lm2Learn Manual

The motivation for developing lm2Learn (Image to Learn) comes from academic, government and industrial collaborations that involve development of new computer methods and solutions for understanding complex data sets. Images and other types of data generated by various instruments and sensors form complex and highly heterogeneous data sets, and pose challenges on knowledge extraction. In general, the driver for the lm2Learn suite of tools is to address the gap between complex multi-instrument raw data and knowledge relevant to any specific application.

Introduction

The motivation for developing Im2Learn (Image to Learn) comes from academic, government and industrial collaborations that involve development of new computer methods and solutions for understanding complex data sets. Images and other types of data generated by various instruments and sensors form complex and highly heterogeneous data sets, and pose challenges on knowledge extraction. In general, the driver for the Im2Learn suite of tools is to address the gap between complex multi-instrument raw data and knowledge relevant to any specific application.

The objective of the Im2Learn suite of tools is to research and develop solutions to real life problems in the application areas of machine vision, precision farming, land use and land cover classification, map analysis, geo-spatial information systems (GIS), synthetic aperture radar (SAR) target and multi-spectral scene modeling, video surveillance, bio-informatics, microscopy and medical image processing, and advanced sensor environments. The main goal of the IM2LEARN research and development is to automate information processing of repetitive, laborious and tedious analysis tasks and build user-friendly decision-making systems that operate in automated or semi-automated mode in a variety of applications. The development is based on theoretical foundations of image and video processing, computer vision, data fusion, statistical and spectral modeling, data mining and pattern recognition, software engineering and sensor design.

Application Examples

Applications areas in which this tool could be used are many. Some application examples are listed below and include bioinformatics, hydrology, precision farming, machine vision in the semiconductor industry, topography, plant biology, medicine, automobile industry, database retrieval, surveillance, and digital library.

- Band Selection
    - Unsupervised and supervised methods for selecting bands from hyperspectral imagery.
- Filtering to Achieve Noise Reduction
    - Image de-noising and de-blurring in microarray data.
- Classification: Two and N-class classification
    - Building land cover and land use maps.
    - Detecting change in hydrology data.
    - Mask generation for data-driven scene modeling
- Matching: Searching for Well-defined Landmarks
    - Image calibration
    - Geo-registration
- Tracking: Landmark-Tracking Applied to Microscopy Data
    - Registration and alignment based on motion imagery.

- Statistical Modeling: Models Applied to Aerial and Satellite Data
    - Crop analysis.
    - Complex multi-spectral scene modeling.
- Contour Detection: Iso-contour Extraction from Historical Maps
    - Environmental preservation project.
- Registration: 3D volume reconstruction
    - Analyses of fluorescent confocal laser scanning microscope imagery.
- Stereo: 3D position estimation
    - Design of hazard aware spaces
- Document analysis: PDF document analysis
    - Building information repository from images and text in PDF documents

Im2Learn Code Architecture

Im2Learn code is decomposed into several components such as Im2Learn core, Im2Learn extensions (re-usable plugins), Im2Learn main (custom configuration files), test routines, and custom code driven by specific projects. This document will primarily focus on Im2Learn core and Im2Learn extensions components. The Im2Learn main component allows us to distribute code configurations according to the needs of researchers, for example, bio-medical plugins belong to a different configuration than geo-spatial analysis plugins. The test routines are an integrated part of code testing in addition to auto-builds performed every night. Finally, the custom code is a repository of Im2Learn plugins that are funded by private industry or use libraries that do not allow code distribution.

Im2Learn Functionality

The Im2Learn - Image to Learn is capable of analyzing multidimensional, multi-variate (multi-band) image data and enables user interaction with the data. The basic set of IM2LEARN tools also provides support for histograms, image band ranking, image statistics, correlation matching, filtering, contour extraction, image calculator, tracking, blob analysis, image thresholding, segmentation and Isodata clustering. The input and output interface can handle loading and saving of individual images in several standard file formats. The current support of standard input image file formats includes: jpg, gif, tif (Ver. 6.0 with no compression), pgm, ppm, bmp, png, pnm, img, and an internal image format denoted as iip (image interleaved pixels). The software also supports some of the file formats used in Geospatial Information Systems (GIS), such as, USGS DEM (Float), ESRI IMG format, HDF4, HDF5, LAN, GeoTiff and ESRI Shape file format. All GIS related files are loaded with their georeferencing information that is used for geo-registration of raster and vector data sets. The input can be a sequence of images (e.g., video frames) or a combination of bands from several image data sources in multiple standard file formats.

Im2Learn User Interface

The Im2Learn user interface incorporates a user-friendly front end with mouse-driven operations for selecting an area of interest and dialog-driven operations for experimenting with parameter variations. The current visualization capabilities include image display as a set of still selected frames (gray-scale or color), movies (temporal sequence of image bands) or plots of attributes (band values) over a local spatial neighborhood. As part of the visualization, numerical results can be shown using a scatter plot tool or a text area dialog.

How to use Im2Learn

Core Code

The core code of Im2Learn contains the datatypes and classes that are needed by almost all IM2LEARN applications. This section describes the core code including the ImageObject datatype used as image representation in memory and the functions that are associated with it. This section also contains various classes for loading images from disk, displaying images as well as classes for extending the Im2Learn functionality.

ImageObject

The heart and soul of Im2Learn is the ImageObject class. It is used to modify and display the images and to produce all the abbreviations of it, like segmenting the image for instance.

The ImageObject class is used to store information about an image. The image is stored in row major order with the bands interleaving the pixels (for example an image with 3 rows, 2 columns and 3 bands is stored as RGBRGBRGBRGBRGBRGB where the first triplet is the pixel at row 1, column 1, the second triplet is the pixel at row 1, column 2 etc.). ImageObject can be of any datatype, any number of bands (samples per pixel), rows (height) or columns (width).

Any additional information about the image can be stored as a property associated with the image. Properties that begin with an underscore are temporary properties and will not be copied when the image is cloned, or saved to disk. Some common properties have their keys predefined in this class.

How to speed up your application: This class is an abstract class, any time you call getXXX() it will have to do a lookup to see what class really to call, and call the functions on this class. So one speedup is to simply do the check your self, typecast the ImageObject to the appropriate class and call the functions on this class. Do this either inside the loop or make multiple loops for each datatype (still keeping the abstract version in case a datatype is missing in your implementation). The fastest way is to use the getData function and typecast the returned object to the appropriate array (byte[], double[] etc.), this will give you direct access to the imagedata. If any data is modified computeMinMax or setMinMaxInvalid will need to be called to mark the fact that the current minimum and maximum values associated with the image are not valid.

An ImageObject can also be used to just store information about an image and will not have any imagedata associated with it in that case. To test if an ImageObject has any data, use either getData() or isDataValid(). Accessing one of the get or set functions when no data is present will throw a NullPointer exception. No checks are done in the get and set function to see if there is any imagedata to speed them up.

Extending Im2Learn

To allow for easy extending Im2Learn a plugin architecture was designed. This architecture

allows for easily adding new functionality as well for extending the fileformats it can recognize. First will be described how to add new functionality to Im2Learn and next will be described how to add new fileformats.

Adding Functionality

To add new functionality to Im2Learn the developer will need to implement the IM2LEARNMenu interface. This interface tells the Im2Learn system the names of the menus to be added to the mainframe. Once the plugin is added to an imagepanel the reference to this imagepanel is passed back to the plugin, allowing the plugin to get the ImageObject currently shown. To be able to add help to the central helpsystem of Im2Learn the plugin needs to implement the HelpEntry interface. Examples of the IM2LEARNMenu implementation and HelpEntry can be found in the next two sections. A full example of a plugin for Im2Learn can be found in Example Plugin.

IM2LEARNMenu

The IM2LEARNMenu class is an interface that needs to be implemented if the class is to be appended to either the ncsa.Im2Learn.main menu or the panel. It returns a list of menu entries that will be added to the ncsa.Im2Learn.main menu bar. If the menuitem is a menu it will be only added to the menu if it does not already exist. If it exists the menu entries are appended to then end of the existing menu.

```
// ------------------------------------------------------------
// IM2LEARNMenu implementation
// ------------------------------------------------------------
/**
 * Store the reference to the imagepanel for later use.
 * The imagepanel passed in as argument is the
 * imagepanel to which this tools is associated.
 */
public void setImagePanel(ImagePanel imagepanel) {
  this.imagepanel = imagepanel;
}

/**
 * For this tool there is no operation that works on the
 * imagepanel directly and thus we return null
 * indicating that no menu entries need to be created on
 * the panel.
 */
public JMenuItem[] getPanelMenuItems() {
  return null;
}
```

```
/**
 * Create a menu entry called Tools, and attach a
 * submenu entry to this for this class, called swap. If
 * the user selects this class
 */
public JMenuItem[] getMainMenuItems() {
  JMenu menu = new JMenu("Tools");
  JMenuItem item = new JMenuItem(new AbstractAction("Swap") {
    public void actionPerformed(ActionEvent e) {
      if (!isVisible()) {
        Window win = SwingUtilities.getWindowAncestor(imagepanel);
        setLocationRelativeTo(win);
        setVisible(true);
      }
      toFront();
    }
  });
  menu.add(item);

  return new JMenuItem[] { menu };
}

/**
 * When a new image is loaded make sure that the current
 * swapped image is removed, i.e. perform a reset. Only
 * need to reset the image if this frame is visible.
 */
public void imageUpdated(ImageUpdateEvent event) {
  if (!isVisible()) {
    return;
  }

  if (event.getId() == ImageUpdateEvent.NEW_IMAGE) {
    reset();
  }
}
```

HelpEntry

The Im2Learn system comes with a build in help system. By using the HelpEntry interface the developer can create help that is specific for the code that is written and add it to the build in help system. The help system for Im2Learn is a tree of nodes, each node can have subnodes and so on. When the user clicks on a node, the help system will first check to see if this node is a leaf node (i.e. no subnodes are available), if this is the case the system will find the class that added the leaf node to the help system and ask for a URL to the help documentation.

The URL that is returned can be either a URL to a webpage outside of Im2Learn or, the preferred method, a URL that points to a HTML file that is shipped with Im2Learn. The easiest method for the developer to create a URL to the help page, is to place the HTML file, and all associated images, in a directory called help. In the code the developer can than point to the HTML file using this.getClass().getResource("help/file.html"). Using this code will make sure that the help file can be found even if the class files (and the HTML and image files) are put in a jar file for distribution.

```
// ------------------------------------------------------------
// HelpEntry implementation
// ------------------------------------------------------------

/**
 * Create a node for the help system called Tools, and a
 * leafnode called Swap. The leafnode will contain the
 * documentation about this function.
 */
public HelpTopic[] getTopics() {
  HelpTopic topic = new HelpTopic("Tools");
  new HelpTopic("Swap", topic);
  return new HelpTopic[] { topic };
}

/**
 * Return the right documentation depending on what node
 * is selected, in this case only the Swap node should
 * exist.
 */
public URL getHelp(String menu) {
  if (menu.equals("Swap")) {
    return getClass().getResource("help/swap.html");
  } else {
    return null;
  }
}
```

ImageLoader

To load an image from disk or to save an image to disk the developer can use the ImageLoader class. This class has static methods readImage methods that will take a String as argument that points to the file that needs to be loaded. The ImageLoader class will then use the loaders provided to find the best loader to load the image. When saving an image the developer can call writeImage with the ImageObject and the filename as a String. Again the ImageLoader will find the best writer to save the image.

The ImageLoader can be extended to include new readers and writers by the application developer. To do this the extension will need to implement the ImageReader class to be able to load images and the ImageWriter class to write images to disk.

ImageReader

To be able to load a new fileformat the ImageReader interface needs to be implemented. The example gives an implementation of an ImageReader that can read serialized ImageObjects from disk.

The ImageLoader will first call canRead with the filename and the first 100 bytes of the file. Based on either one of those the loader should decide if it can read the rest of the file. Most images will have a magic set of bytes at the beginning that identify what type of image it is, for example GIF files start always with GIF.

Next either readImageHeader or readImage is called to load the image from disk. This function should return a reference to the ImageObject that was loaded from disk.

```
import ncsa.Im2Learn.core.datatype.ImageException;
import ncsa.Im2Learn.core.datatype.ImageObject;
import ncsa.Im2Learn.core.datatype.SubArea;
import ncsa.Im2Learn.core.io.ImageReader;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.io.*;

/**
 * Load an object that was saved as a serialized object. This class
 * will load an object that was previously saved as an serialized
 * object. This could fail if the object uses classes in the
 * properties that are not known to the current instance.
 */
public class ObjectLoader implements ImageReader {
    private static Log logger = LogFactory.getLog(ObjectLoader.class);

    /**
     * Try and find a reader, if one found we can actually read the
     * file.
     *
     * @param filename of file to be read.
     * @param hdr      used for magic number
     * @return true if image can be read.
     */
    public boolean canRead(String filename, byte[] hdr) {
```

```
        return (filename.endsWith(".object"));
    }

    /**
     * Loads an image and returns the imageobject containing the
     * image.
     *
     * @param filename of the file to load.
     * @param subarea  of the file to load, or null to load full
     *                 image.
     * @param sampling is the sampling that needs to be done.
     * @return the imageobject containing the loaded image
     * @throws java.io.IOException if an error occurrs reading the
     *                             file.
     */
    public ImageObject readImage(String filename, SubArea subarea, int sampling)
throws IOException {
        return readImage(filename, subarea, sampling, false);
    }

    /**
     * This function will read the file and return an imageobject
     * that contains the information of the image but not the
     * imagedata itself.
     *
     * @param filename of the file to be read
     * @return the file as an imageobject except of the imagedata.
     * @throws java.io.IOException if the file could not be read.
     */
    public ImageObject readImageHeader(String filename) throws IOException {
        return readImage(filename, null, 1, true);
    }

    /**
     * Return a list of extensions this class can read.
     *
     * @return a list of extensions that are understood by this
     *         class.
     */
    public String[] readExt() {
        return new String[]{"object"};
    }

    /**
     * Loads an image and returns the imageobject containing the
     * image.
     *
```

```
     * @param filename of the file to load.
     * @param subarea  of the file to load, or null to load full
     *                 image.
     * @param sampling is the sampling that needs to be done.
     * @param header   true if only header needs to be read.
     * @return the imageobject containing the loaded image
     * @throws java.io.IOException if an error occurrs reading the
     *                             file.
     */
    public ImageObject readImage(String filename, SubArea subarea, int sampling,
boolean header) throws IOException {
        FileInputStream   fis = new FileInputStream(filename);
        ObjectInputStream inp = new ObjectInputStream(fis);
        ImageObject result;
        try {
            result = (ImageObject) inp.readObject();
        } catch (ClassNotFoundException exc) {
            logger.debug("Could not load file.", exc);
            throw(new IOException(exc.toString()));
        } catch (ClassCastException exc) {
            logger.debug("Could not load file.", exc);
            throw(new IOException(exc.toString()));
        }
        inp.close();
        fis.close();

        // subsample and subarea
        if (subarea != null) {
            try {
                result = result.crop(subarea);
            } catch (ImageException exc) {
                logger.debug("Could not crop file.", exc);
                throw(new IOException(exc.toString()));
            }
        }

        if (sampling != 1) {
            try {
                result = result.scale(sampling);
            } catch (ImageException exc) {
                logger.debug("Could not sample file.", exc);
                throw(new IOException(exc.toString()));
            }
        }

        return result;
    }
```

```
    /**
     * Return the description of the reader.
     *
     * @return decription of the reader
     */
    public String getDescription() {
        return "Java Serialized Loader";
    }
}
```

ImageWriter

To be able to write a new file format the ImageWriter interface needs to be implemented. The example gives an implementation of an ImageWriter that can read serialized ImageObjects to disk.

The ImageLoader will first call canWrite with the filename. Based on the filename the writer should decide if it can write the ImageObject to disk. Most images will have specific extensions that identify what type of image it is, for example GIF files start use gif.

Next writeImage is called to write the image to disk.

```
import ncsa.Im2Learn.core.datatype.ImageObject;
import ncsa.Im2Learn.core.io.ImageWriter;

import java.io.*;

/**
 * Write a serialized ImageObject to disk. This class is responsible
 * for serializing an ImageObject and writing the bytes to disk that
 * make up the ImageObject.
 */
public class ObjectLoader implements ImageWriter {
    /**
     * Returns true if the class can write the file.
     *
     * @param filename of the file to be written.
     * @return true if the file can be written by this class.
     */
    public boolean canWrite(String filename) {
        return filename.endsWith(".object");
    }

    /**
     * This function will write the imageobject to a file.
```

```
     *
     * @param filename    of the file to be written.
     * @param imageobject the image image to be written.
     * @throws java.io.IOException if the file could not be written.
     */
    public void writeImage(String filename, ImageObject imageobject)
                                                    throws IOException {
      FileOutputStream fos = new FileOutputStream(filename);
      ObjectOutput out = new ObjectOutputStream(fos);
      out.writeObject(imageobject);
      out.close();
      fos.close();
    }

    /**
     * Return a list of extentions this class can write.
     *
     * @return a list of extentions that are understood by this
     *         class.
     */
    public String[] writeExt() {
        return new String[]{"object"};
    }

    /**
     * Return the description of the writer.
     *
     * @return decription of the writer
     */
    public String getDescription() {
        return "Java Serialized Loader";
    }
}
```

Example Plugin

This appendix shows how to build a plugin.

```
package ncsa.Im2Learn.ext.test;

import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;

import java.net.URL;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import ncsa.Im2Learn.core.datatype.ImageObject;
import ncsa.Im2Learn.core.display.*;

// ----------------------------------------------------------
public class ExamplePlugin extends IM2LEARNFrame implements IM2LEARNMenu,
HelpEntry {
    /**
     * Reference to the imagepanel this menu is associated
     * with.
     */
    private ImagePanel imagepanel;

    /**
     * Preview of the swap operation.
     */
    private ImagePanel ipPreview;

    /**
     * Logger used for debug and warning messages.
     */
    static private Log logger = LogFactory.
                                getLog(ExamplePlugin.class);

    /**
     * Create the UI for this menu. Show a preview panel in the
     * middle of the frame and buttons below to do the swap,
     * apply changes to imagepanel and close the window.
     *
     */
    public ExamplePlugin() {
```

```
    super("COLOR SWAP");

    imagepanel = null;
    createUI();
}

/**
 * Called when the user hides the window.
 */
public void hiding() {
    ipPreview.setImageObject(null);
}

/**
 * Called when the user shows the window.
 */
public void showing() {
    reset();
}

/**
 * Create the UI. The UI consists of a preview imagepanel
 * which will show the swapped image, and three buttons to
 * swap the image, apply changes to the imagepanel and close
 * the window.
 */
private void createUI() {
    ipPreview = new ImagePanel();
    ipPreview.setAutozoom(true);
    getContentPane().add(ipPreview, BorderLayout.CENTER);

    JPanel pnlButtons = new JPanel(new FlowLayout());
    pnlButtons.add(new JButton(new AbstractAction("Swap") {
        public void actionPerformed(ActionEvent e) {
            swap();
        }
    }));
    pnlButtons.add(new JButton(new AbstractAction("Apply") {
        public void actionPerformed(ActionEvent e) {
            if (ipPreview.getImageObject() == null) {
                swap();
            }
            ImageObject imgobj = ipPreview.getImageObject();
            imagepanel.setImageObject(imgobj);
        }
    }));
    pnlButtons.add(new JButton(new AbstractAction("Close") {
```

```
        public void actionPerformed(ActionEvent e) {
            ExamplePlugin.this.setVisible(false);
        }
    }));
    getContentPane().add(pnlButtons, BorderLayout.SOUTH);

    pack();
}

/**
 * Set the image shown in the preview to null.
 */
private void reset() {
    ipPreview.setImageObject(null);
}

/**
 * Change the colors of the imageobject. This will take
 * the current maximum color and subtract the color of
 * the pixel.
 */
private void swap() {
    ImageObject imgobj = imagepanel.getImageObject();
    if ((imgobj == null) || !imgobj.isDataValid()) {
        return;
    }

    // create a copy that we work with
    ImageObject clone = null;
    try {
        clone = (ImageObject) imgobj.clone();
    } catch (CloneNotSupportedException exc) {
        logger.warn("Could not clone image.", exc);
        return;
    }
    double max = clone.getMax();

    // change all pixels, see Appendix B for more
    // efficient methods
    for (int i = 0; i  clone.getSize(); i++) {
        clone.setDouble(i, max - clone.getDouble(i));
    }

    // set the preview pane
    ipPreview.setImageObject(clone);
}
```

```
// ------------------------------------------------------------
// IM2LEARNMenu implementation
// ------------------------------------------------------------
/**
 * Store the reference to the imagepanel for later use.
 * The imagepanel passed in as argument is the
 * imagepanel to which this tools is associated.
 */
public void setImagePanel(ImagePanel imagepanel) {
    this.imagepanel = imagepanel;
}

/**
 * For this tool there is no operation that works on the
 * imagepanel directly and thus we return null
 * indicating that no menu entries need to be created on
 * the panel.
 */
public JMenuItem[] getPanelMenuItems() {
    return null;
}

/**
 * Create a menu entry called Tools, and attach a
 * submenu entry to this for this class, called swap. If
 * the user selects this class
 */
public JMenuItem[] getMainMenuItems() {
    JMenu menu = new JMenu("Tools");

    JMenuItem item = new JMenuItem(new AbstractAction("Swap") {
        public void actionPerformed(ActionEvent e) {
            if (!isVisible()) {
                Window win = SwingUtilities.
                             getWindowAncestor(imagepanel);
                setLocationRelativeTo(win);
                setVisible(true);
            }
            toFront();
        }
    });
    menu.add(item);

    return new JMenuItem[] { menu };
}

/**
```

```
     * When a new image is loaded make sure that the current
     * swapped image is removed, i.e. perform a reset. Only
     * need to reset the image if this frame is visible.
     */
    public void imageUpdated(ImageUpdateEvent event) {
        if (!isVisible()) {
            return;
        }
        if (event.getId() == ImageUpdateEvent.NEW_IMAGE) {
            reset();
        }
    }

    // ------------------------------------------------------------
    // HelpEntry implementation
    // ------------------------------------------------------------
    /**
     * Create a node for the help system called Tools, and a
     * leafnode called Swap. The leafnode will contain the
     * documentation about this function.
     */
    public HelpTopic[] getTopics() {
        HelpTopic topic = new HelpTopic("Tools");
        new HelpTopic("Swap", topic);
        return new HelpTopic[] { topic };
    }

    /**
     * Return the right documentation depending on what node
     * is selected, in this case only the Swap node should
     * exist.
     */
    public URL getHelp(String menu) {
        if (menu.equals("Swap")) {
            return getClass().getResource("help/swap.html");
        } else {
            return null;
        }
    }
}
```

Speeding up Im2Learn

The generic method of accessing the data is slowest (about 4 times). You can speed things up by adding a simple switch statement to typecast the generic object to the correct type object. Having seperate loops for each type will speed this even more up. Fastest is to do a getData() typecast the result to the correct type and access the array directly. However this is also the most code and most errorprone, while generic methods is the least code and less errorprone. Best is to use the generics while developping code, and switch to specifics when optimizing the code.

For example when the loop size is 7500000

Checking method	time (ms)	vs direct
Testing direct	151.6ms	100%
Testing function	181.2ms	120%
Testing image type	204.7ms	135%
Testing image check	225.0ms	148%
Testing image generic	435.9ms	288%

As you can see doing a simple check inside the loop, what imagetype it is and typecasting the ImageObject to this this type will speed things up. The reason for the slowdown, is the fact that the abstract class will need to do a runtime check to see what class it really is and call the right function.

So instead of using:

```
    ImageObject imageobject;
    for(i = 0; i < size; i++) {
        imageobject.set(i, 0);
    }
```

use:

```
ImageObject imageobject;
for(i = 0; i < size; i++) {
    switch (imageobject.getType()) {
        case ImageObject.TYPE_BYTE:
            ((ImageObjectByte)imageobject).set(i, 0);
            break;
        case ImageObject.TYPE_SHORT:
            ((ImageObjectShort)imageobject).set(i, 0);
            break;
        case ImageObject.TYPE_INT:
            ((ImageObjectInt)imageobject).set(i, 0);
            break;
        case ImageObject.TYPE_LONG:
            ((ImageObjectLong)imageobject).set(i, 0);
            break;
        case ImageObject.TYPE_FLOAT:
            ((ImageObjectFloat)imageobject).set(i, 0);
            break;
        case ImageObject.TYPE_DOUBLE:
            ((ImageObjectDouble)imageobject).set(i, 0);
            break;
    }
}
```

of course better is:

```
switch (imageobject.getType()) {
    case ImageObject.TYPE_BYTE:
        ImageObjectByte imgbyte = (ImageObjectByte)imageobject;
        for(i = 0; i < size; i++) {
            imgbyte.set(i, 0);
        }
        break;
    ....
    case ImageObject.TYPE_DOUBLE:
        ImageObjectDouble imgdouble = (ImageObjectDouble)imageobject;
        for(i = 0; i < size; i++) {
            imgdouble.set(i, 0);
        }
        break;
}
```

finally nothing beats:

```
    switch (imageobject.getType()) {
        case ImageObject.TYPE_BYTE:
            java.util.Arrays.fill((byte[])(imageobject.getData()), (byte)0);
            break;
        ....
        case ImageObject.TYPE_DOUBLE:
            java.util.Arrays.fill((double[])(imageobject.getData()), (double)0);
            break;
    }
```

Basicly what I am trying to say is, code smart and the speed will follow. If possible use built-in java code like System.arraycopy (which is a single assembly instruction on a x86).

Software License

The software is owned by the University of Illinois at Urbana-Champaign.

Developed by: Image Spatial Data Analysis Group (ISDAG) http://isda.ncsa.illinois.edu/

National Center for Supercomputing Applications (NCSA) http://www.ncsa.illinois.edu/