Im2Learn Manual

Version 0.1

Im2Learn (Image to Learn) was created at the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. We would like to acknowledge multiple funding agencies for the support including NSF, NIH, NASA, NARA, ONR, DARPA, NAVY and NCSA Industrial Partners. The main creators of Im2Learn are Rob Kooper, David Clutter, Sang-Chul Lee and Peter Bajcsy with the help from graduate and undergraduate students, namely Wei-Wen Feng, and Peter Ferak. This document represents a current description of multiple on-going research and development efforts and hence it is updated on a regular basis.

Revision History

Revision	Date	Author	Notes
0.1	9/16/2005	RK,PF,PB	Initial version of the document
0.2	10/31/2006	SL	HSV Threshold added under Tool menu

Table of Contents

Introduction

Chapter 1 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.

1.1 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.

1.2 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.

1.3 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.

1.4 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.

- 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

Chapter 2 How to use Im2Learn

Im2Learn is consists of a core set of functions and data structures, and extensions (or plugins) that represent solutions for specific tasks. This document will first describe details about the core parts of Im2Learn and next outline the extensions that are available at the time of writing.

2.1 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.

2.1.1 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.

2.2 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.

2.2.1 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 Appendix A, Example Plugin.

2.2.1.1 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();
  }
}
```

2.2.1.2 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;
  }
}
```

2.2.2 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.

2.2.2.1 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";
    }
}
```

2.2.2.2 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 either `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";
    }
}
```

Appendix A 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;
        }
    }
}
```

Appendix B Speed Improvements

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 separate loops for each type will speed this even more up. The fastest implementation is to typecast the result to the correct type using getData() and then access the array directly. However, this leads to a large number of code lines and is the most error-prone, while generic methods are the least code and less error-prone. An optimal approach to coding is to use the generics while developing code, and switch to specifics when optimizing the code.

Following is a sequence of code excerpts where the ImageObject is modified. The examples will go from simply using the generic getter and setter methods of the ImageObject where the developer does not need to know about specific types of ImageObject, to the very specific version where the underlying datastructure is modified. The advantage of the generic get and set methods is that if a new imagetype is introduced the generic methods will still work, but the optimized code will not work with the new imagetype.

In the first example we use the generic getter and setter methods. The developer does not need to know about how the data is stored and what type the data is. By using the getDouble method we guarantee ourselves that it will work with any datatype, since double can represent the largest numbers. The code uses the abstract class ImageObject for the function calls. When the code is executed the runtime system will find the class implementing the functions and call those functions. Inside those functions might be a typecast to convert the data from the underlying datastructure to the more generic double type.

```
ImageObject imageobject;

switch (imageobject.getType()) {
    case ImageObject.TYPE_BYTE:
        ImageObjectByte imgbyte = (ImageObjectByte)imageobject;
        byte            valbyte;
        for(i=0; i<size; i++) {
            valbyte = imgbyte.getByte(i);
            imgbyte.setByte(i, valbyte + 1);
        }
        break;
    ....
    case ImageObject.TYPE_DOUBLE:
        ImageObjectDouble imgdouble = (ImageObjectDouble)imageobject;
        double            valdouble;
        for(i=0; i<size; i++) {
            valdouble = imgdouble.getDouble(i);
            imgdouble.setDouble(i, valdouble + 1);
        }
        break;
    default:
        throw(new ImageException("Unknown imageformat.");
}
```

Finally we can ask the ImageObject for the underlying datastructure and modify it directly. This is done by retrieving the array from the ImageObject using `imageobject.getData()`. This will return an array with the values of the 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.). If the data is modified using this array the class needs to be notified of this and the function `imageobject.getMinMaxInvalid()` needs to be called. This will make the class recalculate the minimum and maximum values of the image when either `imageobject.getMin()` or `imageobject.getMax()` is called or the image is drawn in an imagepanel.

```
ImageObject imageobject;

switch (imageobject.getType()) {
    case ImageObject.TYPE_BYTE:
        byte[] bytearr = (byte[])imageobject.getData();
        for(i=0; i<size; i++) {
            bytearr[i] += 1;
        }
        break;
    ....
    case ImageObject.TYPE_DOUBLE:
        byte[] bytedouble = (double[])imageobject.getData();
        for(i=0; i<size; i++) {
            bytedouble[i] += 1;
        }
        break;
    default:
        throw(new ImageException("Unknown imageformat.");
}
imageobject.setMinMaxInvalid();
```

Appendix C Plugins

This appendix provides a user guide to all plugins that have been documented so far. There are many more tools that we have not ad the time to document and are available only in the on-line help.

The current document contains groups of tools labeled as Edit, Panel, Tools, Hyperspectral, and Segment.

C.1 Edit

C.1.1 Copy

This will allow the copying of the image to the system clipboard. The image that will be copied is the image that is currently visible and is depended on bands for red, green and blue; gamma; crop etc..

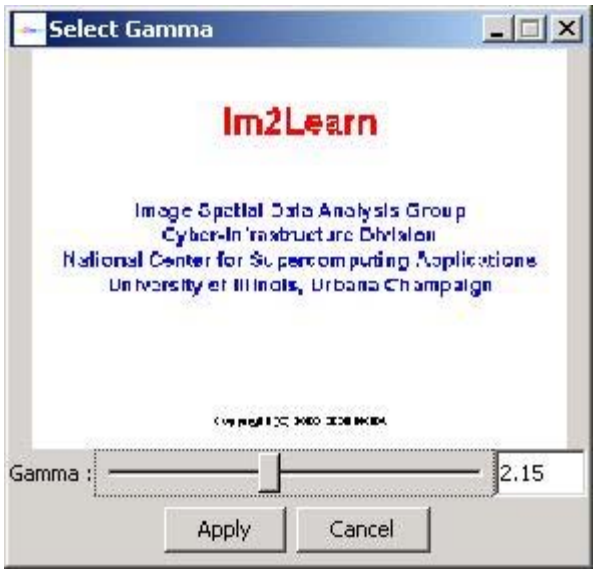
C.1.2 Paste

This will allow the pasting of the image from the system clipboard. The resulting image will be a 3 band image of type BYTE.

C.2 Panel

C.2.1 Class GammaDialog

If an image is over or under saturated some details might not be visible. Fortunatly often this over/under saturation can be compensated by using gamma control. Gamma adjustment allows you to brighten the whole image. The formula used is: `newvalue = oldvalue^(1/gamma)`. For gamma values larger than one this will result in a brighter image, allowing you to see details in darker regions of the image. For gamma values less than one this will result in a darker image, allowing you to see details in parts of the image that are oversaturated.



The dialog allows a user to specify the gamma value by either entering it in the text field, or adjusting the slider bar. To see the result, a user can press the preview button which will apply the gamma correction and show a preview of the image. If the image is not satisfactory then a user can modify the gamma value and check again using preview. If the result meets the desired quality then by pressing the "Apply" button a user will apply the gamma correction to the currently visible image.

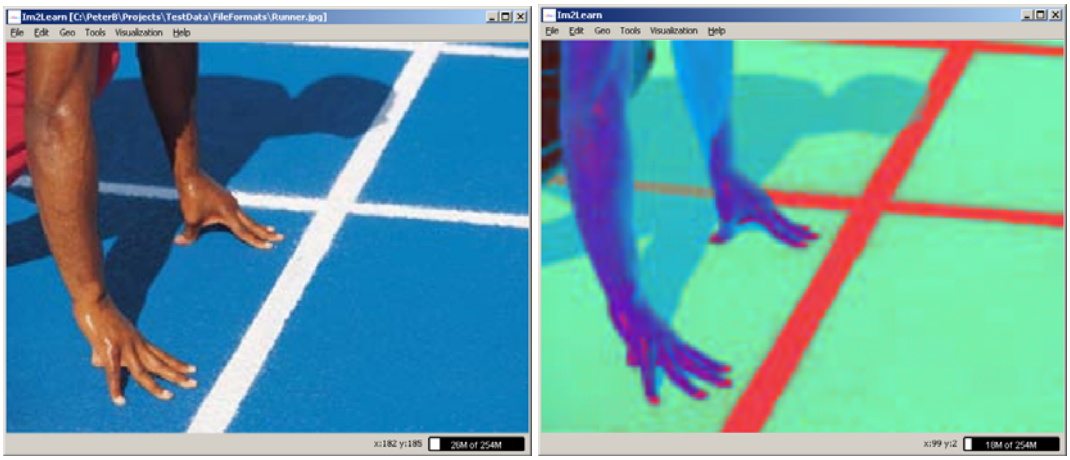
C.2.2 Class ImagePCA

The class ImagePCA provides a tool for performing principal component analysis (PCA) of images. This functionality is available from the main menu in IM2LEARN under features, and will be applied immediately to the current image in the main IM2LEARN frame. If the current image was obtained by forward PCA then it will be converted back to the orifinal image without any loss of information.

Description: The forward PCA of an image is calculated by first creating a correlation matrix of the image. This matrix will show how each band is related to each other band. The PCA code will use the correlation to compute a new transformation based on eigenvectors. This new foward transformation matrix will convert the original image into a PCA image. Each of the bands in the new PCA image will contain information from all the bands of the original image. The first band in the PCA image has the largest contribution from the original image and each additional band in the PCA image has smaller contribution from the original image.

To recover the original image from the PCA image, coefficients of the transformation matrix ("eigenvectors") are stored in image properties, as well as, the type of the original image ("originaltype"). These image properties are used when reverse PCA is being executed.

An example of a PCA image transformation is shown below for the "Runner" image.



This class also provides utility functions to calculate a covariance of an image, calcCovarianceMatrix, as well as, the correlation, calcCorrelationMatrix.

C.2.3 Select Band

This dialog allows you to select which bands in the image are represented by red, green, blue and gray. The gray band is used if the image is shown as grayscale, otherwise the red, green and blue bands are used to draw the image in color.

If the checkbox in front of red, green or blue is unchecked, the image will be drawn without that particular band. For instance unchecking both the green and blue checkbox will show the image with just the red band and makes it easy to see how much red there is in the image. Unlike the gray scale image the image will now be drawn with just the red band.

Changing any of the options will modify the small preview image, giving you immediate feedback on the changes you made to the image.

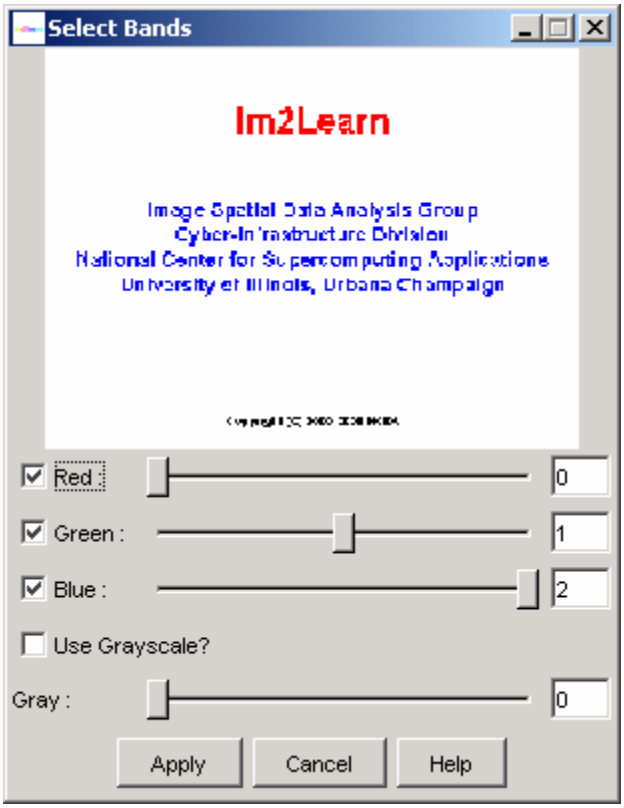


Figure: Select Band Dialog.

C.2.4 Sub-area selection

This option enables to select a sub-area of interest. An area is selected by left mouse clicking and dragging to establish a rectangular window. With the right mouse button, choose "Selection->Show Selection" to display the area of interest. You might use the zoom option to increase the viewed resolution. The full image can be retrieved by choosing "Selection->Full Image" (see Figure). The sub-area selections can be disabled by un-checking the box "Selection->Allow Selection".

Figure : Sub-area selection menu.

C.2.5 Image Zoom

This option change digital resolution of an image. The option "fit" will automatically rescale an image to fit the image panel size. The option "custom" lets a user define the scaling factor either as a multiplicative factor or as a percentage (by appending % to the number) of the original image size. For example using the number 2, or 200% will result in an image that is twice the size of the original.

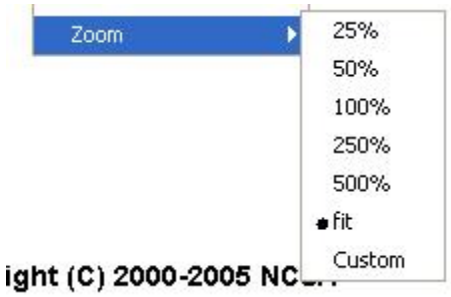


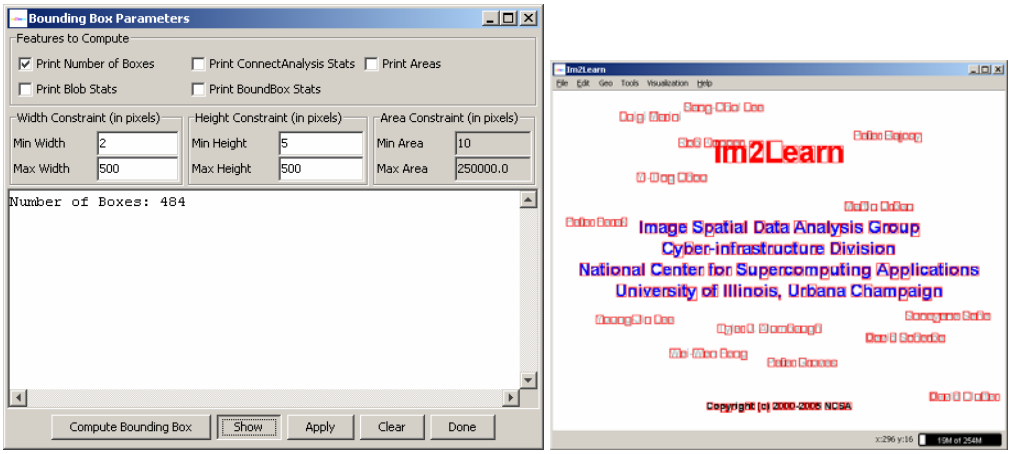
Figure: Zoomselectionmenu.

C.3 Tools

C.3.1 Bound Box

The class BoundBoxDialog provides a graphical user interface (GUI) to the class BoundBox designed for drawing rectangles around areas in an image. The Dialog provides parameters which restrict the showing of undesired Bounding Boxes. For instance noise due to bad quality of images. The parameters are set to a minimum of 5 pixels by 5 pixels and a maximum which should be high enough not to be exceeded by most images (it's the maximum Integer value). The Area is displayed on the right of the width and height parameters and is merely intented as a help for the user. Therefore and especially small or large Bounding Boxes can be filtered out.

Example:

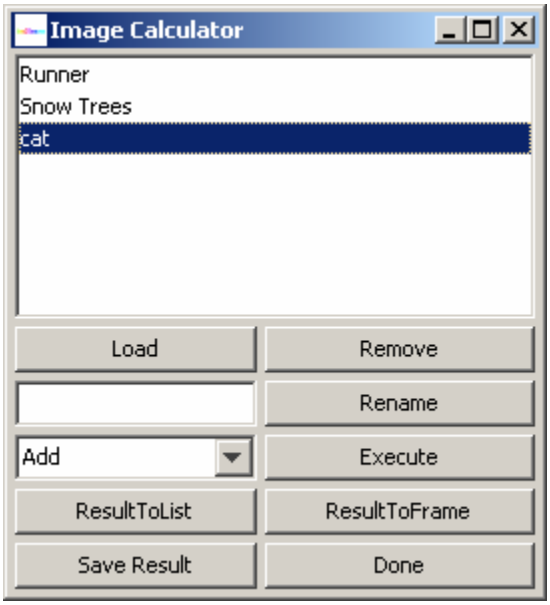


This tool is used to find the text areas inside of a non-PDF file. There is still some noise which might be removed with the use of other tools. But most texts are picked up nicely, unless the font is too small.

C.3.2 Image Calculator

The class ImCalculator provides a tool for performing algebraic and boolean operations with images. The algebraic operations are image addition, subtraction, inversion, multiplication, division and average. The boolean operations include AND, OR and XOR.

Software License



Description: The methods supported by ImageCalculator class operate on images at the level of each image pixel. For example, two color images with red, green and blue bands will be added by summing spatially corresponding pixel values in each band. While the number of bands and data type must be consistent in both input images, the image size can vary. The result image will have the spatial dimension based on the minimum of input image dimensions. If the result of any algebraic operation leads to a value that is outside the bounds of the input data type, then all resulting values are scaled so that the output image contains data of the same data type as the input images, unless they are not of the same type. In this case it will create the output of the type of the image that is bigger, with double being the biggest and byte being the smallest.

The ImageCalculator performs the following algebraic and boolean operations: Addition, Subtraction, Inversion, Multiplication, Division, Average, Boolean AND, Boolean OR, Boolean XOR

Setup: In order to execute any operation, images have to be loaded using the Load button. Visualization of the loaded image is achieved by double clicking on the image name in the top area.

Run: An operation to be performed should be chosen from the dropdown menu (Add, Subtract, etc). By pressing the "Execute" button the calculation will take place and the result will be shown in a new frame. The resulting image can be put into the mainframe by utilizing "ResultIntoFrame"

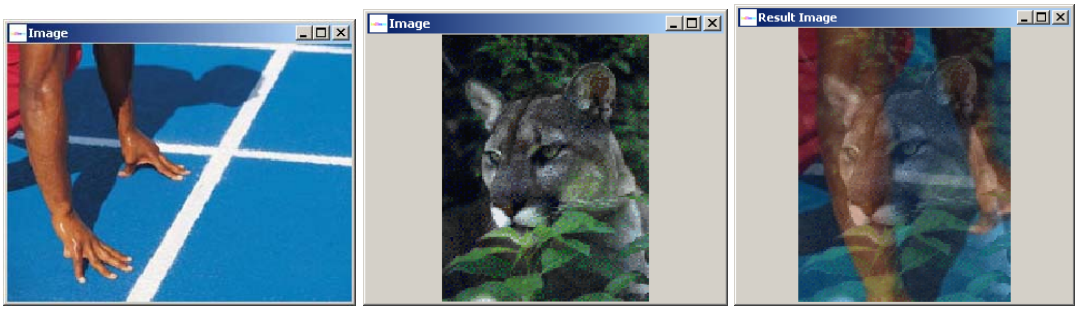
The summary of the additional buttons is provided next:

Load Image - Loads an image.

Execute - The chosen action from the ComboBox will be performed with the images.

Save Result - saves the result image.

Example ADD:



C.3.3 Image Compare

Even though to the human eye two images look the same there can be some minute differences, this dialog will allow the user to select a comparison method to compare two images and visualize the minute differences between the images. The tool will allow the user to load a test image, a mask (or use a predefined mask) and test method, as well as a multiplier for the result image and the type of the result image.

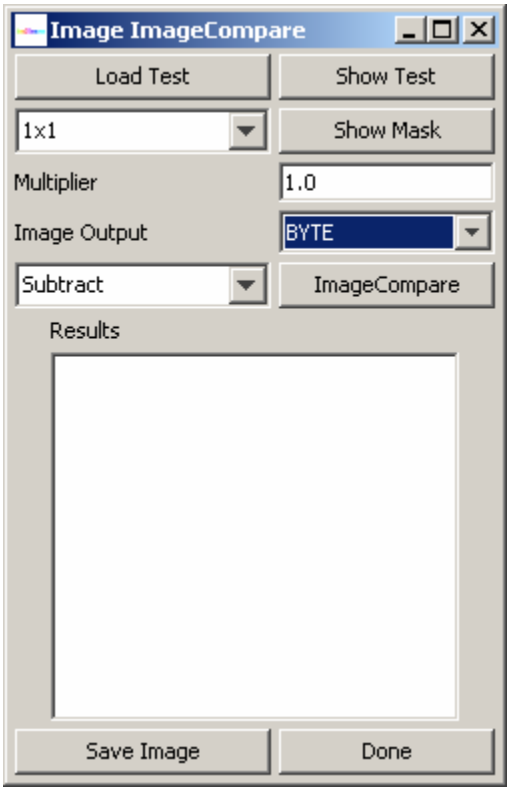


Figure: Image Compare Dialog.

The dialog shows all these options. To perform a comparison the user must first select an image to compare the current image in the mainfram against using the "Load Test" button. This will show a dialog allowing the user to load an image. To make sure the right image is loaded the user can show the test image using the "Show Test" button.

Next the user has to decide which mask to use. The user can use one of the predefined masks or load a custom mask. The predefined mask will create a mask image that splits the original image in squares of the selected size, from 1x1 (pixel) to 100x100 size squares. If the user selected "Load Mask" a dialog will appear allowing the user to load an image to be used as a mask. Only the first band of this mask image wil be used! The user can check the mask by clicking on the "Show Mask" button. Some masks however don't work well with certain comparison methods (for instance pixel (or 1x1) does not work with histogram test.

Next the user has to select which test to use. The user can choose one of the following test:

Subtract

For each region and band, as defined by the mask, this routine will calculate the average for both the original and the test and subtract them from each other. The result image will contain the difference in that region. Setting the mask to 1x1 will result in the difference per pixel for the image and test image.

Correlation

This will calculate the Pearson linnear correlation for each band and mask value. Currently only the pearon value is calculated correctly. If the mask is set to pixel ((1x1) it will compute the correlation per band over the whole image. There is no resulting image.

Histogram

For each band and region in the mask image, we create a histogram of all values. After normalization we calculate the percentage of pixels that are placed in one bin, but should be in another. The resulting image will have this percentage.

Chi Square RGB and Chi Square HSV

This will compute the CHI-Square value on a per band and mask region. The resulting CHI-Square value will be returned in the result image. We have observed that for images with small changes (i.e. a red scratch in the image) the CHI-Square value will be large enough to result in a probability of 0.

Tuple

This will create a subimage by taking the 3 highest bits of the first, second and the 2 highest bits of the third band and combine them into a 8 bit byte. Next it performs a CHI-Square test for each mask region and band on this subimage and the subimage created from the testimage in simmilar way. The result image will contain the CHI-Square value for each mask region.

Long Chi Square RGB

This is the same as Tuple except that instead of only using a few bits we concatenate the first, second and third byte of both images and perform a CHI-Square test on each of these new images. The resulting CHI-Square value will be put in the resulting image.

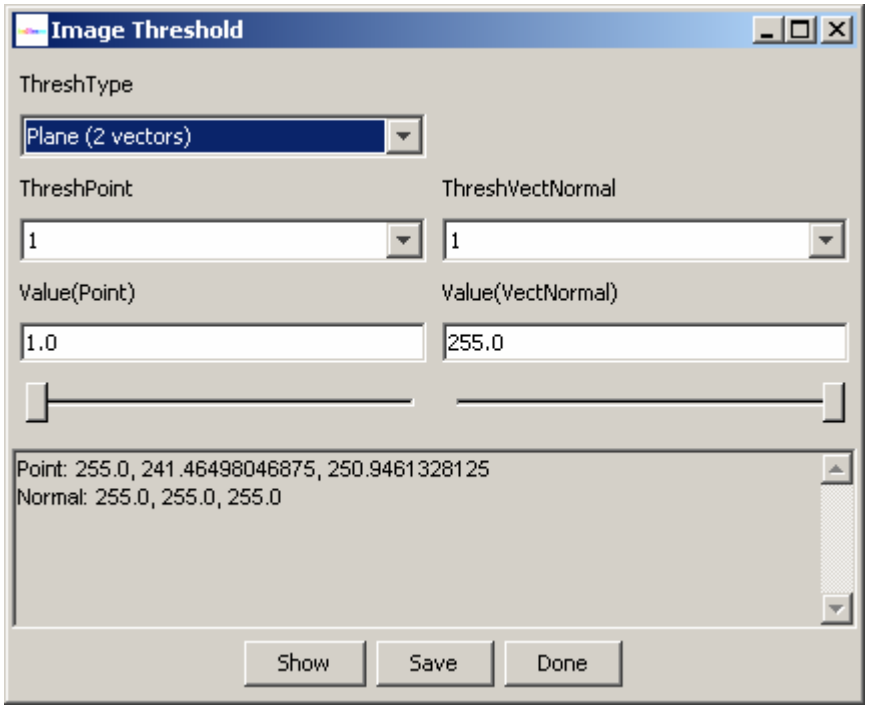
Finally the user can use the Multiplier and Image Output fields to select what imagetype the resulting image will be and what multiplier needs to be used when comparing the images. For instance to create an BYTE image of the Histogram we need a multiplier of 255 since the original range of values will be between 0 and 1.

The user can save the result of the comparison as an image using the "Save Image" button.

C.3.4 Treshold

The class Threshold is a tool for two-class clustering problems using Euclidean distance, a hypercube or a hyper plane for separating clusters in high-dimensional space.

Description: The class operates in three distinct modes that are set in the ThreshType choice list. In these three modes, points in high-dimensional space are classified based on (1) their Euclidean distance from the origin (a scalar value), (2) a hypercube (a vector) or (3) a hyper plane (two vectors).

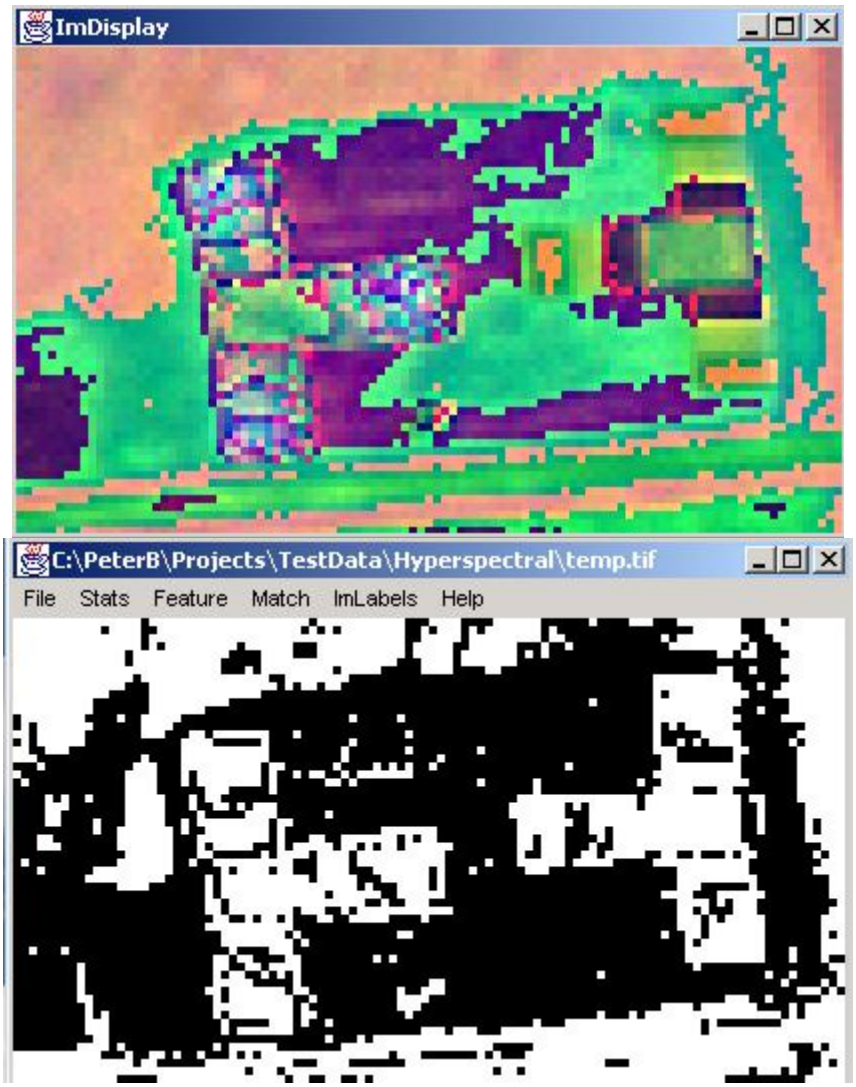


Setup: In the "Distance (scalar)" mode, the scalar value is set either in the edit box or by moving the slider below the edit box denoted as Value(Point). The value is also shown in the text area together with the outcome of the thresholding operation. The outcome of the thresholding operation is the number of points below the threshold denoted as "BlackCount" and above the threshold denoted as "WhiteCount".

In the "Box (vector)" mode, it is possible to set the upper corner of a hypercube by selecting one of the point dimensions in the choice list labeled as "ThreshPoint". The coordinate can then be modified either in the edit box or by moving the slider. The upper corner of a box (hypercube) is defined as the point that encloses the maxima of all points denoted as "Black". The lower corner is always set to the minimum coordinate in each dimension.

The last option "Plane (2 vectors)" of the "ThreshType" choice list defines a hyper plane by setting one point in the plane (ValuePoint) and one point as the tip of the normal to the plane (ThreshVecNormal). The settings can be modified either in the edit boxes or by moving the sliders.

Run: The text area reports input parameters (a scalar, one vector or two vectors) and output parameters (BlackCount and WhiteCount). It is possible to view the thresholded images by clicking "Show" and save them by clicking "Save". Any changes in the input parameters will be automatically updated in the frame showing the thresholded images. Although the default values for each mode correspond to statistical thresholds (e.g., the sample mean of the image), further input parameter tuning is facilitated by on-line visualization of the thresholded image with the "Show" button.



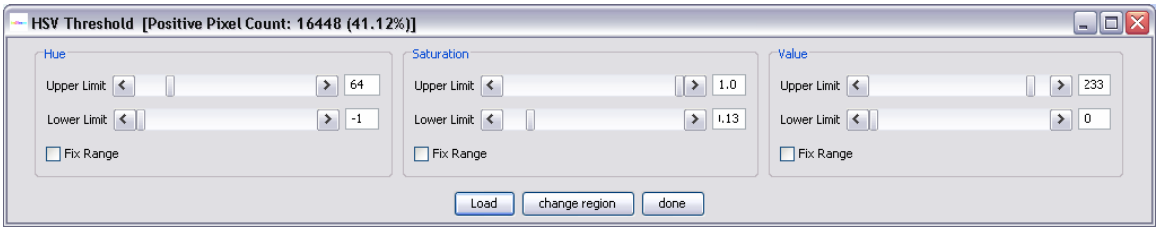
Release notes: Java sliders support only integer data type therefore any double precision thresholds must be entered via edit boxes. The value that is shown in the edit boxes will be the actual value used for thresholding.

C.3.5 HSV Treshold

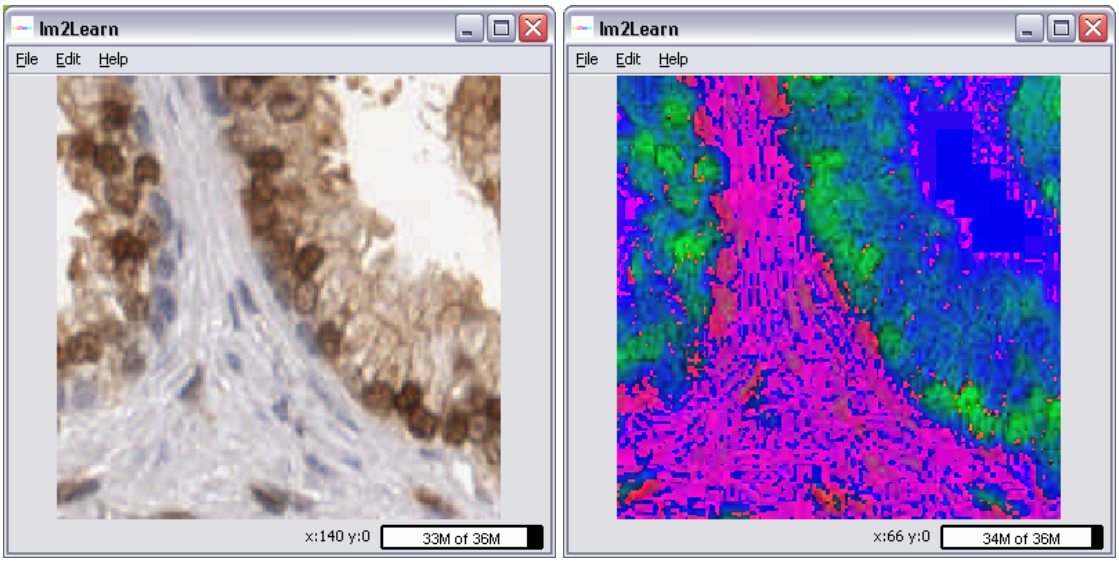
The class HSVThreshold is a tool for clustering problems in HSV (Hue, Saturation, and Value) color space for color-based pixel separation. The main purpose of this tool is to support semi-automated color-based segmentation. Particularly, this tool provides users the decision support about valid pixel value ranges for specific types of object detection.

Description: The class operates in three ranges of HSV values. Points within the user-specified value ranges are appeared in the result image, both as RGB and HSV images. In each scrollbar, a user can adjust the upper and lower limits of the pixel values, and can fix the dynamic value range by checking the “Fix Range” option. For example, the lower/upper limit

will move automatically while keeping the same dynamic value range when changing the upper/lower limit.



Setup: By default, both RGB and HSV image are displayed from the main Im2Learn frame. The two windows dynamically show the classified images when a user adjusts the threshold value ranges.



For large image processing, such as “svs” format, a user can directly load the image into the HSV threshold tool independently from the main Im2Learn frame by pressing “Load” button. If a sub-sampled overview image is available, as a same filename with “tif” extension, an overview window will be displayed where a user can interactively select the currently displayed region (marked as red cross).

Run: First, select a sub-region from the overview image to load a 200 by 200 pixel neighborhood from the large image to the RGB and HSV windows. Next, change the Hue, Saturation, and Value range to acquire the thresholded image. The number of valid (displayed) pixels is shown in the title of the tool as "Positive pixel count" with the percentage with respect to the total number of the pixels in the sub-region.

"Change Region" button can be used to load an overview image with different filename. Note that the overview image has to be a 1:200 sub-sampled image from the original svs image, for example, an image sub-sampled by the factor of 200.

C.4 Hyperspectral

C.4.1 Convert RGB

The class HuperSpectralToRGB provides a tool for display and data conversion of hyperspectral images to red-green-blue (RGB) image.

Description: Hyperspectral images contain too many bands to display at the same time as regular monocromatic or color images. For instance, to be able to display a hyperspectral image covering visible spectral range, a user will often have to select three bands that correspond approximately to the red, green and blue wavelengths. Fortunately the wavelength information associated with each hyperspectral image band is known and a user could select the bands of choice for display.

This tool is designed to automatically convert a hyperspectral image to a RGB image by collapsing each of the hyperspectral bands to the RGB bands. This is done by using the algorithm developed by Dan Bruton (http://www.physics.sfasu.edu/astro/color.html). The algorithm takes the wavelength, determines the percentage it contributes to the Red, Green and Blue color, and multiplies it with the intensity to create the RGB image. In the case of

non-visible light (lower than 380 and more that 780nm) the percentage it adds to the RGB values is 0.

This dialog allows users either to choose the existing wavelenths that describe the hyperspectral image (if available) or to specify the wavelengths of the first and last bands and interpolating the wavelength for all the bands in between first and last bands. The dialog will enable both radio buttons "Use existing wavelengths" and "Specify wavelength range" if the loaded hyperspectral image contains information about band wavelengths. Otherwise the option of "Use existing wavelengths" is disabled. To see the selected spectrum range specified by fisrt and last band wavelengths, users can press the "Show Spectrum" button and view the spectrum range visualization (see below).

Once a user is satisfied with the spectrum range, pressing the "Apply" button will convert the current hyperspectral image to a RGB image.

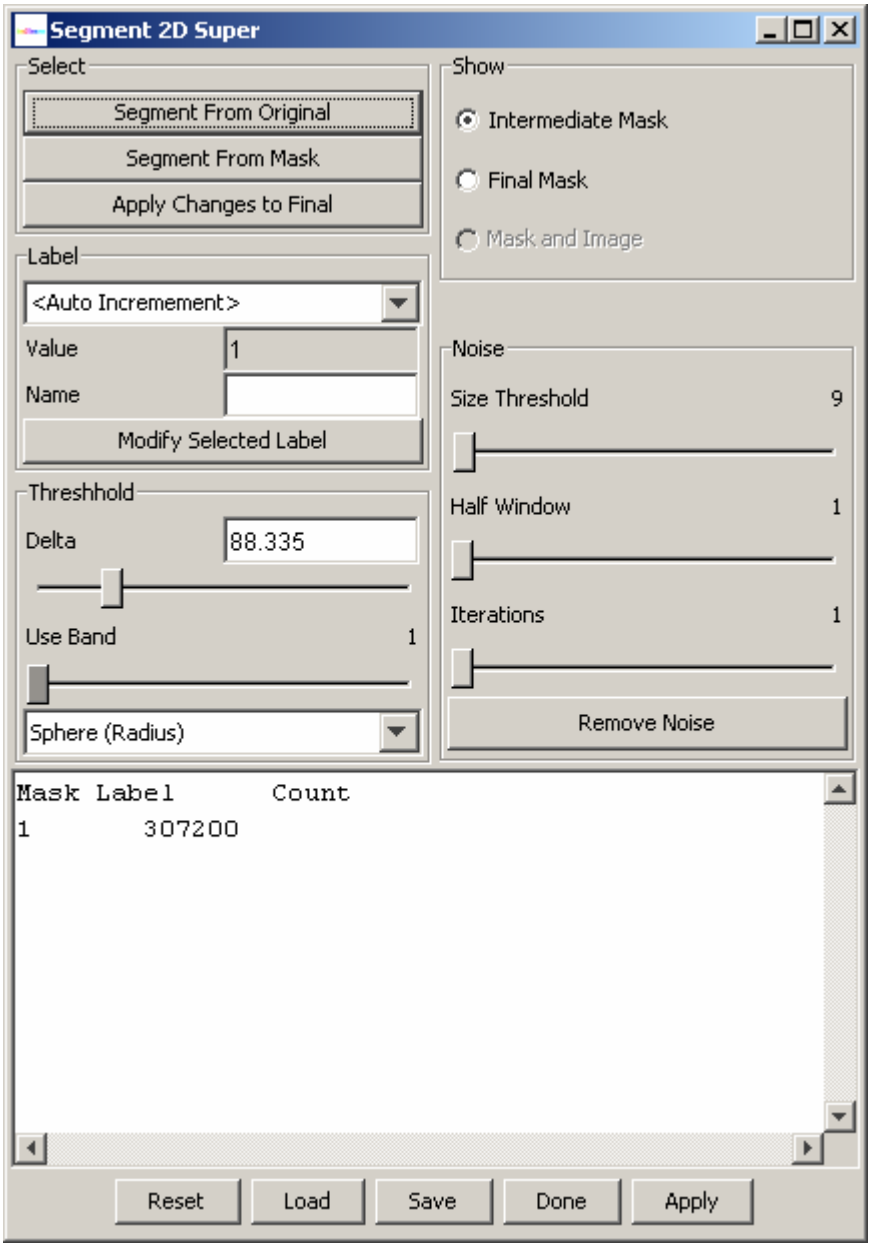
C.5 Segment

C.5.1 Segment 2D Super

The class Seg2DSuperDialog provides a graphics user interface (GUI) to the class Seg2DSuper designed for region growing segmentation tasks given a set of initial spatial locations. The purpose of this tool is to create masks in a supervised way for scene modeling purposes.

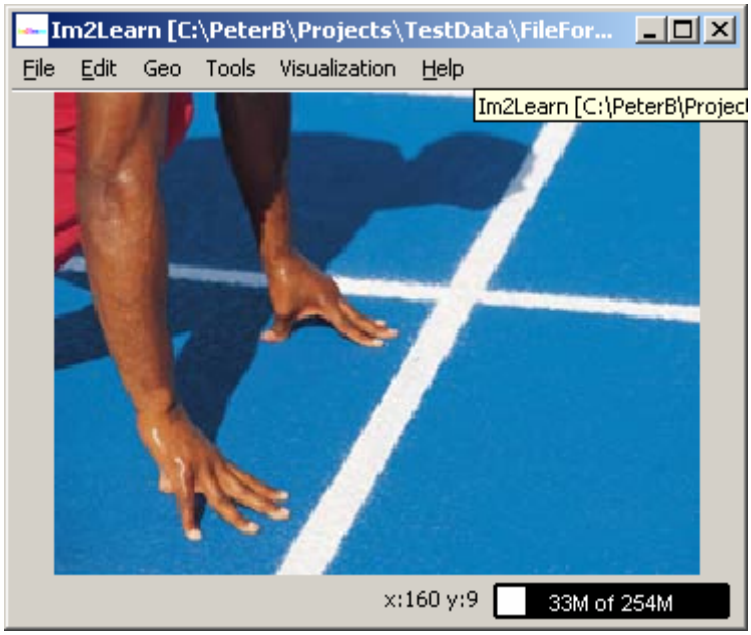
While any unsupervised segmentation algorithm has to estimate the optimal number (or numbers) of image segments, their spatial locations and intensity profile characteristics, this tool allows a user (1) to specify initial spatial locations (seeds) of desired image segments (regions) by a mouse click in the original or mask image, (2) to select minimum similarity of interior pixels forming a contiguous segment, (3) to remove holes from segments that are introduced due to speckle noise or other sharp intensity discontinuities, (4) to assign higher level labels to each formed segment (e.g., grass or sand), and (5) to merge segments that are heterogeneous in terms of intensity profile but belong to the same segment conceptually. Next, we describe how to form a mask and take an advantage of the aforementioned capabilities using this tool.

The dialog of the supervised segmentation tool is shown below.



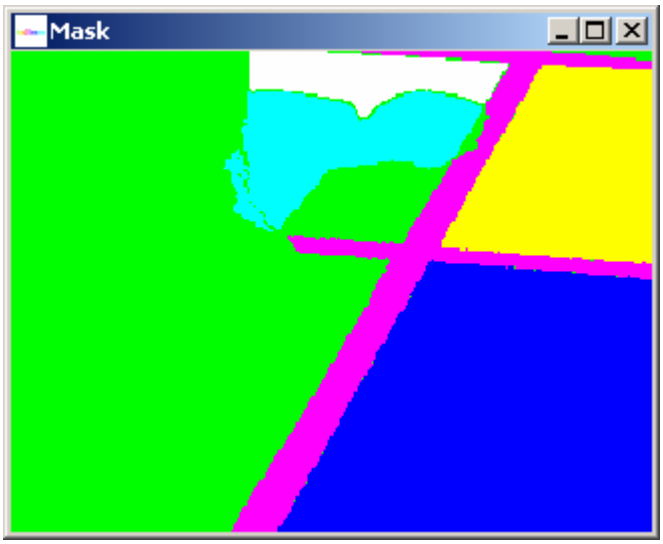
Setup and Run: After launching the dialog, a user will see a new frame displaying the current mask with only one label. By clicking anywhere in the original image and pressing the button "Segment From Original", the region growing algorithm will form a contiguous region (segment) that contains pixels with intensity values less different from the mouse selected one than the value of a threshold. The threshold value can be modified by moving a slider bar denoted as Delta or by entering a new value in the edit box denoted as Value in the Threshold area of the dialog. The distance between any two pixels is computed using a Euclidean distance measure.

Let us assume that we try to segment the image of a runner shown below.



If the obtained segment is satisfactory then it can be added to the final segmentation by clicking on the "Add Change to Final" button. The radio buttons "Intermediate Mask" and "Final Mask" allow switching the mask visualization between the intermediate and final segmentation results.

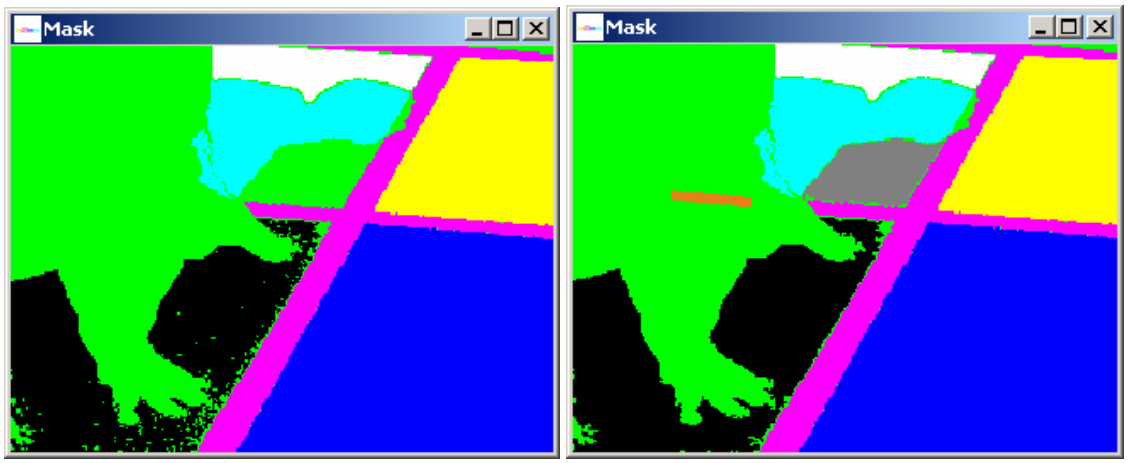
It is possible (and sometimes desirable) to select the region growing origin in the mask image. One would mouse click in the mask image at a selected pixel seed location and then press the button "Segment From Mask" to obtain an intermediate segmentation results starting from the given location. An example of intermediate segmentation is shown below.



A user might want to annotate each segment with its abstract meaning since the abstract meaning is usually very hard to obtain directly from image segment characteristics. This is enabled by typing a name in the "Name" edit box for the associated segment described by a

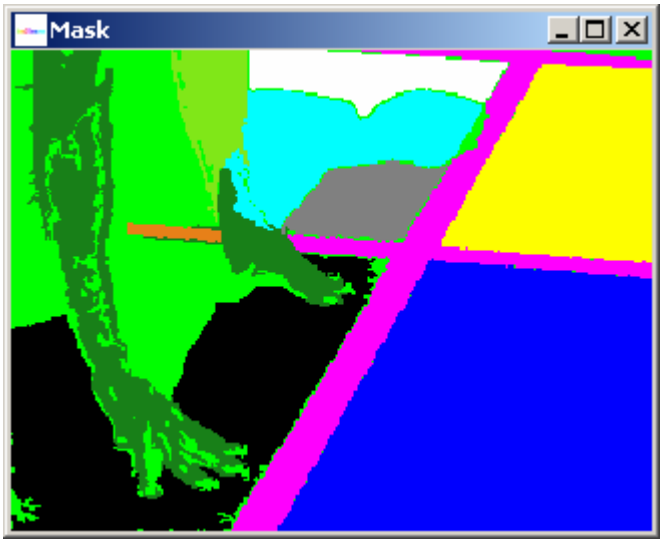
value in the "Value" edit box of the "Label" dialog area. With the drop down menu inside of the "Label" dialog area, one can select already used labels and modified their names. In the "Auto Increment" mode, the values will increase by one and the assigned labels will be set to "unlabeled". Segment names and their pixel counts are reported in the text area as it is shown in the dialog snapshot above.

In the presence of speckle noise, the obtained segments might have undesirable holes since the noise pixels are outside of the allowed range of values and increasing the threshold value would lead to merging multiple segments of different abstract meanings. The "Noise" part of the dialog allows performing noise removal (or removal of segment holes). The "Size Threshold" value denotes the maximum size of a hole that would be considered for removal. The "Half Window" value denotes a half size of a spatial kernel (neighborhood of pixels) that is used for filling the exiting holes of size less than the "Size Threshold" value. This is critical when holes occur at the border of two labeled segments. The "Iteration" slider bar is for choosing the number of iterations that the noise removal step should be applied. Two examples below show a segment with holes and without holes after noise removal.



It is often the case that a desired segment contains heterogeneous intensities and cannot be created by choosing a single threshold and a pixel seed, for example the hands of a runner in the sample image presented above (see obtained labels below).

In this case, merging of multiple segments can be achieved by selecting a label in the "Label" dialog area, choosing a pixel seed location and adding the obtained regions into one segment. Although this procedure can lead to segments that are not spatially contiguous, the formation of such segments is supported because of higher level abstract labels might need such representation.



If the original image contains more than one band then the Euclidean distance threshold for region growing can be replaced by a vector of values containing one threshold per band. This choice of a threshold vector as opposed to a threshold value can be achieved by switching the drop down menu from "Sphere (Radius)" to "Box (Dimension)" in the Threshold area of the dialog. The slider bar "Use Bands" is enabled in the "Box (Dimension)" mode and different threshold values can be set for each band. This option could be used, for instance, if it is desired to obtain a segment from a RGB image that contains red color pixels (red band threshold is set to 0) with a range of green and blue shades (green and blue band thresholds are set to 50).

Finally, the resulting mask image can be saved using the "Save" button. If the output file format is HDF (.h5 suffix) then the label names will be stored as well, otherwise the information will be lost. A mask from previous sessions can be reloaded by clicking the "Load" button for further modifications. If the results during a current session are not satisfactory then the button "Reset" will reset the mask for a new segmentation attempt. Pressing the button "Done" will close the dialog.

Appendix D Software License

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