Image Conversion

How To Use

To convert FITS files to JPEG file, use https://bitbucket.org/gsudmlab/mleco-datacollection/src/master/fits2jpeg/fits2jpeg.py in the *fits2jpeg* package, as follows:

```
from fits2jpeg.fits2jpeg import fits2jpeg

fits_image_path = 'path/to/some/fits/files'
jpeg_image_path = 'path/to/output'

fits2jpeg(fits_image_path, jpeg_image_path)
```

A progress bar will be displayed and the converted files will be stored to `jpeg_image_path`.

Requirements: CFITSIO

The conversion code needs the cfitsio package to be installed. The installation instruction comes with the package (see `README` inside the downloaded source package). For convenience, the steps are summarized below.

For Linux OS (Tested on Ubuntu 22.04)

There are two packages which are needed for `fits2jpeg` code to work on Ubuntu: `libjpeg9-dev` and `libcfitsio-dev`. The first one is usually already installed in Ubuntu (check by running `dpkg -L libjpeg9-dev`). The second one will be installed manually following the steps below:

1. Download the latest version: https://heasarc.gsfc.nasa.gov/fitsio/
2. Extract and `cd` into the directory (e.g., cfitsio-4.2.0)
3. Following the instruction in `README`:
   a. run `./configure --prefix=/usr/local`
   b. run `make`
   c. run `make install` (optional)
4. Test the installation by following these steps:
   a. `make testprog`
   b. `./testprog > testprog.lis`
   c. `diff testprog.lis testprog.out`
   d. `cmp testprog.fit testprog.std` (no output means it passed the test and the generated output is identical to the expected output.)

For Windows OS

Installation of CFITSIO on Windows OS follow the instruction in `README.win`, which can be found in the cfitsio download page mentioned above.

For MAC OS

1. Installation of the `cfitsio` and **JPEG packages** is required prior to running the `makefile` for `fits2jpeg`. Using homebrew, I downloaded these packages for Mac OS. Here are the actions:
   a. Install homebrew
   b. Make sure xcode-select is installed: `xcode-select –install`
   c. install cfitsio: `brew install cfitsio`
   d. install libjpeg: `brew install libjpeg`
2. It is necessary to modify the **makefile** in the fits2jpeg package to specify the include directory for jpeg (`jpeglib.h`) and cfitsio (`fitsio.h`), and to specify the location of fitsio (`libcfitsio.a`) and jpeg (`libjpeg.a`) archive libraries. Further, define the path to binaries too.
3. Cfitsio and libjpeg packages are located in the homebrew directory for Mac OS users. From there, we may discover the locations of these packages: include directory and archive libraries directory.
4. After the above-mentioned modification of the `Makefile`, the resulting executable is `fits2jpeg`.
   Run the command: `make`

Why Conversion?

For annotation of images, we must provide a format that is compatible with most platforms, including V7. Any mainstream format such as JPEG, PNG, JP2, etc. is acceptable.

The Sunpy package seems to only give access to the FITS format of the GONG's observations. The JPEG format of all images is available via the FTP server. Accessing the URL paths to FITS and JPEG images is already implemented in https://bitbucket.org/gsudmlab/mleco-datacollection/src/master/data_collection/vso_search_result.py and https://bitbucket.org/gsudmlab/mleco-datacollection/src/master/data_collection/gong_sampler.py. That said, there are two key challenges in using those JPEG files:

- the existing JPEG images are watermarked with NSO and NISP logos, and some metadata. Therefore, they cannot be directly used for our next task, i.e, training DNN models for detection and segmentation of filaments.
- the annotations carried out on JPEG images may not perfectly align with the RoIs in FTS formats, because we do not know the exact process for the fits-to-jpeg conversion.

Therefore, it would be best to have in-house conversion of FITS files to JPEG.

Usage of `fits2jpeg` (through Original C Code)

```
fits2jpeg -fits input_file -jpeg output_file [options]
```

Options (described in the documentation of NVSS's code):

- nonLinear: This causes the mapping of FITS pixel values to jpeg grayscale colors to use a nonlinear (square root) function to emphasize the lower levels in the image. This generally increases the effective dynamic range of the resultant image. The default is linear mapping.
- max min: The maximum and minimum values to be displayed can be specified. Pixels with values greater than the value specified by max are given the same grayscale as the maximum specified and values less than that specified by min are given the same grayscale as the minimum. The specified values may be adjusted to fall within the range of values actually present. The default is to display the full range of values in the image.
- quality image_quality: This value specifies the desired quality of the image in the range 1 to 100. The higher the quality the larger the resultant file. The default is 100

Usage of `fits2jpeg` (Through Our Python Wrapper)

To run this executable C file, we created a python file called https://bitbucket.org/gsudmlab/mleco-datacollection/src/master/fits2jpeg/fits2jpeg.py and called it as follows:

```
fits2jpeg('./example.fits.fz', './example.jpeg, linear='linear',
quality=100)
```

Results of the experiment

We used the `fits2jpeg` method to convert a sample of 2000 FITS images into jpeg images and observe the code's processing time. Workflow includes:

1. Prepare 2000 compressed FITS images by running the gong_sampler jupyter notebook. This script upload FITS images to `PATH_TO_FITS_IMAGES` directory.
2. Read the 2000 compressed images from the fits image directory by simply executing the fits2jpeg.py code.

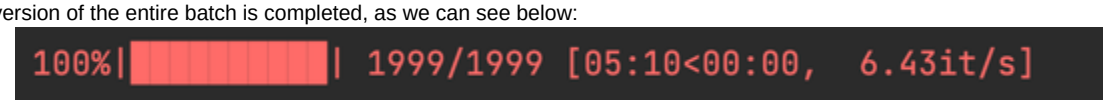
```
# fits_image_path: path where gong_sampler jupyter notebook is
storing
# compressed fits images

# jpeg_image_path: user define path where transformed jpeg images
stored
# linear: nonLinear
# quality: image quality, set to 100 (ranges between 0 to 100)
fits2jpeg(fits_image_path, jpeg_image_path, linear, quality)
```

3. For each image, fits2jpeg.py code decompress the image and store it in a temporary directory. We are using a temporary directory as we may not have write access to the original compressed fits image directory.
4. Convert the uncompressed FITS images to jpeg images and store them in the jpeg images path (user-defined) using fits2jpeg executable C code in the src directory.

Outcome is:

The above result means `1436 images out of 2000 (72%)` have been processed. `03:42 (3 minutes and 42 seconds)` represents elapsed time till 72% of conversion. `01:25 (1 minute and 25 seconds)` represents the remaining time according to the iterations per second value. `6.61it/s` represents iterations per second i.e. how many images are converted per second.

After the conversion of the entire batch is completed, as we can see below:



the process took a total of `5 minutes and 10 seconds` to convert a total of 2000 images with a conversion rate of `6.43 images per second`.

Juxtaposition of Conversions

Our converted images do not look as sharp as those available on GONG's archive. Below, we compare an instance of our conversion (right) with GONG's conversion (left).

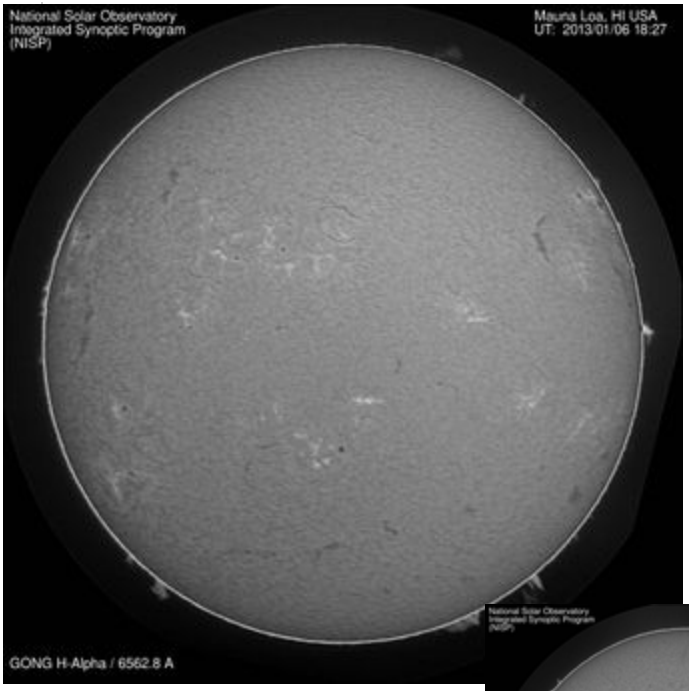
Why transformed image quality degraded?

The reason for this difference is explained by John Britanik (jbritanik@nso.edu) as follows:

> *The H-Alpha sharpening is based on a wavelet filter written in GRASP (Gong Reduction and Analysis Package), which in turn is an IRAF external module. It is difficult for us to be able to support the external building and running of this code since it requires some very specific and out-dated dependencies. I'm attaching the code (wavelet.x) in hopes that you can extract the filter kernal (5x5 kernal) and use that to reconstruct the filter in a more modern code environment. For the most part, the wavelet filter is just this convolution using this 5x5 kernal.*

Later, the adaption of wavelet.x code is sent by Niles Oien (noien@nso.edu) as follows:

> *Please find attached the file georgiaState.tar.gz. If you download that file, you should be able to do this in a terminal :*
>
> *gunzip georgiaState.tar.gz*
>
> *tar xvf georgiaState.tar*
>
> *cd georgiaState/*
>
> *You should then be able to run the script with the command :*
>
> *./runThis.sh*



That should build the C code for the wavelet filter. It will also build the C code for the fits2ppmColorscale /fits2ppmColorscale program, which will let you convert FITS data to PPM images. The steps in the runThis.sh file should be reasonably easy to follow.

All this relies on having the cfitsio library and utilities installed.

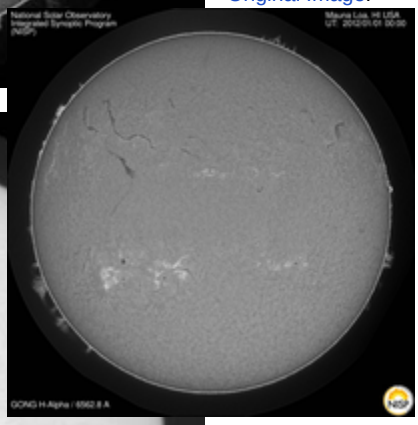
It's also handy to have the linux ImageMagick package installed with its "convert" utility, if that is installed then the runThis.sh will convert the PPM images to the more common JPG format and scale them down.

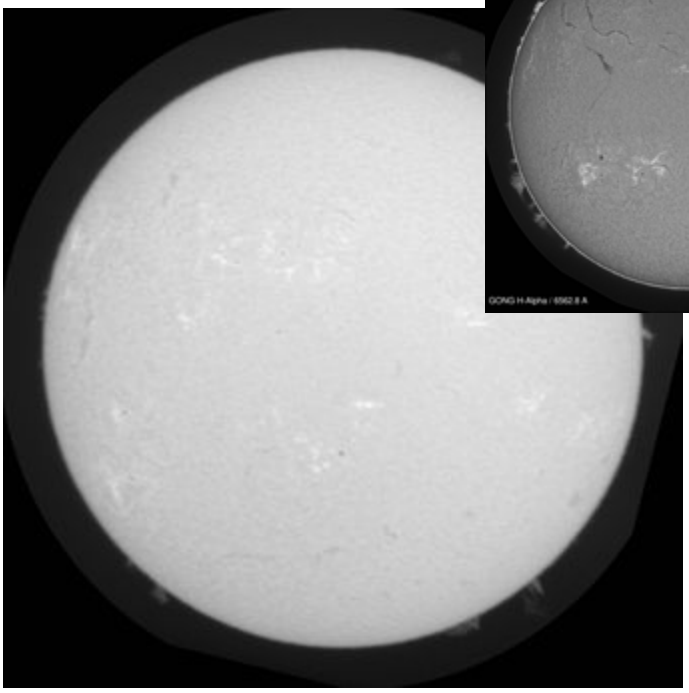
You probably want to work with the wavelet filtered FITS files rather than images, but the images give a nice visual of the FITS data.

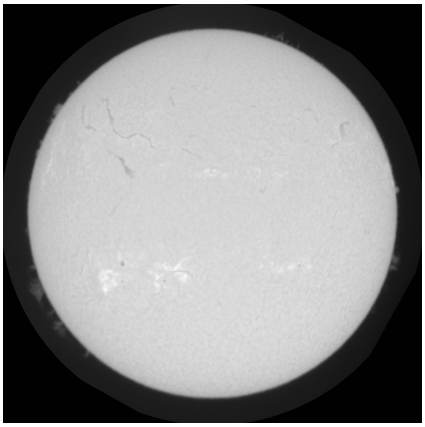
Appending some changes in the makefile and installing the ImageMagick package, we are able to run the C code for the wavelet filter

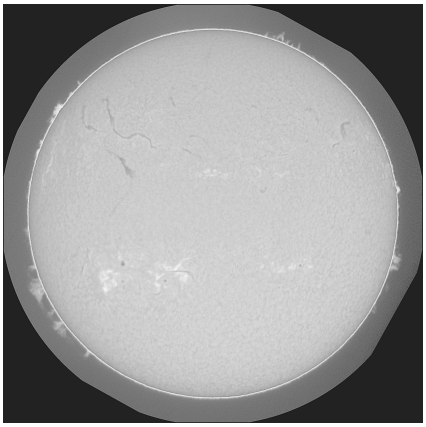
Below is a result in juxtaposition:

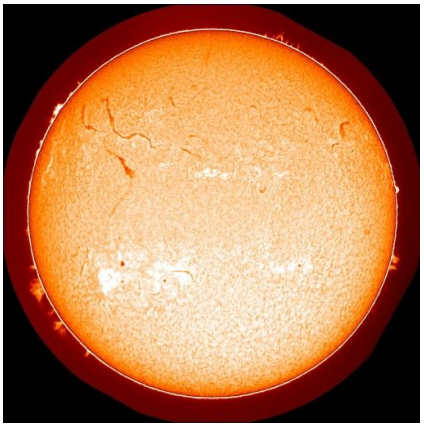
Original Image:











When the wavelet filter is applied to the input FITS image (a subpart of Niles code), our fits2jpeg repository, the results (middle) are better than when no filter is used (left). It is still not as clear as the image from the original GONG network, though. Interestingly, the output jpeg image of Niles code (right) is approximately as sharp as the original image though adding undesired coloring filters. Consequently, if we will do some tweaks with contrast, brightness, or color mappings in the wavelet filter code, we can achieve the exact replica of the original GONG network's image.

Implementation of NSO's C code in Our Project - Final Strategy

As I mentioned, by making a few adjustments to the wavelet filter code's color mapping, we can precisely duplicate the solar photos of the GONG network.

Let's examine the NSO code's workflow and the relevant modifications we made.

- `runThis.sh` is a wrapper shell script working on the top of `wavelet and fits2ppmcolorscale C executable code`.
- This code at first is converting compressed fits image (.fits.fz format) to decompressed fits image (.fits format) using funpack package.
- After getting decompressed fits image, shell script code is calling the `wavelet` C code to apply the wavelet filter on the top of the image. As described in the docstring of the wavelet code:

  > *Code decomposes a full resolution halpha image (.fits format) into 3 wavelet components (using b-spline basic wavelet). Recomposes the image with different weights to enhance small scale structures. Also adjusts image to enhance prominences.*

- After getting the wavelet fits image, the shell script code is calling the `fits2ppmColorscale` C code to apply grey scaling to the fits images, resulting in a ppm image.
  Note:
  - The original shell code applies colored mapping on the top of both fits_image (the image we get after decompressing the original image (.fits.fz)) and wavelet_fits_image (the image we get after passing through the wavelet filter).
  - Because our goal is to get the jpeg image with a wavelet filter, we haven't included the part of NSO's code where color mappings are applied on fits_image in order to save some processing time.
  - Again, the original shell code involves color mapping using `sdoaia304 with pixel range 0 3500`. However, we don't need color mapping as we need grey scaled output. Therefore we changed these settings to `grey with pixel range 640 5335` based on our discussion with the NSO team member Niles Oien (developer of the wavelet code).
- After getting wavelet_ppm_image, the shell script is converting it to wavelet_jpeg_image which is our desired output.
  Note:
  - The original shell code scaled down the resulted wavelet_jpeg_image dimensions by 25% i.e from 2048×2048 to 512×512. However, we don't require this scaling, so we haven't included that.

Code Execution

Installation of CFITSIO package:

funpack package is used in runThis.sh script to decompress the compressed FITS images. Further, Both the wavelet.cc and fits2ppmcolorscale.cc require `fitsio.h` package in order to complete the conversion process. For both, we have to make sure that cfitsio package is installed in our local system.

In order to install cfitsio, please refer to NASA's CFITSIO installation explanation. For MAC OS, simply execute `brew install cfitsio`. Please follow the provided explanation for cfitsio installation on linux and windows OS.

Once cfitsio package is installed, we need to make couple of changes in Makefiles.

Changes in Makefile:

We need to specify the path of the cfitsio package's `include` and `library` directories in wavelet and fits2ppmColorscale makefile in order to execute the shell script.

For mac, when cfitiso package is downloaded using homebrew, changes in both the makefiles generally are:

```
INC_DIR1 = /opt/homebrew/opt/cfitsio/include
INC_DIR2 = /opt/homebrew/opt/cfitsio/lib
```

Please provide the equivalent path of the include and lib directories under cfitsio package for linux and windows OS.

Once that is done, execute both makefiles using `make` command respectively. To do the testing, execute the fits2jpeg package from main_pipeline_sketch.ipynb jupyter notebook.

Results

We used the fits2jpeg_v2.py method to convert a sample of 2000 FITS images into jpeg images and observe the code's processing time. Workflow includes:

1. Prepare 2000 compressed FITS images from the gong_sampler script and upload those images to `PATH_TO_FITS_IMAGES` directory.
2. Read the 2000 compressed images from the fits image directory and transform them to JPEG using NSO's wavelet filter code.

3. For each image, execute the runThis.sh script in src_v2 directory, delete all the intermediate images (like wavelet_image_name.fits, wavelet_image_name.ppm) created during the transformation and store the output jpeg image to jpeg images path (user-defined)
4. In order to test the fits2jpeg_v2.py code, we just need to specify the FITS images path and JPEG images path.

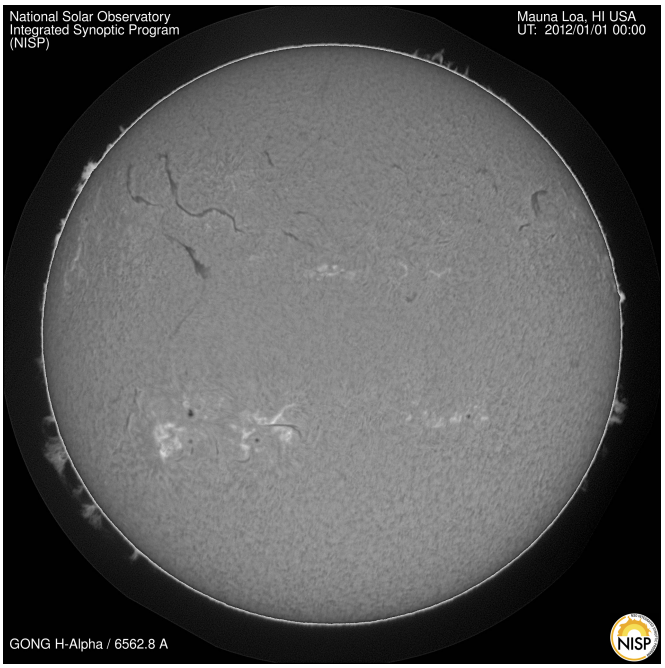
```
# Set the path of downloaded jpeg images, it is user defined path
jpeg_image_path = os.path.join(ROOT, "fits2jpeg",
"jpeg_images_path")
fits_image_path = PATH_TO_FITS_IMAGES

fits2jpeg(fits_image_path, jpeg_image_path)
```

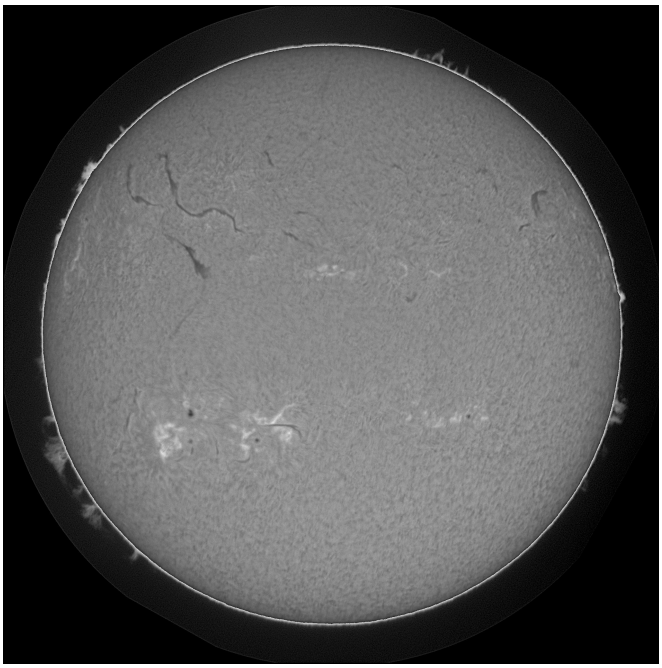
Outcome is:

```
72%|███████   | 1436/1999 [1:42:13<21:56,  2.34s/it]
```

The above result means `1436 images out of 2000 (72%)` have been processed. `01:42:13 (1 hour 42 minutes and 13 seconds)` represents elapsed time till 72% of conversion. `21:56 (21 minutes and 56 seconds)` represents the remaining time according to the iterations per second value. `2.34it/s` represents iterations per second i.e. how many images are converted per second.

Image Comparison





Both images are exactly the same, as can be seen. Consequently, we are able to accomplish our goal.

Image Quality-Processing Time tradeoff

When comparing the processing times of the two packages (NVSS's GitLab code and NSO's code), the former is nearly 6 times faster than the latter. However, due to the wavelet filter that was used, the latter's final image quality is far superior to the former. It is therefore wise to prioritize image quality at the moment above processing speed.

Summary

- In order to do annotation of H-alpha images of the sun FITS images are not compatible with platforms like V7. As a result, we need to convert it to some standard image formats like JPEG, and PNG in order to make the annotation process simpler.

- Initially, we understood the fits2jpeg conversion repository of NVSS code and make it work by wrapping it around our python code. Transformations of FITS images to JPEG were fast but the quality of the original image was not preserved. The solution to this issue was to apply the wavelet filter on the top of converted JPEG image.
- Consequently, we adapted NSO's wavelet filter code in order to preserve the quality of the original FITS image. Finally, we are now able to get the JPEG image of exact replica of the compressed FITS image.
- The next objective surely can be to optimize our python wrapper code and NSO wavelet filter code in order to make the transformation process much faster.