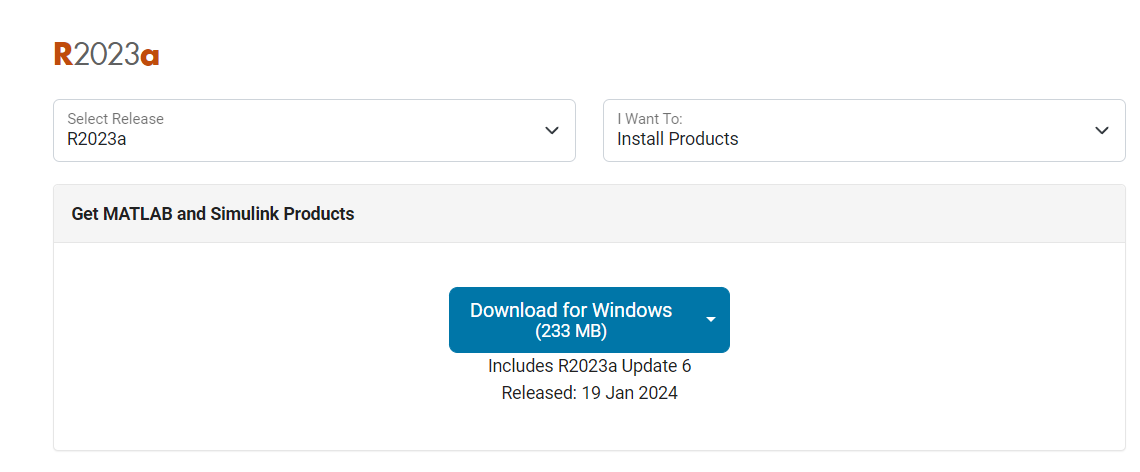
• For ‘**How will you use MathWorks software?**’, select “Academic use”

**Step 2: Associate License**

By logging with college email – id college license will be activated to your account directly.

**Step 3: Download the Software.**

1. Go to https://in.mathworks.com/products/matlab.html
2. Click on the >> Get MATLAB link
3. It will redirect to your Account page. There in the right corner click >> INSTALL MATLAB.
4. By clicking Install MATLAB it will redirect to the page below, to select the version you want to install and your device preference.
5. Automatically the download will start as (.exe) file.

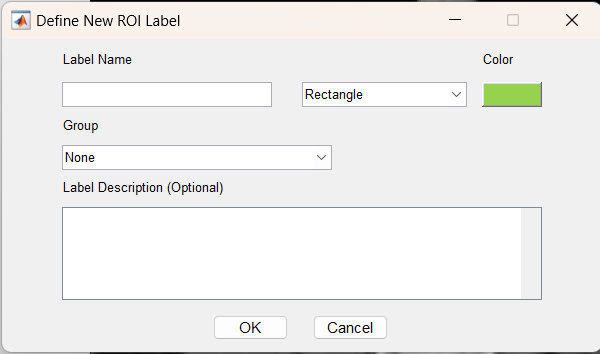


**Step 4: Install the Software.**

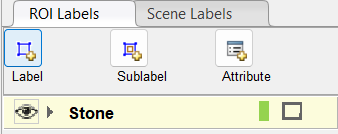
1. Next it will prompt you select an installation Folder
2. At product selection window don’t change anything and click Next, and check the box for shortcut on to the desktop. (if they ask).
3. Verify the installation and click enter.
4. The process will take about 20-30 minutes.
5. Click finish.
6. MATLAB will be successfully installed to your system.

Before code we should prepare our dataset. You can download the dataset form Kaggle or we can get the real time data set. After installing data set, we should make that to work with our code. For that we should do the following steps:

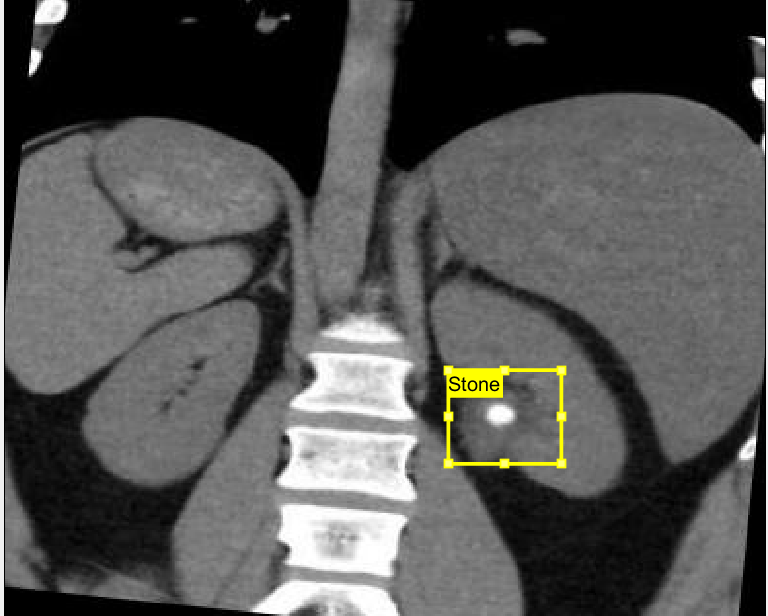
1. Open the image Labeler app from the app session. (If not available install the deep-learning, computer vision tool boxes from the add-ons section).
2. After opening the Labeler app ther we can see import option. Select from files in that drop down.
3. Select the data set you have downloaded. And select open. Whole images from the data set will be displayed in the app.
4. There we can see a option named label from ROI labels. Select that label option.



1. Above dialogue box will appear. Enter the label name (in my case it is stone, we can enter what object we are working with).
2. If we have single label, we can select none from the Group option. If we have multiple labels, we should select New group to create separate label.
3. By clicking ok we can proceed for further steps.



1. We can see label has been created. Now we should draw a rectangle box over our object from the images.



1. Label each and every image in the same way as shown in the above image. If we have multiple also, we should do the same process.
2. After all the labelling, select export option form the top ribbon. Select to workspace It will display a dialogue box and in that select as table. There we can select variable name for that.
3. It will be exported to the workspace in the form of table in which 1st column we will have the path of the image, and in the 2nd column we will have the bounding box details for that particular label.
4. We should convert the table file to .mat file for using that in our code. For that drag and drop that table from workspace to the folder you are working and name that file. It will save as .mat file in the folder.

WORKING OF CODE.

data = load('file\_name.mat');

variable\_name = data.variable\_name;

1. Load the .mat file saved in the folder as data and save data in the name of variable name saved during the step 10 above. While exporting to the workspace.

rng(0)

shuffledIndices = randperm(height(variable\_name));

idx = floor(0.6 \* height(variable\_name));

trainingIdx = 1:idx;

trainingDataTbl = variable\_name (shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.1 \* length(shuffledIndices) );

validationDataTbl = variable\_name (shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);

testDataTbl = variable\_name (shuffledIndices(testIdx),:);

imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});

bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));

imdsValidation = imageDatastore(validationDataTbl{:,'imageFilename'});

bldsValidation = boxLabelDatastore(validationDataTbl(:,2:end));

imdsTest = imageDatastore(testDataTbl{:,'imageFilename'});

bldsTest = boxLabelDatastore(testDataTbl(:,2:end));

trainingData = combine(imdsTrain,bldsTrain);

validationData = combine(imdsValidation,bldsValidation);

testData = combine(imdsTest,bldsTest);

1. If we run this section, it will create 60% data for training 20% for testing and 20% for validation. Basically we have imds in the sense (image data store). Which will store the image paths and images, and blds (box label data store) which stores the bounding box details for the labels. We should combine both and and we should prepare the training data, testing data, and validation data.

data = read(trainingData);

I = data{1};

bbox = data{2};

annotatedImage = insertShape(I,'Rectangle',bbox);

annotatedImage = imresize(annotatedImage,2);

figure

imshow(annotatedImage)

1. The above section gives an image result displaying the bounding box on the image.

inputSize = [391 320 3];

preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));

numAnchors = 5;

anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)

featureExtractionNetwork = resnet50;

featureLayer = 'activation\_40\_relu';

numClasses = 3;

lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);

1. From the above section of code the important components like what is the backbone network, and what is the feature extraction layer that can be used for the backbone all these information will be loaded.

And faster r cnn network will be created using the function fasterRCNNLayers, which include the size of image,number of classes we are working with, anchor boxes, backbone network(featureExtraction network) and the feature layer. By combining all these a neural network will be created. This network can be visualized by using deep network designer. Open deep network designer, there we can see NEW option, by clicking we can see a dialogue box showing to import pre-existing model or from work space. Select work space because we have already created and it was stored in workspace as lgraph. Open it we can see neural network design. If we want to change the backbone network we can replace the resnet-50 with interested or with which we want to work with like (vgg16, inception, mobilenet etc) can be used as backbone. For each network there will be different feature layer.

|  |  |
| --- | --- |
| VGG-16 | ‘relu5\_3’ |
| DenseNet201 | relu5\_blk |
| MobileNetV2 | 'block\_13\_expand\_relu' or 'out\_relu' |
| Inception | ‘mixed7’ |

augmentedTrainingData = transform(trainingData,@augmentData);

augmentedData = cell(4,1);

for k = 1:4

data = read(augmentedTrainingData);

augmentedData{k} = insertShape(data{1},'rectangle',data{2});

reset(augmentedTrainingData);

end

figure

montage(augmentedData,'BorderSize',30)

1. In this section that data we took for training that will be augmented like flipping, rotating etc. And that result will be displayed as an images showing the bounding box for flipped, rotated images.

trainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));

validationData = transform(validationData,@(data)preprocessData(data,inputSize));

1. In this we will prepare the training data for training the model, for which we will transform both augmented data and pre processed data. In this we will used augmented and pre processed data.

options = trainingOptions('sgdm',...

'MaxEpochs',20,...

'MiniBatchSize',2,...

'InitialLearnRate',1e-3,...

'Plots','training-progress',...

'ExecutionEnvironment','auto');

1. In this section for training, we need some options for training the model like epochs, batch size, learning rate, etc. By changing these values, we can see changes in the execution time and also in the results. As the number of epochs increases the execution time also increases, batch size we should assign based on the memory of our system. Mostly prefer 2,4,8,16,32 in these. For our application 2 and 4 are preferred.

detector = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...

'NegativeOverlapRange',[0 0.3],...

'PositiveOverlapRange',[0.6 1]);

1. By using trainFasterRCNNObjectDetector we can train our detector using training data and the network we have created, and with th training options.
2. NegativeOverlapRange which specifies the threshold for bounding boxes to overlap. If we have bounding box overlap region from 0 to 0.3 that is considered as negative region and same with positive overlap range.
3. By this our detector will be created for detection.

I = imread(testDataTbl.imageFilename{65});

I = imresize(I,inputSize(1:2));

[bboxes,scores,labels] = detect(detector,I);

annotations=string(labels) +":" +string(scores);

I = insertObjectAnnotation(I,'Rectangle',bboxes,cellstr(annotations));

% Display the image with the title

figure

imshow(I)

1. In this we will run our detector over the test image for verifying the working of detector. And displaying the results with detection. We can change the number with any number within the testData table for testing images.

testData = transform(testData,@(data)preprocessData(data,inputSize));

%%

detectionResults = detect(detector,testData,...

Threshold= 0.1,...

MiniBatchSize=2);

%%

classID = 3;

metrics = evaluateObjectDetection(detectionResults,testData);

precision = metrics.ClassMetrics.Precision{classID};

recall = metrics.ClassMetrics.Recall{classID};

figure;

plot(recall, precision);

grid on

title(sprintf("Average Precision = %.2f", metrics.ClassMetrics.mAP(classID)))

xlabel('Recall');

ylabel('Precission');

1. Evaluate the trained object detector on a large set of images to measure the performance. Computer Vision Toolbox™ provides an object detector evaluation function (evaluateObjectDetection) to measure common metrics such as average precision. We use the average precision metric to evaluate performance. The average precision provides a single number that incorporates the ability of the detector to make correct classifications (precision) and the ability of the detector to find all relevant objects (recall).
2. Run the detector on all the test images. Set the detection threshold to a low value to detect as many objects as possible. This helps you evaluate the detector precision across the full range of recall values.
3. Evaluate the object detector using the average precision metric.
4. The precision-recall (PR) curve highlights how precise a detector is at varying levels of recall. The ideal precision is 1 at all recall levels. The use of more data can help improve the average precision but might require more training time. Plot the PR curve.

image = I;

% Detect the object and get bounding box coordinates (assuming you have them)

bbox1 = bboxes; % Adjust according to your bounding box

border\_width = 5; % Adjust the border width as needed

bbox\_adjusted = [bbox1(1) + border\_width, bbox1(2) + border\_width, bbox1(3) - 2\*border\_width, bbox1(4) - 2\*border\_width];

% Extract the region of interest (ROI)

roi = imcrop(image, bbox\_adjusted);

% Convert the ROI to grayscale if necessary

if size(roi, 3) == 3

roi\_gray = rgb2gray(roi);

else

roi\_gray = roi;

end

% Convert the grayscale ROI to a binary image using thresholding

threshold = graythresh(roi\_gray);

binary\_roi = imbinarize(roi\_gray, threshold);

% Find the number of white pixels in each column

white\_pixels\_per\_column = sum(binary\_roi, 1);

% Find the column index with the maximum number of white pixels

[max\_white\_pixels\_column\_count, max\_white\_pixels\_column\_index] = max(white\_pixels\_per\_column);

% Find the number of white pixels in each row

white\_pixels\_per\_row = sum(binary\_roi, 2);

% Find the row index with the maximum number of white pixels

[max\_white\_pixels\_row\_count, max\_white\_pixels\_row\_index] = max(white\_pixels\_per\_row);

num\_white\_pixels = sum(binary\_roi(:) == 1);

% Assuming you have physical dimensions (pixels per millimeter)

pixels\_per\_mm = 0.264; % Adjust according to your actual data

% Convert size of all white pixels from pixels to millimeters

area\_white\_pixels\_mm2 = num\_white\_pixels\*pixels\_per\_mm;

heigth = max\_white\_pixels\_column\_count\*pixels\_per\_mm;

width= max\_white\_pixels\_row\_count\*pixels\_per\_mm;

% Display the number of white pixels and the size of those pixels in millimeters

disp(['Number of white pixels in the binary ROI: ' num2str(num\_white\_pixels)]);

disp(['Area of all white pixels in the binary ROI (mm): ' num2str(area\_white\_pixels\_mm2)]);

% Display results

disp(['Column with max white pixels: ', num2str(max\_white\_pixels\_column\_index)]);

disp(['Count of white pixels in max column: ', num2str(max\_white\_pixels\_column\_count)]);

disp(['Row with max white pixels: ', num2str(max\_white\_pixels\_row\_index)]);

disp(['Count of white pixels in max row: ', num2str(max\_white\_pixels\_row\_count)]);

disp(['The height of the stone in mm : ', num2str(heigth)]);

disp(['The width of the stone in mm: ', num2str(width)]);

1. This section is for finding the size of the detected stone from the image.
2. First take the output image I as input for this section.
3. For output detected image we will have bounding box with object detected. Extract those bounding box co-ordinates from the output image and store in the variable bbox1.
4. We should crop that bounding box region. Using that bounding box values. We will use imcrop function for that. By doing this directly we will get bounding box colour edges in the cropped image. Which will give false rsults. So we should adjust the border of the cropped image by excluding the border with some vale. Assume it as 3 or 4 or 5. By this it will give the adjusted cropped image with bounding box excluded.
5. The region of interest will be extracted using the imcrop function with the adjusted bounding box co-ordinates.
6. Convert the cropped image to grayscale image.
7. Convert the grayscale image to binary image with some threshold. The threshold will be assigned based on the image by the method Otsus using the function greythresh.
8. Now calculate number of white pixels in the binary image. Why only white pixels, the white pixels represent the stone. So, we will calculate number of white pixels.
9. Calculating the size of the pixel, for this we should know the resolution of the image. The dataset we took is of 96 DPI (dots per inch). Based on the resolution the pixel value changes.
10. Calculating the pixel size

*Dpi is the pixel intensity or dots per inch.*

*96 dpi means there are 96 pixels per inch.*

*1 inch is equal to 25.4 millimetres.*

*1 inch = 25.4 mm*

*Dpi = 96 px / in*

*96 px / 25.4 mm*

*Therefore, one pixel is equal to*

*1 px = 25.4 / 96*

*1 px = 0.2645833 mm*

1 pixel is 0.2645833 millimetres.

1. By multiplying the number of white pixels with the size of single pixel we will get the area of total white pixels. Which is size of the stone.
2. For calculating the length of the stone and width of the stone. For length if we calculate max number of white pixels present in the column and for width max number of white pixels in particular row shows the width of the stone.

% Display the original image and ROI

figure;

imshow(image);

hold on;

rectangle('Position', bbox1, 'EdgeColor', 'r', 'LineWidth', 2);

hold off;

% Display the ROI

figure;

imshow(roi);

title('Region of Interest');

figure;

imshow(roi\_gray);

title('ROI- GRAYSCALE')

% Display the binary ROI

figure;

imshow(binary\_roi);

title('Binary ROI');

figure;

subplot(1,2,1);

imshow(roi\_gray);

subplot(1,2,2);

imshow(binary\_roi);

1. At last display the results.