Stand-Alone Programs for Image Restoration

Geoff Daniell
gjd@lionhouse.plus.com

May 25, 2017

Introduction

The documents **restore1.pdf** and **restore2.pdf** describe two plug-ins for the `gimp` image processing package which attempt to correct for the deterioration of colour photographs with age. Although using these in the context of `gimp` is convenient, particularly if further corrections or other operations such as cropping are needed, `gimp` only acts as a wrapper for the image processing operations. This document describes the implementation of the same algorithms as **restore2** and **restore3** in stand-alone programs which are particularly convenient for batch processing large numbers of photographs. These avoid the necessity of installing `gimp` and getting plug-ins to work, but obviously preclude additional processing at the same time.

Since the critical part of the plug-in is written in python this is also the most appropriate scripting language for joining together the various image processing operations. It is not immediately obvious that the choice of software for these operations matters, however experiments show that the result with different image processing software is not always the same. It is worth listing possible reasons for this; the steps required in the calculation are

1. Conversion of the `.jpg` or other photographic file format to an uncompressed internal format which is more suitable for calculations. The question of compensation for an embedded colour profile could affect the outcome of the restoration.

2. Calculation of a small copy of the image to speed up subsequent processing. Different packages offer choices for the interpolation method, but these are unlikely to affect the results significantly.

3. Calculation of a colour map reducing the number of colours. This is a critical step and one most likely to affect the outcome of the restoration. A number of algorithms are in use; some do not give a fixed number of colours and in general they are poorly described; often the criterion for optimality is not explained. I have also invented an algorithm, which is implemented in python.

4. Estimation of the parameters for the restoration. This step is done in python and is independent of the image processing environment. It is a two stage process repeating some of the steps in this list.

5. The actual restoration. This involves a scaling and 'gamma' correction. The method of performing the necessary arithmetic varies between image processing packages but only very small differences in the output can be expected.

6. Estimation of the 'saturation' and its enhancement. There is more than one definition of 'saturation' but provided the correct one is used the fact that the details of this calculation differ between packages should not be important.

7. Conversion of the internal image format to `.jpg` or other photographic format. As with the initial conversion the question of embedded colour profiles needs to be noted.

We review the possible advantages and disadvantages of three image processing packages.

Python Image Processing Library

Since a python interpreter is essential for the central calculation it is clearly a good option to do the whole process in python. The **Python Image Processing Library (PIL)** ceased being maintained some years ago but more recently a new version: **Pillow (PIL Fork)** has been introduced and this seems to work well. The advantage of this approach is that it will work under any operating system including `windows`. The reading of images handles embedded colour profiles but these must be re-imposed when saving. Four re-sizing filters are provided; the choice should not matter much. For colour quantization three different algorithms are available but they are poorly documented. Method 0, the median cut, is well known and is the one I have used. One can easily get the palette as a python list but the ordering of the colours is the reverse of that used in `gimp`.

For the restoration step an image is easily split into red, green and blue components and applying a mathematical function identical to the `gimp levels` procedure is straigtforward. Internally this operates by constructing a look-up table with 256 entries and using this to transform each pixel. It is therefore fast although the function is defined using python code.

A conversion to **HSV** mode is built-in and one can get the necessary statistics for the saturation adjustment which is done by another mathematical function on the **S** channel, followed by conversion back to **RGB** mode.

The advantage of this implementation is that it is pure python and therefore portable. The disadvantage is that the quantization procedure is not known in detail. In practice the results are usually similar but not identical to those obtained with the `gimp` plug-in and differences can be attributed to

the different colour quantization algorithm. To bypass this unsatisfactory feature another version has been written which uses my own quantization procedure, written in python. This was developed because there was some discussion of removing the procedure from `gimp` and therefore making the plug-in fail to work. The advantage of this method is that it is a fixed piece of code and the results of restoration should be very close to those from the `gimp` plug-in using it. It can act as a reference with which other code for quantization can be compared. It does appear to perform well but the code included in the python image processing library is hopefully more fully tested than mine.

Netpbm

An extensive collection of programs for image processing is available under the name `netpbm` and includes code for most of the required operations. The programs can be called from python using the `subprocess` module. The only advantage over using the python imaging library is that the code is better documented. The format `.pnm` is used for the uncompressed image. There appears to be no action on embedded colour profiles. There are several options for producing the small working image but the choice should not matter. The quantization is provided by the `pnmcolormap` program and the algorithm used is documented and a technical reference provided. Unfortunately the program fails if the image contains too many colours when it performs an automatic change in the maximum value used to represent a pixel. This is disastrous for the subsequent use of the colour map in the restoration. I am not clear why it does this, possibly historically it was used to save memory. The problem of producing a set of N colours that 'best' represent an image is always soluble even if one can dispute the interpretation of 'best'. The only option when using the **Netpbm** package is to use a different quantization program, such as my own python one.

There are also problems in doing the restoration using `Netpbm`. The program `pamfunc` provides simple arithmetic but not power law changes. There is a program `pnmgamma` that does this but it confuses the use of 'gamma' by display hardware with simple the arithmetical calculations that we need. The only way to be certain about the result of the calculation is to do it in python. There is a further problem in that conversion to **HSV** mode is not supported so one is forced to do this in python. The program `ppmbrighten` can then be used to change the saturation.

In summary the `netpbm` suite of programs offers little advantage over the python imaging library and a lot of disadvantages.

ImageMagick

ImageMagick is another powerful suite of image processing programs. There is a full discussion of colour profiles when accessing image files. It offers a

large choice of re-sizing filters but as emphasised above which one is used is probably not important. The quantization algorithm is described in detail, but it may produce fewer colours than requested; this is a nuisance rather than a fundamental problem.

The main difficulty with the `ImageMagick` suite is that it is difficult to get a list of the colours except embedded in a long list of information about the image. Expression matching code would then be needed to extract the part required. The effectiveness of the list of colours produced by the algorithm in actual restoration has not been investigated.

There is a very comprehensive set of mathematical operations that can be applied to perform the restoration.

For the saturation adjustment **HSV** appears to be supported as a colorspace although it is then omitted from the list of descriptions.

In summary the ImageMagick operations are well documented and could be valuable in developing a better stand-alone program for image restoration but they do to offer any immediate advantages over the pure python described above.

Conclusions

We have investigated three sets of image processing software with a view to producing a stand-alone program for restoring faded photographs similar to the `gimp` plug-in described earlier. The final programs are pure python code and use the python imaging library **Pillow**.

Two versions exist: `pyrestore2.py` uses the built-in colour quantization code in the library while `pyrestore3.py` uses my own algorithm for this. The results are not identical; in general the built-in code produces slightly warmer colours which are more pleasing but occasionally it produces results which are *much* worse. In fact there are small differences between the results from `pyrestore3.py` and the corresponding `gimp` plug-in, which are not explained.

To run the programs type

```
python pyrestoreN.py dir=<directory> Light-Dark=<value> saturate=<[True|False]>
```

in a terminal. Choose `N` equal to `2` or `3` as required. `<directory>` is the directory containing the files to be processed, default = current directory, `<value>` is a parameter for adjusting the brightness of the restored image, the default is 1.0 and the value of the saturate parameter should be `True` or `False`, the default is `True` which increases the saturation of the restored image.

This processes all the files in `<directory>` and puts the restored ones in a subdirectory `restored`, which must exist. Parameters that take default values may be omitted.