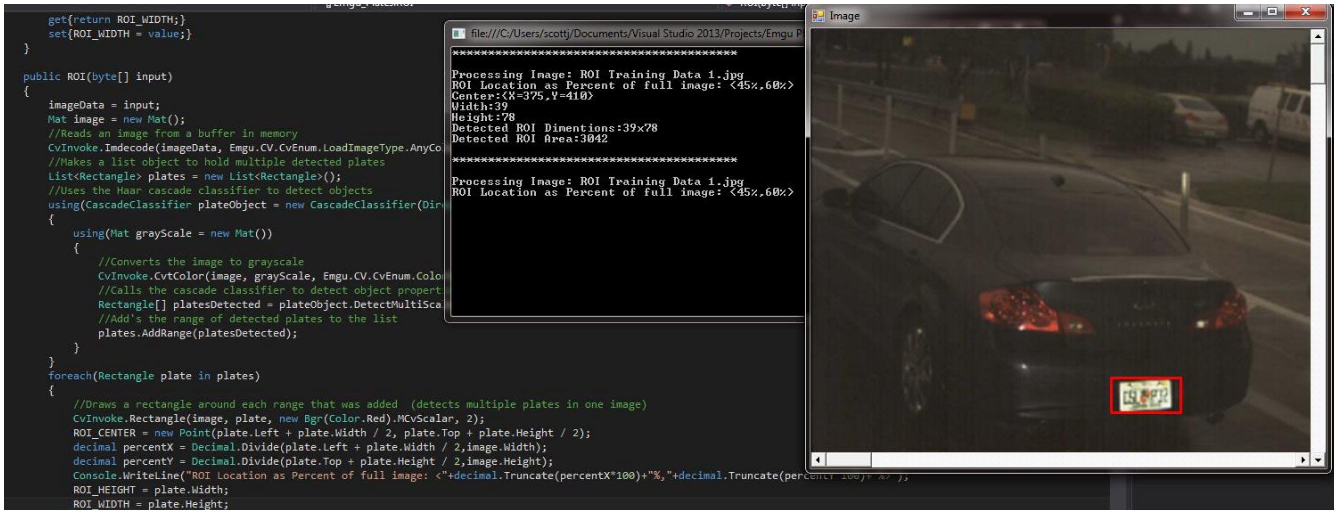
Tolls VES Auto Region of Interest Selection: Developed by Jamal D. Scott



```
        get{return ROI_WIDTH;}
        set{ROI_WIDTH = value;}
    }

    public ROI(byte[] input)
    {
        imageData = input;
        Mat image = new Mat();
        //Reads an image from a buffer in memory
        CvInvoke.Imdecode(imageData, Emgu.CV.CvEnum.LoadImageType.AnyCo
        //Makes a list object to hold multiple detected plates
        List<Rectangle> plates = new List<Rectangle>();
        //Uses the Haar cascade classifier to detect objects
        using(CascadeClassifier plateObject = new CascadeClassifier(Dir
        {
            using(Mat grayScale = new Mat())
            {
                //Converts the image to grayscale
                CvInvoke.CvtColor(image, grayScale, Emgu.CV.CvEnum.Colo
                //Calls the cascade classifier to detect object propert
                Rectangle[] platesDetected = plateObject.DetectMultiSca
                //Add's the range of detected plates to the list
                plates.AddRange(platesDetected);
            }
        }
        foreach(Rectangle plate in plates)
        {
            //Draws a rectangle around each range that was added  (detects multiple plates in one image)
            CvInvoke.Rectangle(image, plate, new Bgr(Color.Red).MCvScalar, 2);
            ROI_CENTER = new Point(plate.Left + plate.Width / 2, plate.Top + plate.Height / 2);
            decimal percentX = Decimal.Divide(plate.Left + plate.Width / 2,image.Width);
            decimal percentY = Decimal.Divide(plate.Top + plate.Height / 2,image.Height);
            Console.WriteLine("ROI Location as Percent of full image: <"+decimal.Truncate(percentX*100)+"%,"+decimal.Truncate(per
            ROI_HEIGHT = plate.Width;
            ROI_WIDTH = plate.Height;
```

```
file:///C:/Users/scottj/Documents/Visual Studio 2013/Projects/Emgu P
*********************************************

Processing Image: ROI Training Data 1.jpg
ROI Location as Percent of full image: <45%,60%>
Center:{X=375,Y=410}
Width:39
Height:78
Detected ROI Dimentions:39x78
Detected ROI Area:3042

*********************************************

Processing Image: ROI Training Data 1.jpg
ROI Location as Percent of full image: <45%,60%>
```

This write up will demonstrate how the Tolls VES auto ROI selection process works for C# Windows OS. For this process, you will need several tools that you can find online for free.

- Microsoft Visual Studio 2013 or higher.
- Emgu_CV 3.2.0
- Notepad++
- A Haar Cascade Classifier

For a copy of Microsoft's Visual Studio 2013 (or higher) you can find it on Microsoft's product download page at: https://www.visualstudio.com/

Emgu_CV is a C# wrapper class for OpenCV libraries which were originally written in C++; Emgu_CV allows OpenCV to operate within a C# environment. You can install the latest version through Visual Studio's NuGet package installer. We will later discuss how to accomplish this, but for now you can review Emgu_CV at: http://www.emgu.com/wiki/index.php/Main_Page

Notepad++ will be used to edit the properties of a program, you may already have it installed on your system, if you do not it can download it at this location: https://notepad-plus-plus.org/

The final piece is the Haar Cascade Classifier. The primary function of the cascade classifier is to note specific features of a specified object and save the average of those properties into an XML file. This file will then be referenced when called by the Cascade Classifier class. These properties are noted by comparing the features of many "positive" images to those of many "negative" images. Positive images are images solely of the object that you wish to detect where negative images are images that do not contain the object whatsoever or items similar to the object.

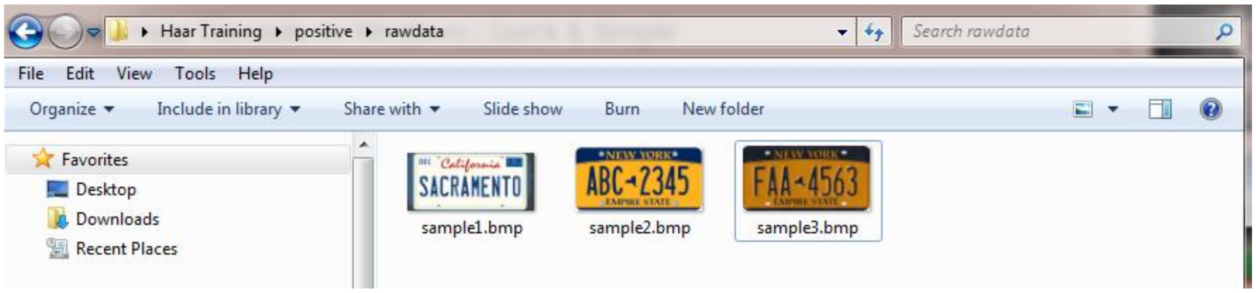
A Cascade Classifier can be trained in various ways, for the Tolls VES application, a prebuilt trainer was used; these files are packaged with this documentation. This application is a 3rd party application and was not developed with Tolls VES.

Training a classifier:

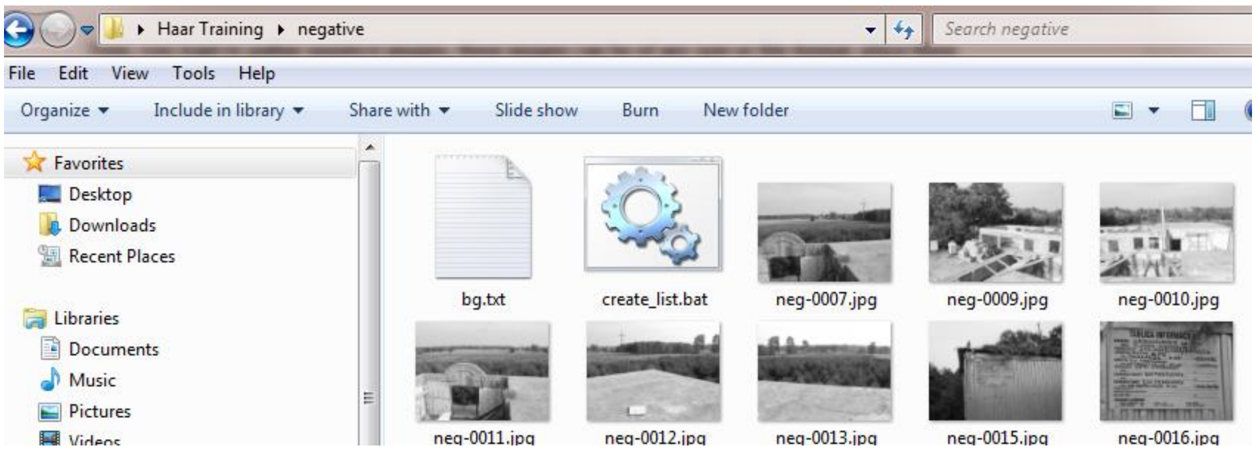
First, we need to generate the Haar Cascade Classifier XML file. This is achieved by running a training program on positive and negative images. For positive images, you can either capture your own, or do a simple Google search to find the desired images. This part is the most crucial as the types of images chosen to train the Cascade Classifier will greatly determine the success or failure of your application's abilities to detect objects when using the XML file. For example, not all of the images below will be detected as a license plate.

The quantity of the images is very important as well, the cascade classifier simply will not work on a small image count. With some luck, others have been able to achieve minor and perhaps inconsistent object detection with around 40 positive images and about 200 negative images. More accurate classifiers have used anywhere from 1000 to 5000 images. Ideally you will want more negative images than positive. Defining an exact amount for a success threshold is unclear but rather it is worth while noting that the more images tested against, the more accurate the classifier will be. 100% object detection is not guaranteed.

Also, if possible, it might be best to shrink the images down to a size of no more than 500KB and convert them to a bitmap file as large images may cause the application to slow down and the application was originally designed to work with the bitmap file type. Place these images into the `Haar Training/positive/rawdata` folder.

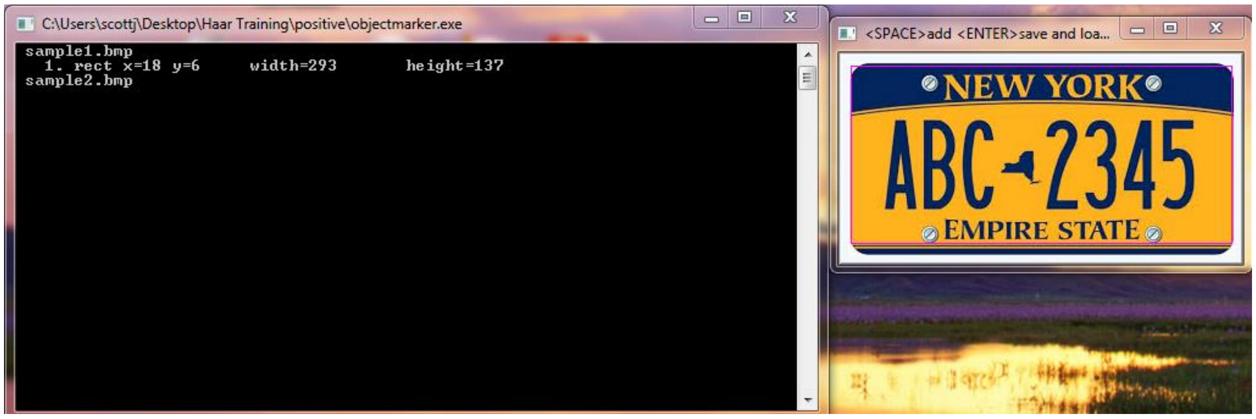

Haar Training ▸ positive ▸ rawdata
Search rawdata
File Edit View Tools Help
Organize Include in library Share with Slide show Burn New folder
Favorites
Desktop
Downloads
Recent Places
sample1.bmp
sample2.bmp
sample3.bmp

Next, you want to gather negative images, these images can be of any size or file format, place these images into the `Haar Training/negative` folder:



After doing so, click on the `create_list.bat` file to generate the bg.txt file, this will list all of the negative images that the classifier needs to refer to.

Next we can begin marking our objects. Navigate to `Training/positive` and run `objectmarker.exe`. This is the pre-built tool that will allow you to mark the area of the positive regions for detection. If the image files are not in bitmap format, the application will most likely close immediately upon startup. When the application starts, the images located in the positive training data folder will then appear in a window along with a second console application. Use the mouse to select the region of interest around the object that you wish to detect. You can always reselect and ROI (region of interest.) The space bar must be hit to save the region's location and you can press the enter key to save that data and move on to the next image. The program will close upon completion of all images.

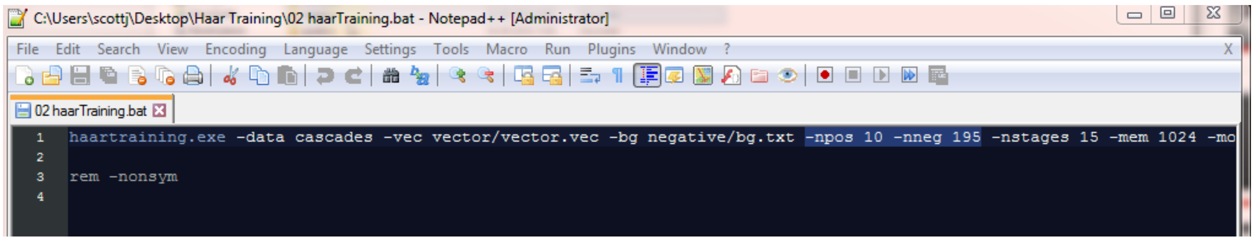

C:\Users\scottj\Desktop\Haar Training\positive\objectmarker.exe
<SPACE>add <ENTER>save and loa...
NEW YORK
ABC 2345
EMPIRE STATE

```
sample1.bmp
  1. rect x=18 y=6      width=293       height=137
sample2.bmp
```

If the image you chose has multiple objects in it that you wish to detect, you can select each of those objects then hit the space bar to save each individual ROI's location.

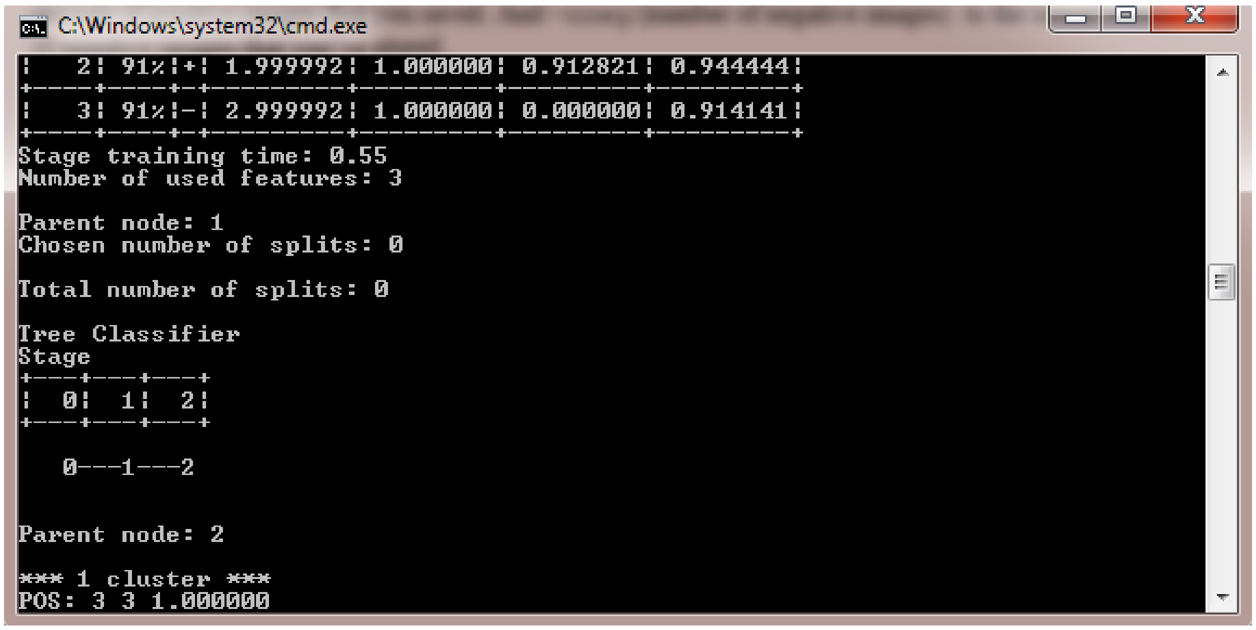
After marking all of the objects, we need to create vector file regarding the positive images. To do this, navigate back to the Haar Training root folder, and run `samples_creation.bat`. This will generate our vector file which is located in the same directory under the "vector" folder.

Next, navigate to the “Cascades” folder and delete any pre-existing data, this is important to avoid potential writing errors when the program wants to write out the cascade data. After all the old cascade data has been removed, navigate back to the root directory and open up `haarTraining.bat` in Notepad++ . This next step is very important as well as it tells your program what it’s looking at and what it should be outputting. Change the values of `–npos` (number of positive images) to the actual amount of positive images that you saved. And `–nneg` (number of negative images) to the actual number of negative images that you’ve stored.


C:\Users\scottj\Desktop\Haar Training\02 haarTraining.bat - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
02 haarTraining.bat

```
haartraining.exe -data cascades -vec vector/vector.vec -bg negative/bg.txt -npos 10 -nneg 195 -nstages 15 -mem 1024 -mo

rem -nonsym

```

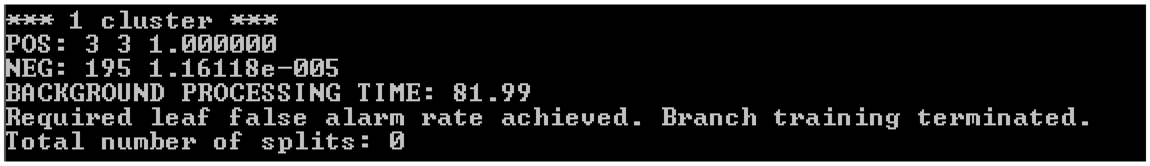
Finally, we can begin training our Haar Cascade Classifier. Navigate back to the root directory and run `haarTraining.bat.` This will begin processing all of the images and generating the cascade classifier xml. **<u>Note</u>**: training can be a very slow process, with a decent amount of images, training may take several hours or even days. The exact time is difficult to put an estimation on. Again, a successful completion of training may not be guaranteed, errors may always occur.



```
¦   2¦ 91%¦+¦ 1.999992¦ 1.000000¦ 0.912821¦ 0.944444¦
+----+----+-+---------+---------+---------+---------+
¦   3¦ 91%¦-¦ 2.999992¦ 1.000000¦ 0.000000¦ 0.914141¦
+----+----+-+---------+---------+---------+---------+
Stage training time: 0.55
Number of used features: 3

Parent node: 1
Chosen number of splits: 0

Total number of splits: 0

Tree Classifier
Stage
+---+---+---+
¦  0¦  1¦  2¦
+---+---+---+

   0---1---2


Parent node: 2

*** 1 cluster ***
POS: 3 3 1.000000
```

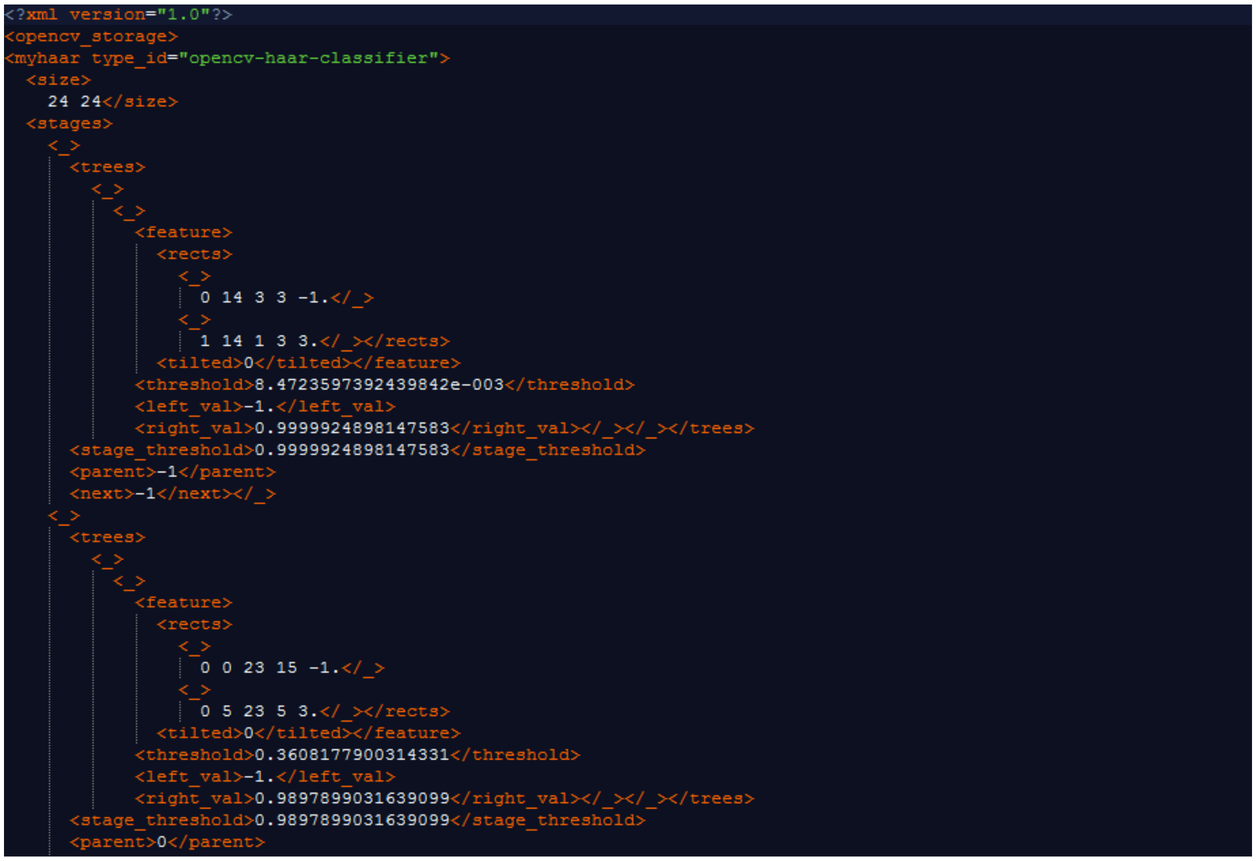
The following line of text will notify you when the training has been completed:

“Branch training terminated.”



```
*** 1 cluster ***
POS: 3 3 1.000000
NEG: 195 1.16118e-005
BACKGROUND PROCESSING TIME: 81.99
Required leaf false alarm rate achieved. Branch training terminated.
Total number of splits: 0
```

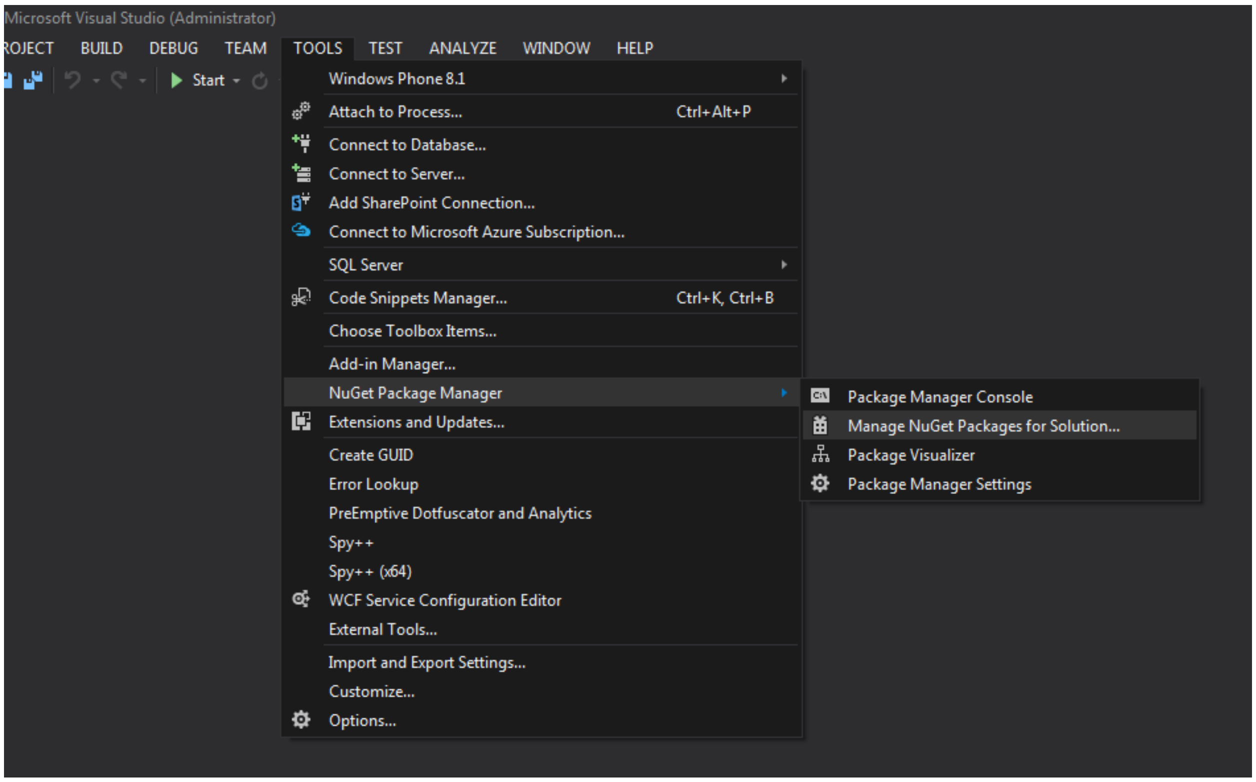
Finally, we can then run `convert.bat`, which is located in the root directory. This will convert all training data into a usable XML file that we can reference in our program.



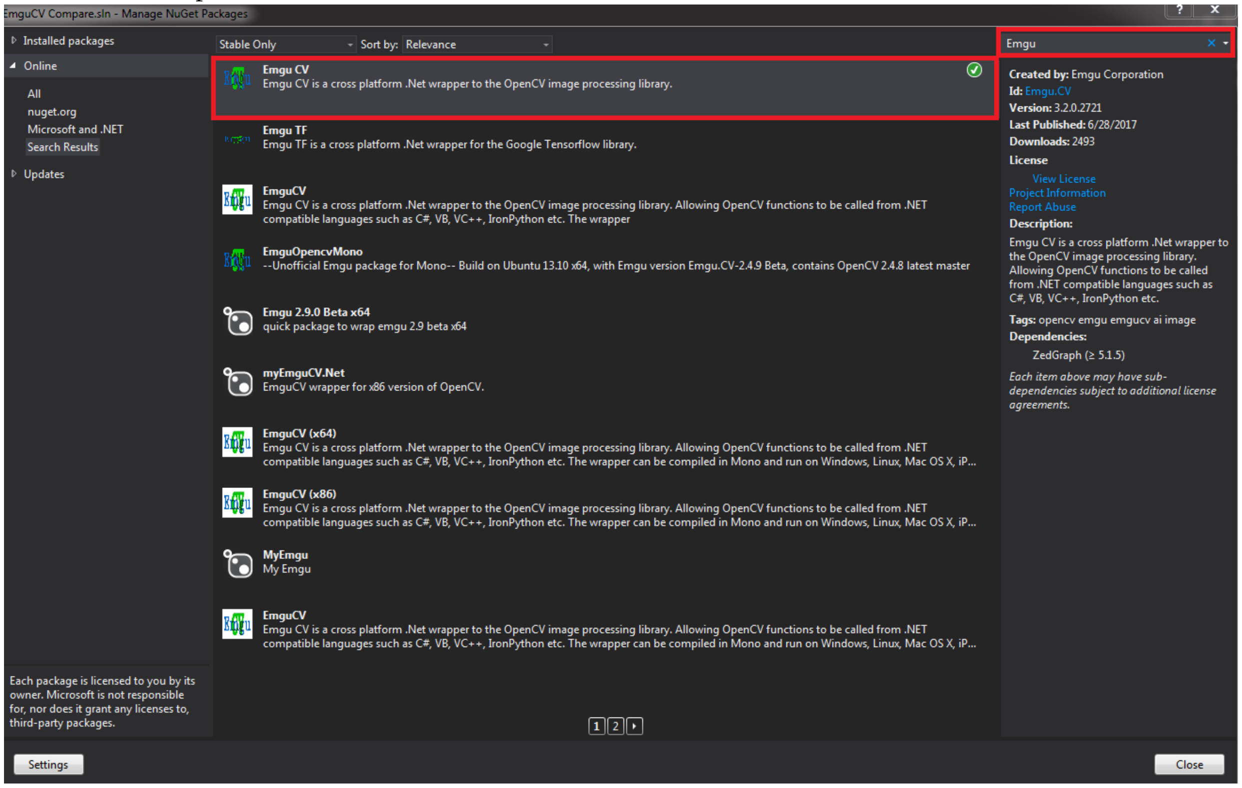
```
<?xml version="1.0"?>
<opencv_storage>
<myhaar type_id="opencv-haar-classifier">
  <size>
    24 24</size>
  <stages>
    <_>
      <trees>
        <_>
          <_>
            <feature>
              <rects>
                <_>
                  0 14 3 3 -1.</_>
                <_>
                  1 14 1 3 3.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>8.4723597392439842e-003</threshold>
            <left_val>-1.</left_val>
            <right_val>0.9999924898147583</right_val></_></_></trees>
      <stage_threshold>0.9999924898147583</stage_threshold>
      <parent>-1</parent>
      <next>-1</next></_>
    <_>
      <trees>
        <_>
          <_>
            <feature>
              <rects>
                <_>
                  0 0 23 15 -1.</_>
                <_>
                  0 5 23 5 3.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>0.3608177900314331</threshold>
            <left_val>-1.</left_val>
            <right_val>0.9897899031639099</right_val></_></_></trees>
      <stage_threshold>0.9897899031639099</stage_threshold>
      <parent>0</parent>
```

Installing Emgu_CV and using the XML:

Now that we've completed our XML generation, we're ready to test it in a program. Open up Visual Studio 2013, create a new project, and go to `Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution`


Microsoft Visual Studio (Administrator)
ROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start
Windows Phone 8.1
Attach to Process... Ctrl+Alt+P
Connect to Database...
Connect to Server...
Add SharePoint Connection...
Connect to Microsoft Azure Subscription...
SQL Server
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...
Add-in Manager...
NuGet Package Manager
Extensions and Updates...
Create GUID
Error Lookup
PreEmptive Dotfuscator and Analytics
Spy++
Spy++ (x64)
WCF Service Configuration Editor
External Tools...
Import and Export Settings...
Customize...
Options...
Package Manager Console
Manage NuGet Packages for Solution...
Package Visualizer
Package Manager Settings

This will bring up the NuGet manager screen. In the upper right hand corner, search for "Emgu" and install the first option.


EmguCV Compare.sln - Manage NuGet Packages

Emgu CV
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library.

Emgu TF
Emgu TF is a cross platform .Net wrapper for the Google Tensorflow library.

EmguCV
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper

EmguOpencvMono
--Unofficial Emgu package for Mono-- Build on Ubuntu 13.10 x64, with Emgu version Emgu.CV-2.4.9 Beta, contains OpenCV 2.4.8 latest master

Emgu 2.9.0 Beta x64
quick package to wrap emgu 2.9 beta x64

myEmguCV.Net
EmguCV wrapper for x86 version of OpenCV.

EmguCV (x64)
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled in Mono and run on Windows, Linux, Mac OS X, iP...

EmguCV (x86)
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled in Mono and run on Windows, Linux, Mac OS X, iP...

MyEmgu
My Emgu

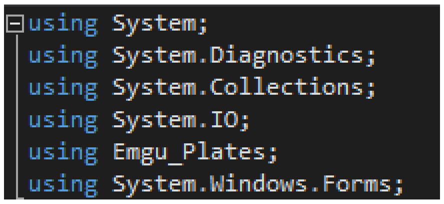
EmguCV
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled in Mono and run on Windows, Linux, Mac OS X, iP...

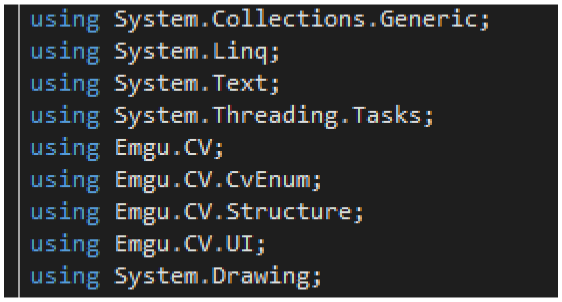
Created by: Emgu Corporation
Id: Emgu.CV
Version: 3.2.0.2721
Last Published: 6/28/2017
Downloads: 2493
License
View License
Project Information
Report Abuse
Description:
Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc.
Tags: opencv emgu emgucv ai image
Dependencies:
ZedGraph (≥ 5.1.5)

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

This will install all of the tools and libraries that you need to run Emgu_CV for C#. You may also want to use the following libraries in your program:


```
using System;
using System.Diagnostics;
using System.Collections;
using System.IO;
using Emgu_Plates;
using System.Windows.Forms;
```

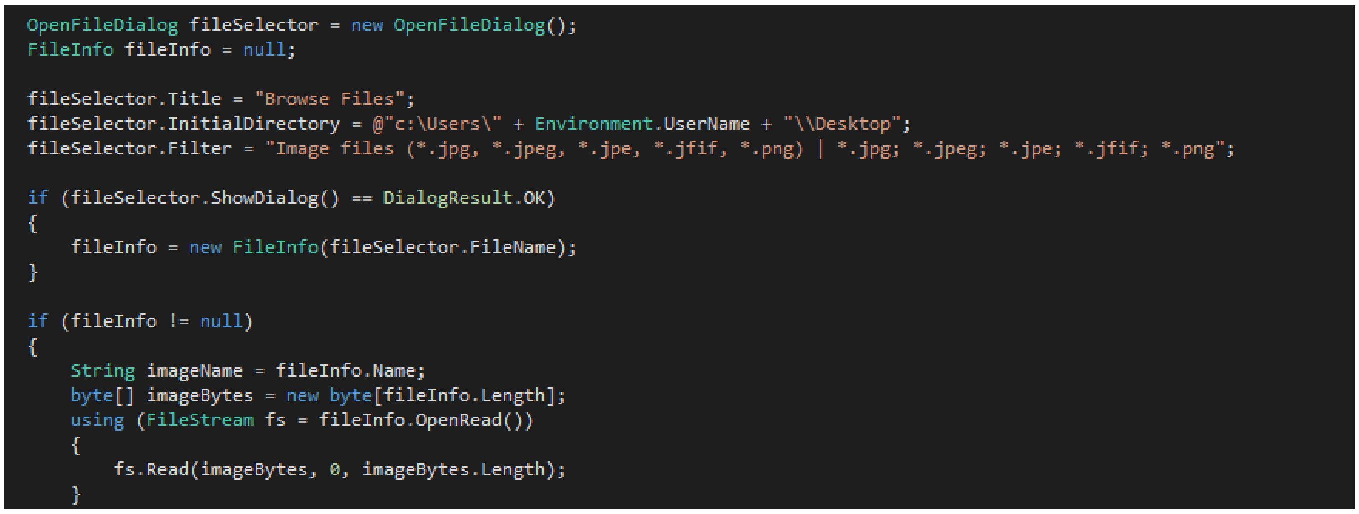

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.UI;
using System.Drawing;
```

Finally, we can begin the code. There are several major parts to the code to get object detection. We only focus on these:

- Reading in an image.
- Storing the image in a Mat() object
- Referencing the cascade XML
- Converting the image to grayscale
- Detecting the object(s)
- Drawing a rectangle around the detected object
- Outputting the image

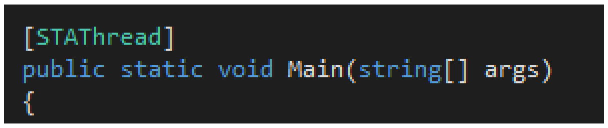
First, we want to allow the user to select any image they want manually, below is the following code that will open up a window browser so that the user can browse their file system to select an image.

`OpenFileDialog` allows the navigation and retrieval of the images on the system, and `FileStream` will read in the image to the program.



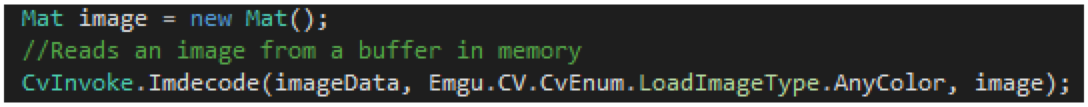
```
OpenFileDialog fileSelector = new OpenFileDialog();
FileInfo fileInfo = null;

fileSelector.Title = "Browse Files";
fileSelector.InitialDirectory = @"c:\Users\" + Environment.UserName + "\\Desktop";
fileSelector.Filter = "Image files (*.jpg, *.jpeg, *.jpe, *.jfif, *.png) | *.jpg; *.jpeg; *.jpe; *.jfif; *.png";

if (fileSelector.ShowDialog() == DialogResult.OK)
{
    fileInfo = new FileInfo(fileSelector.FileName);
}

if (fileInfo != null)
{
    String imageName = fileInfo.Name;
    byte[] imageBytes = new byte[fileInfo.Length];
    using (FileStream fs = fileInfo.OpenRead())
    {
        fs.Read(imageBytes, 0, imageBytes.Length);
    }
```

Be sure to use the following attribute to whatever method you are using the `OpenFileDialog` in or else an exception will occur and your program will not run: (in this case, we are calling `OpenFileDialog` from the main method.)



```
[STAThread]
public static void Main(string[] args)
{
```

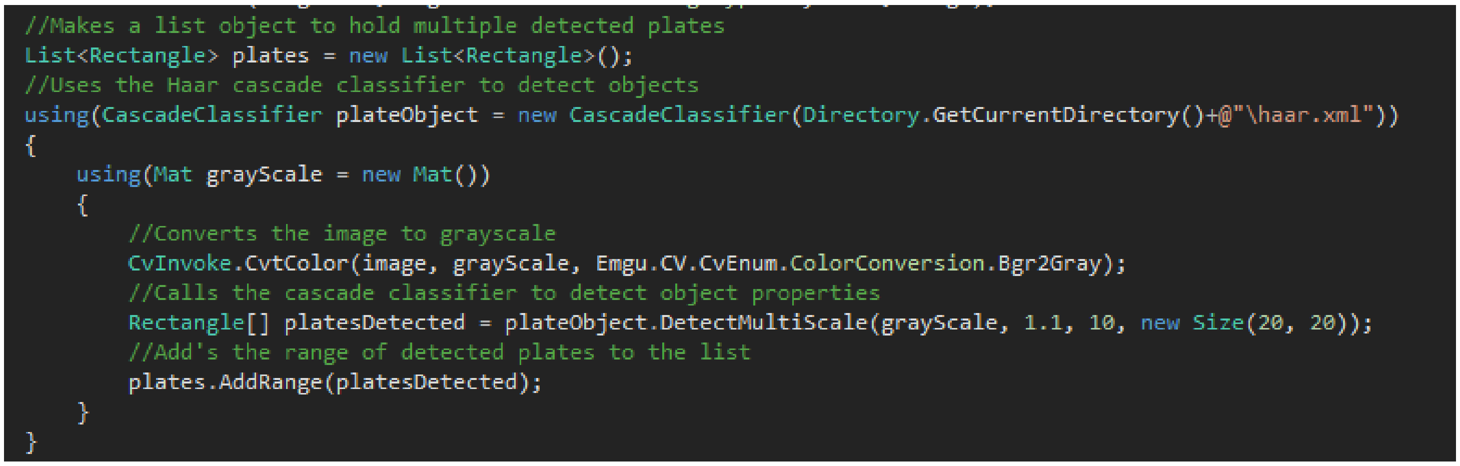
Now that we read in our image and stored it into a byte array via the `FileStream` reader, we can then covert the image to a Mat() object and detect it. Create a new, empty Mat() object and use the following line of code to read the image from a bye array into a mat object:



```
Mat image = new Mat();
//Reads an image from a buffer in memory
CvInvoke.Imdecode(imageData, Emgu.CV.CvEnum.LoadImageType.AnyColor, image);
```

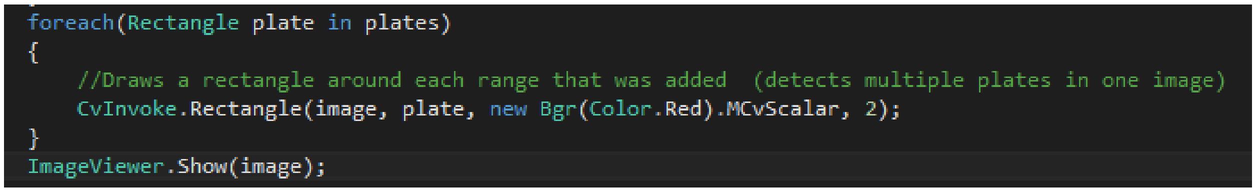
imageData is a Byte[] array that we used to initially read in our image. Next we will need to make a list data structure of type Rectangle to hold a range of different rectangles. This will allow us to detect multiple objects in one image. After creating our data structure, we can then call the cascade classifier class and load our XML file. Note that you would want to place the XML file in the same directory as your program. If not, you must specify the exact file path to the file. `Directory.GetCurrentDirectory` gets the current directory of the executing program.

Next we will use a new Mat() object to hold the grayscale of our image, detect the object using the CascadeClassifier class and add the ROI to the list data structure:



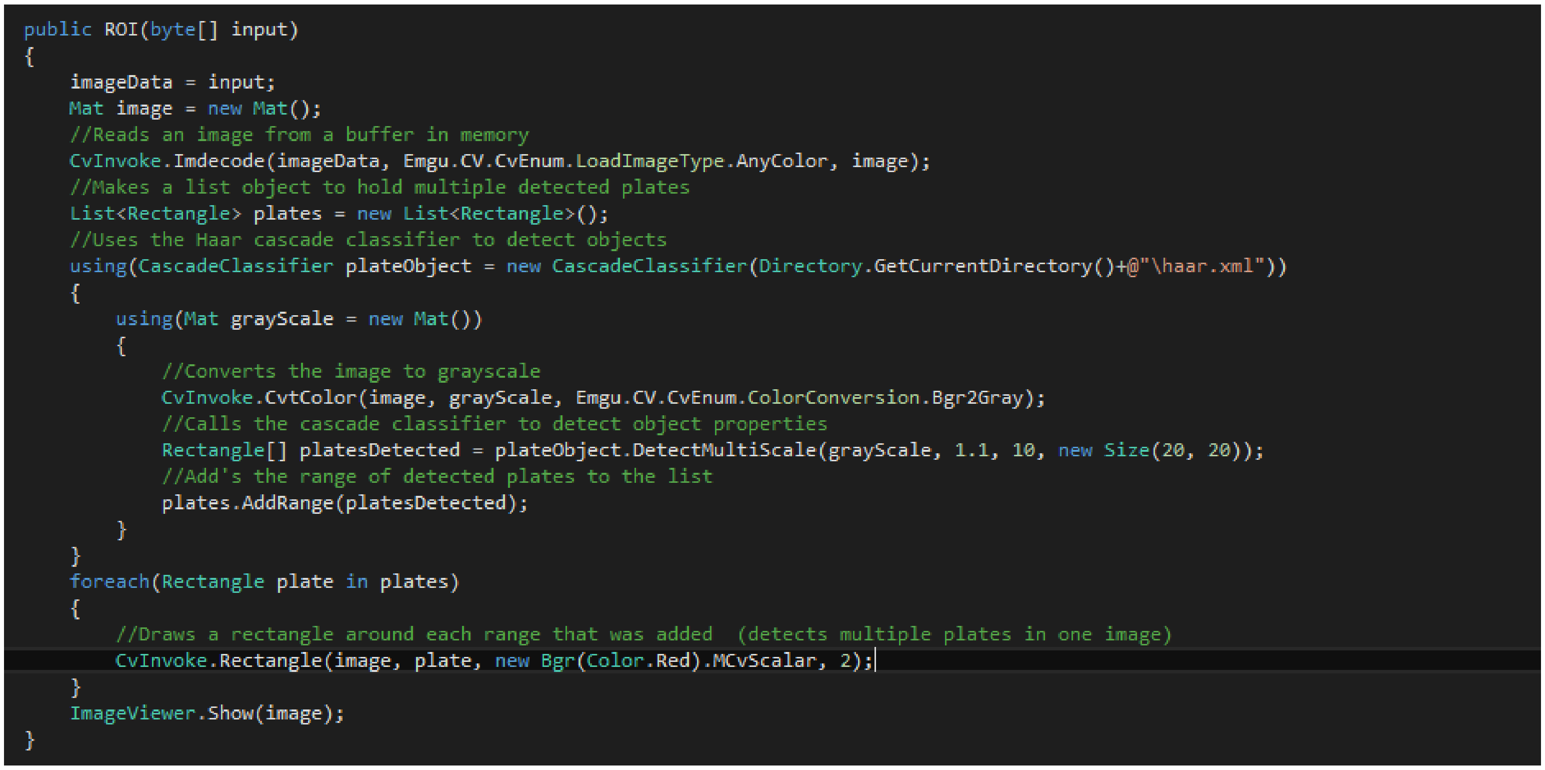
```
//Makes a list object to hold multiple detected plates
List<Rectangle> plates = new List<Rectangle>();
//Uses the Haar cascade classifier to detect objects
using(CascadeClassifier plateObject = new CascadeClassifier(Directory.GetCurrentDirectory()+@"\haar.xml"))
{
    using(Mat grayScale = new Mat())
    {
        //Converts the image to grayscale
        CvInvoke.CvtColor(image, grayScale, Emgu.CV.CvEnum.ColorConversion.Bgr2Gray);
        //Calls the cascade classifier to detect object properties
        Rectangle[] platesDetected = plateObject.DetectMultiScale(grayScale, 1.1, 10, new Size(20, 20));
        //Add's the range of detected plates to the list
        plates.AddRange(platesDetected);
    }
}
```

Finally, now that we have our detected regions, we can draw the rectangle around each detected object and output the image:



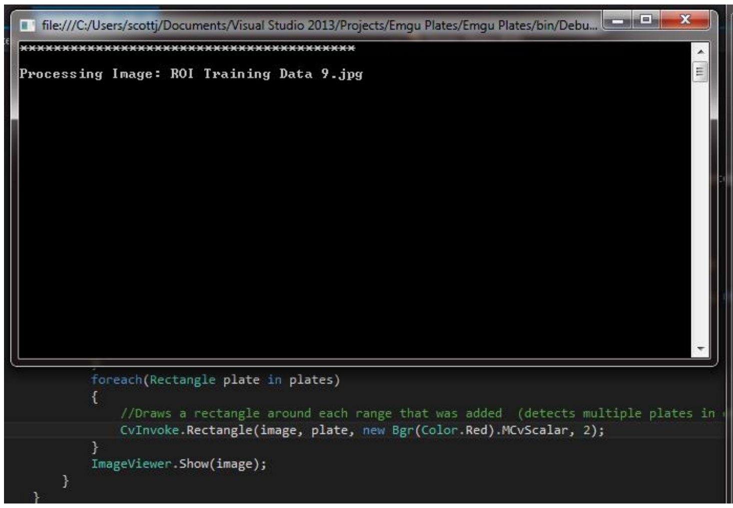
```
foreach(Rectangle plate in plates)
{
    //Draws a rectangle around each range that was added  (detects multiple plates in one image)
    CvInvoke.Rectangle(image, plate, new Bgr(Color.Red).MCvScalar, 2);
}
ImageViewer.Show(image);
```

Collectively, the algorithm follows:



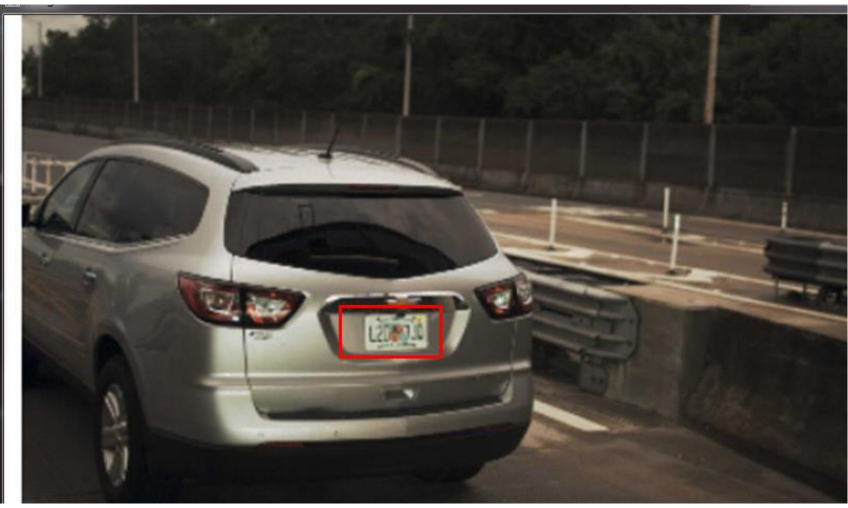
```
public ROI(byte[] input)
{
    imageData = input;
    Mat image = new Mat();
    //Reads an image from a buffer in memory
    CvInvoke.Imdecode(imageData, Emgu.CV.CvEnum.LoadImageType.AnyColor, image);
    //Makes a list object to hold multiple detected plates
    List<Rectangle> plates = new List<Rectangle>();
    //Uses the Haar cascade classifier to detect objects
    using(CascadeClassifier plateObject = new CascadeClassifier(Directory.GetCurrentDirectory()+@"\haar.xml"))
    {
        using(Mat grayScale = new Mat())
        {
            //Converts the image to grayscale
            CvInvoke.CvtColor(image, grayScale, Emgu.CV.CvEnum.ColorConversion.Bgr2Gray);
            //Calls the cascade classifier to detect object properties
            Rectangle[] platesDetected = plateObject.DetectMultiScale(grayScale, 1.1, 10, new Size(20, 20));
            //Add's the range of detected plates to the list
            plates.AddRange(platesDetected);
        }
    }
    foreach(Rectangle plate in plates)
    {
        //Draws a rectangle around each range that was added  (detects multiple plates in one image)
        CvInvoke.Rectangle(image, plate, new Bgr(Color.Red).MCvScalar, 2);
    }
    ImageViewer.Show(image);
}
```

Then you can run the program. If no object is detected, there are several scenarios, how you process your image in the code may be incorrect, make sure the grayscale Mat() is being used. If not, ensure that the image you wish to detect an object in is not difficult for the program to recognize the object. Finally, the cascade trainer's XML file simply may not have successfully documented properties well. If so, you may need to repeat the training process.



```
file:///C:/Users/scottj/Documents/Visual Studio 2013/Projects/Emgu Plates/Emgu Plates/bin/Debu...
****************************************
Processing Image: ROI Training Data 9.jpg
```

```
        foreach(Rectangle plate in plates)
        {
            //Draws a rectangle around each range that was added  (detects multiple plates in
            CvInvoke.Rectangle(image, plate, new Bgr(Color.Red).MCvScalar, 2);
        }
        ImageViewer.Show(image);
    }
}
```



These are the barebones of simple object detection and may not always be the most efficient. Tweaks or complete revamps of code may need to be made depending on what you wish to accomplish.

For further research about motion and object detection, OpenCV or different utilities, refer to OpenCV's website: http://opencv.org/