EmguCV 入门指南

翻译：gola　E-mail：njuidog@163.com

1 封装 Opencv

函数映射 –Emgu.Cv.CvInvoke

CvInvoke 类使得 C#语言可以直接调用 Opencv 函数。在这个类中的每个函数均对应于 Opencv 中的同名函数。举个例子：

```
IntPtr image = CvInvoke.cvCreateImage(new System.Drawing.Size(400, 300),
CvEnum.IPL_DEPTH.IPL_DEPTH_8U, 1);
```

等价于下面这个在 c 中的调用：

```
IplImage* image = cvCreateImage(cvSize(400, 300), IPL_DEPTH_8U, 1);
```

以上两者均创建一个 400*300，8-bit unsigned，grayscale 图像.

结构映射 Emgu.CV.Structure.Mxxx

这种结构类型对应于 Opencv 中的结构：

Emgu CV Structure	OpenCV structure
Emgu.CV.Structure.MIplImage	IplImage
Emgu.CV.Structure.MCvMat	CvMat
...	...
Emgu.CV.Structure.M*xxxx*	*xxxx*

通过命名规则，我们可以发现 Emgucv 中 M 的后缀同 Opencv 中的结构名相同。Emgucv 还借用了一些.Net 中存在的结构来对应于 Opencv 中的结构：

.Net Structure	OpenCV structure
System.Drawing.Point	CvPoint
System.Drawing.PointF	CvPoint2D32f
System.Drawing.Size	CvSize
System.Drawing.Rectangle	CvRect

枚举常量映射 Emgu.CV.CvEnum

CvEnum 名空间提供了对 OpenCV 枚举常量（宏定义）的映射。举个例子：`CvEnum.IPL_DEPTH.IPL_DEPTH_8U` 和 `IPL_DEPTH_8U` in OpenCV 有相同的值都等于 8.

2.托管类

Images 如何使用

Depth 和 Color 作为泛型参数

一张图片被 Depth 和 Color 定义。创建一个 8bit unsigned grayscale 图像，在 Emgucv 中这样调用：

```
Image<Gray, Byte> image = new Image<Gray, Byte>( width, height);
```

这种句法不仅让你意识到图像的 depth 和 color，而且在编译时限制了你使用函数和捕获错误。例如： Image<TColor, TDepth> 类中的 `SetValue(TColor color, Image<Gray, Byte> mask)` 函数只接受一通道的颜色值并必须以一个 8bit usdigned grayscale 图像存储和表示。任何试图使用一个 16bit floating point 或者 non-grayscale 图像作为参数调用该函数都将得到一个 compile time error！

创建图像

尽管可以通过调用 CvInvoke.cvCreateImage 来创建图像，但推荐的方法是构建一个 Image<TColor, TDepth>对象。这种方法有以下几点优点：

1. 内存会被垃圾回收器自动回收
2. Image<TColor, TDepth>类可以使用可视化调试工具调试
3. Image<TColor, TDepth>类中有 Opencv 中没有的更好的方法，例如图像像素的通用操作、转换为位图等。

图像 Color

Image 类的第一个泛型参数标明图像的 **color** 类型。例如：Image<Gray, ...> img1;表明 img1 是一个一通道 grayscale 图像。Emgucv 1.3.0.0 支持的 color 类型包括：

- Gray
- Bgr (Blue Green Red)
- Bgra (Blue Green Red Alpha)
- Hsv (Hue Saturation Value)
- Hls (Hue Lightness Saturation)
- Lab (CIE L*a*b*)
- Luv (CIE L*u*v*)
- Xyz (CIE XYZ.Rec 709 with D65 white point)
- Ycc (YCrCb JPEG)

Image 深度

Image 类的第二个泛型参数标明图像的深度类型。Emgucv 1.4.0.0 支持的 color 类型包括：

- Byte

- SByte
- Single (float)
- Double
- UInt16
- Int16
- Int32 (int)

创建一个新图像

若要创建一个 480*320 的 Bgr color，8-bit unsigned 深度图像，在 C#中的方法是：

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320);
```

若要想指定图像的背景 color 值，假设颜色为蓝色，在 C#中的方法是：

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320, new Bgr(255, 0, 0));
```

从文件中读入图像

从文件中读入并创建图像也同样简单。如果图像文件的目录名是"MyImage.jpg"，在 C#中的方法是：

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>("MyImage.jpg");
```

从位图创建图像

从一个.Net Bitmap 对象创建图像在 C#中的方法是：

```
Image<Bgr, Byte> img = new Image<Bgr, Byte>(bmp); //where bmp is a Bitmap
```

自动垃圾回收

内存会被垃圾回收器自动回收。

一旦垃圾回收器扫描到没有对 Image<TColor, TDepth>类的引用，它将调用处理函数释放内存空间。

内存垃圾回收扫描的时间是无规律的。当处理大图像时，建议调用 Dispose()函数来明确的释放对象。或者，使用 C#中的关键字 using 限制图像范围，例如：

```
using (Image<Gray, Single> image = new Image<Gray, Single>(1000, 800))
{
   ... //do something here in the image
} //The image will be disposed here and memory freed
```

像素的获取与赋值

慢却安全的方法

假设你正在使用一个 Image<Bgr, Byte>。你可以获得在 y 行 x 列上的像素点：

```
Bgr color = img[y, x];
```

对 y 行 x 列上的像素点的赋值也是比较简单的：

```
img[y,x] = color;
```

快速的方法

图像像素是以三维数组的形式存储的，你可以通过使用迭代器来对像素点做获取或赋值。

方法

命名规定

在 Image<TColor,TDepth>类中的方法（函数）和 Opencv 中的 cvXYZ 函数是等价的。例如：

Image<TColor,TDepth>.Not()函数同 cvNot 函数一样，返回值都是结果图像。

方法 _XYZ()和 XYZ()差不多，但其操作时在原图像之上的，而不是返回一个结果图像。例如： Image<TColor,TDepth>._Not()函数即在原图像上实现了操作。

操作符重载

操作符+-*/都已被重载过，它们可以像下面的代码一样完美的使用：

```
Image<Gray, Byte> image3 = (image1 + image2 - 2.0) * 0.5;
```

泛型操作

使用 Emgucv 的一大优势就是可以使用泛型操作。对此我们将举例说明。假设我们有一个 grayscale 的图像：

```
Image<Gray, Byte> img1 = new Image<Gray, Byte>(400, 300, new Gray(30));
```

为了转换图像中的所有像素点我们可以调用 Not 函数

```
Image<Gray, Byte> img2 = img1.Not();
```

我们同样可以使用 Image<TColor, TDepth>类中的泛型转换方法：

```
Image<Gray, Byte> img3 = img1.Convert<Byte>( delegate(Byte b) { return (Byte) (255-b); } );
```

函数返回图像 img2 和 img3 值相同。

乍一看并不会觉得泛型方法有多大的优势，其实事实上，Opencv 中的 Not 函数在性能上有优化，好于泛型方法。但是，泛型方法通过牺牲一点点性能而换来了灵活性的极大提高。

假设我们有一个 Image<Gray, Byte> img1。我们想要创建一个 size 相同的单通道浮点数型图像，要求新的图像和 img1 中每个像素点的值相对应，该操作泛型方法是：

```
Image<Gray, Single> img4 = img1.Convert<Single>( delegate(Byte b) { return (Single) Math.cos( b * b / 255.0); } );
```

这个语法简单而明确。而另一方面，在 Opencv 中这个操作则无法通过简单的等价关系从 Math.cos 函数实现转化。

图像绘图

Draw()函数可以在 Image< Color, Depth>类的对象上各种不同的图形，包括 fonts，circles，rectangles，boxes，ellipses，contours。可以阅读 document 来学习各种绘图函数。

Color 和深度的转换

对一个 Image< Color, Depth>类的对象的 color 和深度做转换比较简单。比如，如果你有一个 Image<Bgr, Byte> img1，你希望转换为<Gray,Single>图像，你只需要这样做：

```
Image<Gray, Single> img2 = img1.Convert<Gray, Single>();
```

显示图像

使用 ImageBox

Emgucv 建议 ImageBox 来显示图像，主要有以下几点原因：

1. ImageBox 是个高性能显示图像的控件，它显示一张图像时和 Image 类共享内存控件，无需额外的内存控件，非常快。
2. 用户可以在使用 ImageBox 显示图像时测试图像像素点值，视频帧率，color 类型等。
3. 可以通过鼠标点击即可方便的实现简单图像操作的执行。

转换为位图

Image 类有一个 Bitmap()函数可以返回一个 Bitmap 类的对象，位图可以简单的使用 windows form 中的 PictureBox 控件。

XML 序列化

为何序列化

Image<TColor, TDepth> 可以 XML 序列化，你可能会问我们为什么需要对一个图像序列化。答案很简单，我们需要在 web 服务中使用它！

Image<TColor, TDepth> 类提供了 ISerializable 接口，当你在 WCF 上工作时，你可以自由的使用 Image<TColor, TDepth> 类作为 web 服务的参数或者返回值。

这是非常理想的方法。例如，你正在使用计算机集群来做模式识别并使用一台中央计算机来调度任务。

Image<TColor, TDepth>转化为 XML

你可以使用下面的代码实现从 Image<Bgr, Byte> 到 XmlDocument 的转换：

```
            StringBuilder sb = new StringBuilder();
(new XmlSerializer(typeof(Image<Bgr, Byte>))).Serialize(new StringWriter(sb), o);
            XmlDocument xDoc = new XmlDocument();
            xDoc.LoadXml(sb.ToString());
```

XML 转化为 Image<TColor, TDepth>

你可以使用下面的代码实现从 XmlDocument xDoc 到 Image<Bgr, Byte>的转换：

```
Image<Bgr, Byte> image = (Image<Bgr, Byte>)(new XmlSerializer(typeof(Image<Bgr,
Byte>))).Deserialize(new XmlNodeReader(xDoc));
```

Matrices 如何使用

深度作为泛型参数

一个 Matrices 被其 depth 定义。假设我们创建一个 32bit –floating 矩阵：

```
Matrix<Single> matrix = new Matrix<Single>(width, height);
```

矩阵深度

EnguCV 1.4.0.0 中支持的深度类型包括：

- Byte
- SByte

- Single (float)
- Double
- UInt16
- Int16
- Int32 (int)

XML 序列化

矩阵转化为 XML

你可以使用下面的代码实现从 Matrix<double> 到 XmlDocument 的转换：

```
StringBuilder sb = new StringBuilder();
(new XmlSerializer(typeof(Matrix<double>))).Serialize(new StringWriter(sb), o);
XmlDocument xDoc = new XmlDocument();
xDoc.LoadXml(sb.ToString());
```

XML 转化为矩阵

你可以使用下面的代码实现从 XmlDocument xDoc 到 Matrix<double>的转换：

```
Image<Bgr, Byte> image = (Image<Bgr, Byte>)(new XmlSerializer(typeof(Image<Bgr, Byte>))).Deserialize(new XmlNodeReader(xDoc));
```

错误异常处理

Emgucv 注册了一个自定义的错误处理方法，当遇到错误异常时，一个 **CvException** 会被抛出。

代码文档

Xml 文档

你可以通过我们的 Online Documentation 来查看最新的代码和代码说明。

类函数文档

一个包含了很多类函数用法的代码例程库： Online Code Reference.

Visual Studio 中的自动补全

如果你使用 **Visual Studio** 作为你的开发工具，当你在开发 **Emgucv** 应用时会有自动补全支持。

例程

Online Code Reference

C#

图像处理例程

引入

- Hello World
- User Guide to EMGU and Accessing Image Data
- Camera Capture in 7 lines of code

中间件

- Shape (Triangle, Rectangle, Circle, Line) Detection
- SURF Feature Detector
- Windows Presentation Foundation (WPF)
- Face detection in Csharp
- Pedestrian Detection, Histogram of oriented gradients (HOG)
- Traffic Sign Detection
- License Plate Recognition (LPR), Optical Character Recognition (OCR)
- Image Stitching
- Using the Kalman Filter

计算几何学例程

- Delaunay's Triangulation and Voronoi Diagram
- Convex Hull
- Ellipse Fitting
- Minimum Area Rectangle
- Minimum Enclosing Circle

机器学习例程

- Normal Bayes Classifier
- K Nearest Neighbors
- Support Vector Machine (SVM) - thanks to Albert G.
- Expectation-Maximization (EM)
- Neural Network (ANN MLP)
- Mushroom Poisonous Prediction (Decision Tree)

C++

- Hello World

IronPython

- Setting up Emgu CV and IronPython
- Face Detection from IronPython

VB.NET

- Face Detection in VB.NET

- Hello World in VB.NET