IFSTOOL, DIGITAL SUPPORT FOR IMAGE ANALYSIS RESEARCH

THE AUTHOR

Ifstool is a GUI which supports the Image Analysis researcher. It supports images of any size[1], any number of dimensions[2], and a wide variety of formats.

The program runs on Linux, Mac OSX, and Windows 7, 8 and 10. Files are interchangeable between platforms. Conversion from 3d images to movies (xxx) is also included. The program can do Fourier analysis, and filtering.

Finally, the program can run an external script, writing an image out, running the script, and reading the script back in, thus providing general image analysis capability.

1. Input File Formats

Internally, Ifstool uses IFS (Image File System). All images are converted into ifs. Ifs supports any computer data type, including complex. Ifstool can read images in any of the following formats

- Portable Network Graphics (PNG, png)
- Joint Photographics Experts Group (JPG, jpg, jpeg)
- Tagged Image File Format (TIFF, tif, tiff)
- Image File Format) (IFS, ifs)
- Graphics Interchange Format (GIF)
- Bitmap File Format (bmp)

2. Organization of Data

If the user has bound the Ifstool to a class of files (e.g. png), then one may simply double click a file and the application will load the file. Or, the user may choose to press the open button. A file select window will open, and the user may select the desired file.

Ifstool maintains eight image buffers, numbered 0-7, as illustrated in Figure 1. Any of the 8 image buffers may be denoted as INPUT1, INPUT2, or OUTPUT. The Open operation will load the target file into the image buffered selected as INPUT1.

The internal buffer will be of type "float" or "complex float", and have the same number of dimensions as the selected input image. If the image is color (as is default for png), it will be three dimensions and have 3 frames. An image may also be three dimensions and not be color. For example "movie loops" may allow the user to play a sequence of images.

[1]Until the process runs out of memory

[2]Also limited by memory capacity

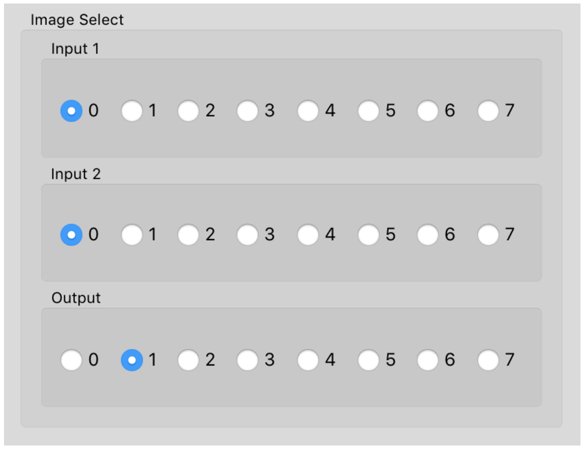


FIGURE 1. Any of eight different images may be used as input or output.

2.1. **Viewing Complex Images.** If the image selected as OUTPUT is complex, the user should select it as INPUT1, and use the COMPLEX pulldown. Operations are listed below:

Magnitude: will compute the magnitude of the INPUT1 buffer and return it in the selected output buffer. This operation requires complex input.

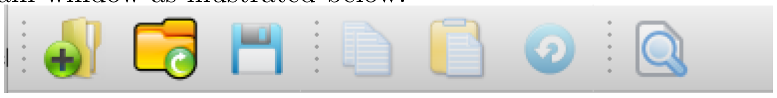
Phase: will compute the phase of the INPUT1 buffer and return it in the selected output buffer. This operation requires complex input.

Real: will extract the real part of the INPUT1 buffer and return it in the selected output buffer. This operation requires complex input.

Imaginary: will extract the imaginary part of the INPUT1 buffer and return it in the selected output buffer. This operation requires complex input.

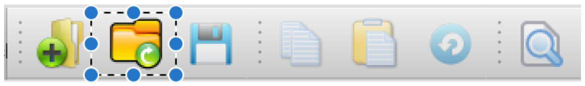
3. File Operations

Most file operations may be accessed by either the file pulldown, or across the top of the main window as illustrated below:



3.1. **Open.** This operation will open a file selection window, allowing the user to choose any data type of image file listed above. It will be read into whichever buffer is marked INPUT1.

4. Reopen



Once a file has been read, Ifstool saves the name of that file in two places, in the program and on the disk. Clicking Reopen will read the file as stored in the archive program, and if that string is empty, it open the file as specified in the file list on disk. Thus, even the first time the program is loaded, using Reopen will load an image (unless, of course, the program has never previously been run).

5. SaveAs

Save the buffer selected as OUTPUT to any data type or format. Including png, jpg, and ifs. If the output data type is ifs, a popup will ask the user to specify the data type. At this time, only the ifs data type is implemented.

6. Copy

Copies a selected output buffer.

7. Paste

Pastes a copied buffer to currently selected output buffer.

8. Viewing an Image

When the *view* button is pushed, the image selected as OUTPUT will be displayed into a new window, called *the viewing window* here. If the image selected as OUTPUT is 2-D or color, the *View* button will display that image. Complex color is not supported.

There are additional functions available on the viewing window. These include:

Choice of Frame Number: A color image may be considered as composed of three images, red, green, and blue. Usually, the user seeks to view the three together as a single color image, and if the color check box is checked, this is the model. However, the user may choose to view each frame independently as a single gray scale image. That view mode is used if the color check box is unchecked.

Images consisting of more than three frames can also be viewed by unsetting the color check box and clicking the frame select button.

Color Model: The appearance of an image may be modified by adding or subtracting yellow or blue to increase the “warmth” or “coolness” of the image. These modifications are only made in the display buffer, and modifying them does not affect the master images, (those stored in buffers 0-7).

Color Enhancement: The appearance of an image may also be modified by adjusting the degree of red, green, or blue in f the image. These modifications are only made in the display buffer, and modifying them does not affect the master images, (those stored in buffers 0-7).

Zoom: Using the zoom slider makes the viewed image larger or smaller, but does not affect the master images, (those stored in buffers 0-7).

Window and Level: Window and level are parameters used to enhance or reduce contrast. The terms *window* and *level* are most meaningful if one considers an image in which the brightness values range between, say, 0 and 1023, but use a display which can only display brightnesses between 0 and 255. The *window* defines the brightness rang which is mapped (linearly) to lie between 0 and 255. For example, that might be brightnesses between 0 and 50. The *level* defines where the window lies. For example, a window with wi

9. Unary (one input) Operations

The input image to all the unary operations is the one selected as INPUT1. The output will be the one selected as OUTPUT.

$\frac{\partial f}{\partial x}$: Computes a fast estimate of the derivative of brightness with respect to x (the column direction).
$\frac{\partial f}{\partial y}$: Computes a fast estimate of the derivative of brightness with respect to y (the row direction).
$\frac{\partial^2 f}{\partial x^2}$: Computes a fast estimate of the second derivative of brightness with respect to x (the column direction).
$\frac{\partial^2 f}{\partial y^2}$: Computes a fast estimate of the second derivative of brightness with respect to y (the row direction).
$\frac{\partial^2 f}{\partial y \partial y}$: Computes a fast estimate of the second derivative of brightness with respect to x and y (the cross term).
Exponential: Computes $\exp(f(x,y)$ at each point in the image.
Inverse:
Laplacian: Computes $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ at each point in the image.
NatLog: Computes $\ln f(x,y)$ at each point.
HistEq: Histogram equalization finds the image whose histogram is as close to flat as possible.
PseudoColor: Given a 2D grayscale image in INPUT1, OUTPUT will be a color image where the brightness of each pixel has been remapped to a color. The user must specify how the mapping occurs using a "colormap." Possible maps are

- 0: gray scale
- 1: inverse gray scale
- 2: Bronson
- 3: Hot metal
- 4: Heated Spectrum
- 5: Log
- 6: Random

The Random map is surprisingly useful when evaluating noise removal algorithms. Because the colors are assigned randomly, if two adjacent points are identical in brightness, they will be displayed as the same color. Therefore large areas of constant color suggest a very smooth image.

The output from colormap is a 3D, 3 frame, unsigned char image. That image may be saved using saveAs.

10. Binary Operations

At this time, the only operations with two inputs are add, subtract, multiply, and divide. They work correctly with both float and complex data types.

11. Frequency Domain Operations

All frequency domain operations are accessed via the COMPLEX pulldown.

fft: will compute the Fourier transform of INPUT1. Internally, this uses the complex two-dimensional fast Fourier transform. Images having dimension which are not a power of 2 will be padded. If the input is real, it will be copied into a complex image first. If the input is three-dimensional (a movie) or color, the fft is only applied to the first frame.

Note that the origin of the FFT will be at 0,0, the upper left of the image. Fourier transforms are often displayed with the origin in the center of the image, showing negative frequencies as positive. If the user is expecting this, the results may be surprising. In particular, instead of the DC term being in the center of the image, it will be at the upper left.

ifft: will compute the inverse Fourier transform of INPUT1. Internally, this uses the complex two-dimensional fast Fourier transform. Images having dimension which are not a power of 2 will be padded. The input image must be complex.

Complex Magnitude: Constructs a float image from a complex image by calculating the magnitude of the complex number.

Complex Phase: Constructs a float image from a complex image by calculating the phase of the complex number.

Real: Constructs a float image from a complex image by extracting the real part of the complex number.

Imaginary: Constructs a float image from a complex image by extracting the imaginary of the complex number.

12. Window and Level

The Window and Level operations allow the user to modify the contrast and brightness in a convenient way, even if the input image has brightnesses which range outside the usual 0 to 255.

Window: is the brightness range which will be mapped to 0-255 on the display. Making this larger or smaller is equivalent to making the contrast lower or higher.

Level: is the location, in brightness, of the center of the window.

Note that if a pixel brightness is below level-window/2 it will be mapped to zero, and if it is above level+window/2, it will be mapped to 255. So the window-level operation has the same result to setting an upper threshold and a lower threshold, and scaling the levels in between to 0-255.