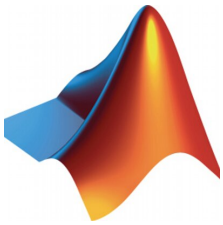
INM460 / IN3060 Computer Vision

In this lab we learn how to perform classification on images using support vector machines and cascading classifiers.



Resources:

The lab will make use of Matlab, and the following data file in the Lab materials 06 folder on Moodle:

1. Stop_Sign.jpg

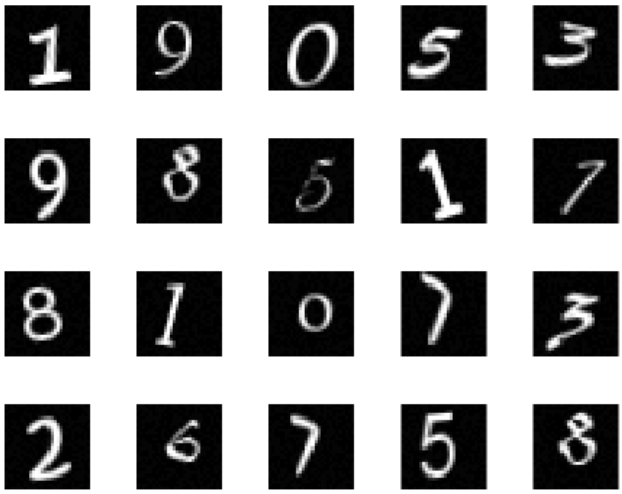
We will also use the following data file in the Lab materials 03 folder:

2. Class.jpg

Task 1: Training a Support Vector Machine (SVM) for Digit Classification

In today's lecture we have described support vector machines, which are a very powerful classification algorithm that can be used for image classification. Let's train them on a dataset of digits and let's see how they perform.

You can load synthetic images which have been generated by applying random affine transformations to digit images, using different fonts. These images are part of Matlab and you do not need to download them.



Each digit image is 28-by-28 pixels, and there are 5,000 training examples. You can load the training data, and view some of the images.

```
% Load the training data into memory
[xTrainImages, tTrain] = digittrain_dataset;
% Display some of the training images
clf
for i = 1:20
    subplot(4,5,i);
    imshow(xTrainImages{i});
end
```

Here we convert the training images into vectors and put them in a matrix. Note that this means that we are directly using pixel intensities as feature descriptors, which is not the best solution, especially for images. We will go more into details about this aspect in the upcoming lectures.

```
% Get the number of pixels in each image
imageWidth = 28;
imageHeight = 28;
inputSize = imageWidth*imageHeight;  % Load the test images

% Turn the training images into vectors and put them in a matrix
xTrain = zeros(inputSize,numel(xTrainImages));
for i = 1:numel(xTrainImages)
    xTrain(:,i) = xTrainImages{i}(:);
end
```

Here we convert the test images into vectors and put them in a matrix.

```
[xTestImages, tTest] = digittest_dataset;
% Turn the test images into vectors and put them in a matrix
xTest = zeros(inputSize,numel(xTestImages));
for i = 1:numel(xTestImages)
    xTest(:,i) = xTestImages{i}(:);
end
```

We can change the format of the labels to 1-10 for simplicity in viewing the labels:

```
for i = 1 : 5000
    tTrainLabels(i) = uint8(find(tTrain(:,i)));
    tTestLabels(i) = uint8(find(tTest(:,i)));
end
```

Then we need to define a multiclass support vector machine using our training data and their labels:

```
SVMMdl = fitcecoc(xTrain',tTrainLabels); %SVM MULTICLASS TRAINER
```

We can use function 'predict' to evaluate the performance of the classifier on the training or test images and look at the confusion matrix:

```
labelsOut = predict(SVMMdl,xTest');
ConfMatTest = confusionmat(tTestLabels,uint8(labelsOut));
```

```
Accuracy = 1 - (5000 - sum(diag(ConfMatTest))) / 5000
```

What accuracy do you get?

Task 2: Testing the Viola-Jones object detector

In today's lecture we have introduced the famous Viola-Jones object detector framework, which was originally used for face detection. Matlab has a specific function for this framework called `vision.CascadeObjectDetector` that comes with already trained models.

> Exercise 1: If you remember, in Lab tutorial 03 we had to blur the faces in the image 'Class.jpg' (still available in the Lab materials 03 on Moodle) and we did it by manually selecting the coordinates in the image. Repeat the same exercise, but this time use `vision.CascadeObjectDetector` and one of the available pre-trained models.

Task 3: Training our own model with the Viola-Jones object detector

The pre-trained models in Matlab can be useful, but in general we will usually need to train our own models to detect our objects of interest. In this section we will train an object detector for stop signs. We can use the 'stopSign.mat' file (which contains the information of the ROIs of 85 images containing stop sign. The actual images are already available in Matlab too). We will then create and train a cascade object detector model.

First load the stopSigns.mat file:

```
load('stopSigns.mat');
```

Add the images location to the Matlab path (this way we can access them without the full path):

```
imDir =
fullfile(matlabroot,'toolbox','vision','visiondata','stopSignImages')
; addpath(imDir);
```

Specify the folder for negative images:

```
negativeFolder =
fullfile(matlabroot,'toolbox','vision','visiondata','nonStopSigns');
```

Train a cascade object detector using HOG features (default):

```
trainCascadeObjectDetector('stopSignDetector.xml',data,negativeFolder
,'FalseAlarmRate',0.2,'NumCascadeStages',5);
```

Spend some time to understand the exact function of the parameters called by trainCascadeObjectDetector (for instance `'FalseAlarmRate'`), as these are what you would possibly have to tune for your own applications.

Now we use the newly trained classifier to detect a stop sign in an image:

```
detector = vision.CascadeObjectDetector('stopSignDetector.xml');
```

Read the test image:

```
img = imread('stopSignTest.jpg');
```

Detect a stop sign:

```
bbox = step(detector,img);
```

Insert bounding boxes and return marked image:

```
detectedImg = insertObjectAnnotation(img,'rectangle',bbox,'stop
sign');
```

Display the detected stop sign:

```
figure;
imshow(detectedImg);
```

Exercise 2: Download the image 'Stop_Sign.jpg' from the Lab materials 06 folder on Moodle and test the detector you just trained on that image. What is the reason for this behaviour? How would you fix the detection?

Task 4: Label Images for Classification Model Training

The Training Image Labeler provides an easy way to label positive samples that the `trainCascadeObjectDetector` function uses to create a cascade classifier. Using this app, you can:

- Interactively specify rectangular regions of interest (ROIs).
- Using the ROIs, you can detect objects of interest in target images with the `vision.CascadeObjectDetector` System object.

You can load multiple images at one time, draw ROIs, and then export the ROI information in the appropriate format for the `trainCascadeObjectDetector`. The labeler app supports all image data formats that the `trainCascadeObjectDetector` function uses.

Open the Apps tab, under **Image Processing and Computer Vision**, click the app icon.

You can alternatively use MATLAB command prompt:

```
trainingImageLabeler
```

You can add an unlimited number of images to the **Data Browser**. You can then select, remove, and create ROIs, and save your session. When you are done, you can export the ROI information to an XML file.

You can find more instructions on Matlab's website if the user interface is not easy to interpret: https://uk.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html