Documentation of ALASKAv2 dataset scripts: A Starting point for Moving Towards Steganography and Steganalysis into the wild.

Rémi Cogranne*, Quentin Giboulot* and Patrick Bas†,

* Troyes University of Technology, Lab. for System Modelling and Dependability, ROSAS Dept., ICD, FRE 2019 CNRS, Troyes 10010 Cedex, CS 42060, France
† CNRS, École Centrale de Lille, University of Lille, CRIStAL Lab. , France.

Corresponding author: remi.cogranne@utt.fr

Very Brief "how to"

Before describing with more details the way the proposed scripts works, we provide in this preamble a very brief "how to" guide. Within the folder `Download_script` you can find the download script, `ALASKA_v2_RAWs_download_script.sh` which can be run under macOS/Linux/Unix as follows:

```
bash ./ALASKA_v2_RAWs_download_script.sh
```

The dataset will automatically be downloaded into the folder `Conversion_script/devHome/ALASKA_v2_RAWs` after which, those raws can be converted using the proposed randomized deveopement script using the following *python3* command:

```
python3 ./ALASKA_v2_conversion.py > ./conversion_output.txt 2> ./conversion_errors
```

which prevent flooding the terminal with many output that are dump to files *conversion_output.txt* and *conversion_errors*.

A brief overview of various folders path is represented in Figure 1 right below.

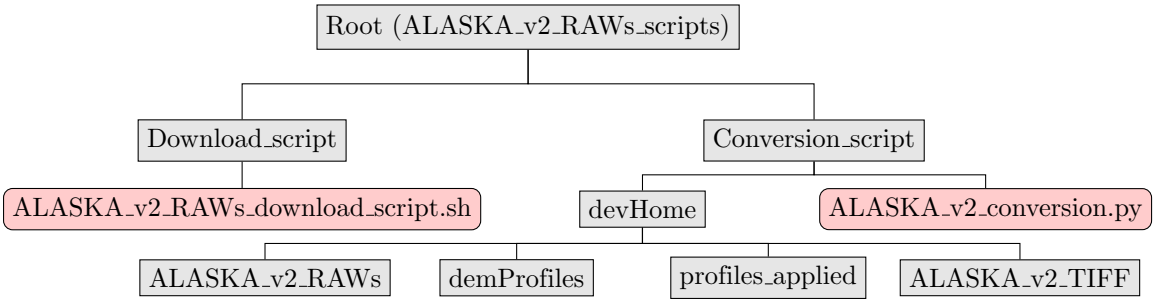

Figure 1. Representaion of path of the folders and scripts

1 Introduction

This documentation explains briefly how to use the scripts provided within ALASKAv2 dataset and available for download for any research or education purpose at alaska.utt.fr.

More precisely, all codes and dataset available on this website are provided under Creative Commons BY-NC-ND license which, in brief, states that you are free to use the dataset and codes provided that it is not for commercial purposes and that appropriate credits are given.[1].

The goal of the present document is to help anyone using the script for simple execution but also to provide general information such that anyone can modify and adapt those script to generate a dataset that will answer its needs.

First of all, we would like to recall that, for better use of the ALASKAv2 datasets we has tried to provide various datasets that will certainly answer most of the researchers needs. Those dataset are:

- of different sizes: fixed size of 512×512, fixed size of 256×256 and various image sizes ;
- of different compression: uncompressed (tiff); jpeg compression with quality factors 100, 95, 90, 85, 80, 75 and various quality factors. ;
- either color or grayscale.

Note that for your own research, you are free to download one of the aforementioned datasets of the set of $80,000$ files from which those images have been obtained. However, if one of the pre-prepared datasets fits your needs, it is highly recommended to use this dataset directly.

We also kindly ask you to credit our (enormous) work by either referring to the Alaska website URL (`https://alaska.utt.fr`) or, more relevant, by simply citing one of the associated papers [1], or the present document [2].

Eventually note that the dataset of raw images is about 1.65 TeraBytes ; the downloading time may be very long. Be sure that you use a very high-speed Internet access prior to the download.

2 How to download ALASKA dataset of RAW images (to be updated)

The first script to use is the download script `ALASKA_download_script.sh` in the directory `Download_script`. This is scripted uses the associated file `ALASKA_list_RAWs.txt`, which list the URL of all available RAW images, to download each and every image from the dataset.

First of all, be sure you are using the very last version of the scripts archive (download it right away before use) to ensure that you all the full list of RAW images.

The download script `ALASKA_download_script.sh` is written is *bash* programing language and, hence, work under Linux/UNIX OS (and possibly macOS). It only relies on *wget* tool, for downloading Internet resources, which is extremely common. In the very unlikely case it is not available under your system, install it using either *yum* or *apt-get* package managers.

Note also that this script does not require any argument, therefore, you can launch it simply, for instance as follows:

```
cd PATH_WHERE_SCRIPTS_ARE/Download_script
bash ./ALASKA_v2_RAWs_download_script.sh
```

We highly recommend the use of this script because it features the following advantages:

1. it allows stopping the download and restarting it later on without having to re-download all the files (hence saving bandwidth from our server side),
2. it does not overload your terminal with many outputs.

[1] The dataset and codes come without any warranties and we are liable for any consequence that may follow from the use of this provided material

More precisely, prior to the download of each and every raw image, we check if a file with the same name exists on your local system; if so the download of this file is skipped.

Besides, to preserve file integrity, we download all raw images in the current working directory prior to moving them in the directory specified into the script, hence ensuring that partial download of files are not considered.

Regarding to the output printed on the terminal, we used *wget* in a quiet mode and only print out the time required to download images every 10 raw images. Note that, however, a log file, which contains all the details about the download of each file, is created in the same directory (from which the download script is launched, and named `ALASKA_log_downloads`).

There is one extremely important variable to set in the script, this is `Image_local_path` which specified the directory where RAW images will be downloaded.

By default, the raw images will be downloaded in the directory `./Conversion_script/devHome/IMAGES/ALASKA_RAWs/` which is the one used later on, by default, by the conversion script `ALASKA_conversion.py`. This directory can be changed by simply modifying the variable `Image_local_path`.

There are way less important variables that can be changed such as: `ALASKA_list_path`, which specifies where the file that contains the list of ALASKAv2 raw images URL is located.

Eventually, note that you can use, with minor adjustments, the same script to download all the datasets we provide (color / grayscale, various size and compression factors). All you have to change is the directory of the files in the URL; for instance, the image raw images 00888.dng will be downloaded from the following URL: `alaska.utt.fr/DATASETS/ALASKAv2_RAWs/00888.dng`

To download the corresponding TIFF image of size 512×512 all you have to do is replace the directory `ALASKAv2_RAWs` with the name `ALASKAv2_TIFF_512_Color`.

The name of the existing folders (and associated datasets) are the following:

`ALASKAv2_TIFF_256_COLOR`, for dataset of size 256×256 uncompressed ;

`ALASKAv2_TIFF_512_COLOR`, for dataset of size 512×512 uncompressed ;

`ALASKA_v2_TIFF_VariousSize_COLOR`, for dataset of various sizes uncompressed ;

`ALASKA_v2_JPG_512_QF100_COLOR`, for dataset of size 512×512 compressed in JPEG with quality factor 100 ;

Note that with the last example you can replace `512` with either `256` (for size 256×256) or with "`VariousSize`" (for random sizes).

You can also replace `QF100` with `QF95`, `QF90`, `QF85`, `QF80`, `QF75` or `QFVarious` (this latest being for randomly select JPEG quality factors).

For instance: `alaska.utt.fr/DATASETS/ALASKA_v2_JPEG_256_QFVarious_Color/00888.jpg`, corresponds to an image with quality factor randomly selected and size 256×256 (which is especially relevant for Deep learning).

Similarly: `alaska.utt.fr/DATASETS/ALASKA_v2_JPEG_512_QF75_Color/00888.jpg`, corresponds to an image with quality factor 75 and size 512×512.

Eventually, you can replace, to any folder name the following suffix "`COLOR`" with "`GrayScale`" (beware: it is CAPS sensitive) to download a grayscale version of any of the aforementioned dataset.

If you have any trouble to download one dataset you

3 How to convert ALASKA RAWs into JPEG images

We have prepared a set of scripts that will convert the ALASKA dataset of RAW images into JPEG. First of all, as explained on the ALASKA website disclaimers, you are absolutely free to modify and redistribute the conversion script (provided that credit to ALASKA contest is explicitly given) such that

you can create a dataset that fits your need; note, however, that there is a huge work behind the design of those script and that a modification may result in poor results.

Note also that the **development of the whole** $80,000$ **raw images takes about one day** (using 30 cores out of 40 cores from a dual Xeon @ 2.20GHz).

The main script to be used is located in the `Conversion_script` directory and named `ALASKAv2_conversion.py`. Once again, this script does not require any argument, you can therefore simply use it, for instance, as follows:

```
python3 ./ALASKAv2_conversion.py
```

This script calls the two other supporting functions, namely `image_conversion_fun.py`, which contains all image processing function (mostly for resizing) and `random_dev.py` which is used to generate and output a random development pipeline from RAW to JPEG (including, random selection of demosaicing, sharpening, denoising, resizing, final image size and JPEG quality factor).

The role of those script is slightly more detailed below.

It is important to point out that this script should also work with python2 though this has not been tested extensively.

Please, note also that the conversion script is quite verbose, in order to let you know which image is processed and possible sources of conversion error. Therefore, to avoid having many message printed out in the terminal, it is recommended to launch the script as follows:

```
python3 ./ALASKA_v2_conversion.py > ./conversion_output.txt 2> ./conversion_errors
```

The conversion script `ALASKA_v2_conversion.py` contains in it very beginning a set of variables among which `keepUncompressed` is a switch that allows keeping (or not) the uncompressed tiff images and `baseName` is the prefix that will be used in folders name. By far, the most important variables are the following:

- first, `config_path` which defines the directories that will be used during the conversion (for saving processing parameters, temporary TIF images, etc.)
- the second, which is crucial if you wish to modify the development/processing pipeline is `config_process`: it specifies the probabilities of applying each and every possible processing/development tool (such as denoising, demosaicing algorithm, JPEG quality factor, etc.)

The definition of those variables is made in the preamble of the code (between lines 37 and 111).

In brief, after an initialization (lines 217 to 240) phase that consists in creating the non-existing directories and list of raw images, the conversion script calls the function `From_RAW_to_JPG` that works into six steps which are the following:

1. A random development process is drawn, using the probabilities from `config_process` variable and the script `random_dev.py` which specifies the distribution for each parameter. This is done in `ALASKA_v2_conversion.py` between lines 129 and 151.
   Note that to create a development process randomly, for each and every image, while ensuring reproducibility we have used, as a seed of the random process generator, the md5 hash sum of image file name (lines 133-134) ; as noted in comment, one replace this by a hash a timestamp to generate a random development.
2. Then, a raw image is subject to a first conversion using *rawtherapee* which essentially applies one of the demosaicing algorithms and some automatic color-adjustment. Is the present version, we used four demosaicing algorithms (in variable "`config_process["demosaicing"]`" and "`config_process["demosaicing_probabilities"]`" and, for each, a processing pipeline file (pp3 extension) is used. Those files are stored, by default, is the directory `Conversion_script/devHome/demProfiles/`. Note that from this step, we stored images with 16-bits depth, to avoid rounding effect at each step.

As a side note, one can notice that X3F raw image files (from Sigma X3 foveon sensor) seem to poses some problem to rawtherapee, therefore, we try using this software and, if it fails, we use an ad-hoc C software for dumping the raw data into a TIFF image. (lines 157-170)

3. The third step consists in resizing the image with a given factor (line 180); for this resizing the method is either crop / resize or / crop and then resize. The probability of those are given in variable `config_process`. The resizing factor (if used) is randomly selected in `image_conversion_fun.py` between the minimal value and 1.25 (upload by 25%).

4. The random development pipeline is applied using the randomly generated pipeline; the scripts in `random_dev.py` allow the writing of a pp3 file (line182) that can be used as an input to `rawtherapee` to carry out the processing pipeline specified in this file (line 187). Those file are written in folder `Conversion_script/devHome/profiles_applied/` (default) and the output image is a 8-bit TIFF which is the final image if no compression is used.

5. Eventually, the resulting image is compressed using the jpeg standard to get the final converted picture (line 191). The JPEG quality factor is randomly selected from a distribution that has been estimated from 2.75+ millions images downloaded from FlickR. This is defined in the variable `config_process["jpeg_qf_probabilities"]` ; for simplicity we picked JPEG QF between 60 and 100.

The conversion script `ALASKAv2_conversion.py` is written is pyhton and requires and calls the ras image editing software *rawtherapee-cli* (command line interface, we used version 5.7). Beyond this software the (main) requires libraries are the following:

- numpy (we used version 1.17.2);
- scipy (we used version 1.3.1);
- Python Imaging Library (PIL, not pillow, we used version 5.1),
- joblib (version 0.13.2) and multiprocessing, for counting cores but can also be used as an alternative.
- eventually, we used Python3 version 3.6.8

4 Sub-functions / Sub-scripts

The present conversion / development script use two subscripts, namely `image_conversion_fun.py` and `random_dev.py`. Though those are quite verbosely commented in order to be easy to understand, we provide below a brief description of the main functions they contain.

The former contains all image processing function which are briefly listed below:

- Functions "`center_crop`" , "`jpeg_compression`" and "`rgb2gray` and extremely simple; the first, "`center_crop`" downsize an image to a given size via centeral cropping and is used either when "smart-crop" raises an issue and when using re-sampling only to obtain the desired width or height. The second, function "`rgb2gray`, is used to speed up the computation when searching for areas with most of the content (the highest number of edges). In addition, in this context, it also pays off to avoid having different area for each color channel, hence the importance of converting into grayscale prior.
  The third, "`jpeg_compression`" is used for final JPEG compression only and is extremely straightforward.
  Now, the main image processing operations/functions are the following.

- Function "`image_randomizeResizing`" is called directly from the main script (at line 180) and constitutes the starting point of image resizing. Since there are three possible ways to resize an image, the function includes all those three possibilities. First case is the most complex one made of both resizing (by resampling) and cropping. In this case we start by generating a random resampling / resizing factor uniformly within the range $[\text{resize}_\text{F}\text{actorMin}; 1.25]$ where $\text{resize}_\text{F}\text{actorMin}$ stands for the smallest possible resizing factor in order not to end up with an image smaller than the desired final dimension. We then apply the "smart-crop" to get the final image dimension.
  Two important things must be clarified here; first the smart crop is inspired from [3] and is based on the selection of the area with the highest number of edges. Edges are detected using wavelet transform "approximation" while wavelets residuals are used to adjust the edge detection threshold w.r.t. noise level.
  Second, the "smart-crop" requires a variable, namely `GRIDSIZE` which specifies the shift in terms of the number of pixels applied when searching for the area with most edges. If the image size is too small (smaller than desired size plus twice the `GRIDSIZE`) we assume that selecting an area specifically is not relevant and that a center-crop will produce an image with always the same content.
  The part for cropping is made on this principle while the code for resizing is straightforward; however to preserve the image ratio we have written our own version of image resizing code (describe right below). We also adopted an approach used in the BOSS competition, that is to ensure that both image dimensions are at least of the desired size; if needed we crop one dimension to obtain exactly the desired dimensions.

- Function "`resize_keep_aspect`" has been written to ensure that the image ratio (height/width) is preserved during resizing (by resampling) operation. To this end, the code operates by first selecting the relevant resizing factor (the maximum of ratios computed for each dimension).

- Function "`edge_crop`" is also quite complex since it aims at selecting, during a resize by cropping operations, the area with the highest number of edges; it thus must proceed to the edge detection first. This operation is carried out via a simple wavelet filtering (into approximation and details) ; approximation is used to detect edges (by filtering to differentiate along horizontal et vertical directions) while the details are used to adjust the threshold according to noise level.

The latter is used to generate the random process through the two following subfunctions:

- the initialiser / constructor "`__init__`" defines, first, the distributions over which the (continuous) parameter for denoising and sharpening are drawn / sampled (lines 32-46).
  Then, within the same function, once distributions are set, all random development parameters are generated; drawn from those distributions (lines 49-63). Those parameters are (1) the demosaincing algorithm ; (2) the settings of sharpening ; (3) the settings of denoising ; (4) the settings of microcontrast ; (5) the jpeg quality factor ; (6) the final image size ; (7) the resizing kernel and (8) the amount of subsampling (when used in conjunction with cropping).

- The second subfunctions within this script is mostly used to output the randomly generated parameters into two file: first into a pp3 file that is used by *rawtherapee* into order to develop the image using the randomly generated parameters and, second, to dump all processing parameters for all images into the file specified by variable "`backup_file_path`" for logging purpose.
  One can note that in the second sub-function one parameter is chosen, the order in which denoising and unsharpening are carried (line 80).

5 Miscellaneous Minor Point

As of beginning of October 2019 we have fixed the main known issues that could be found in the previous version ; those were:

1. The Parallelization which, because of the use of lambda functions, was not straightforward. Now, you can easily use either Parallel (as in the example provided) or Pool to run the script in multiprocessing way to speed up the conversion process by up to 5 times.
   Note that *rawtherapee-cli* run, by default, in multi-processing mode, which limits the gain when running our script in parallel.
   Similarly, we also provide, as a comment, an alternative to use *python* module *Pool*: the main function "`From_RAW_to_JPG`" has been written to fits with the requirements of both parallelization modules.

2. We have stopped using the "L1-smart-crop" that was originally proposed in [4] ; indeed this was used to crop an image to a given smaller size by selecting an area whose noise residuals are similar to the one computer the over the whole image. However for images with large flat area (such as blue skies, over/under-exposed zones, etc. ...) this approach often fails. Therefore we have replaced this "smart-crop" with an "edge-crop" that uses a simple wavelet decomposition to detect edges [3] and select an area of fixed size with the highest number of edges with the underlying idea that it corresponds to the area with the most interesting content.

Eventually one can note that we have provided grayscale images datasets as well as a dataset entitled *ALASKA v2 BOSS Style* that corresponds to imags converted into pgm / ppm using the very script that has been used to convert RAW images from the BOSSbase dataset [].

While we provide an example code showing how we (very slightly) adapted BOSS conversion script it is worth noting that the color to grayscale "scripts" are not provided since it merely consists of a single command-line.

It is important to note that we used two different strategies for converting color images to grayscale. On the one hand, for spatial domain (uncompressed TIFF files) we used the *convert* command from *imagemagick* as follows (Linux / Unix / MacOS: bash):

```
for (( i=1 ; i< 80006 ; i++ ));
do
  j=$(printf "%05d" $i)
  convert -colorspace Gray ./ALASKA_v2_TIFF_256_COLOR/$j.ppm ./ALASKA_v2_TIFF_256_GrayScale/$j.pgm
done
```

In this example all the images from folder `ALASKA_v2_TIFF_256_COLOR` are converted into grascale using command `convert -colorspace Gray` and placed into folder `ALASKA_v2_TIFF_256_Grayscale` with a change of the extension from ppm to pgm.

On the other hand, to convert colored jpeg compressed images into grayscale we have used *jpegtran* command-line tool from *libjpeg* library that allows transformation and conversion of JPEG files directly into the DCT domain (without decompression / recompression). Conceptually, a color to grayscale conversion consists only in removing the chrominance channels and keeping the sole Luminance channel. This can be done, for instance, using the following command (Linux / Unix / MacOS: bash):

```
for (( i=1 ; i< 80006 ; i++ ))
do
  j=$(printf "%05d" $i)
  jpegtran -grayscale ./ALASKA_v2_JPG_512_QF95_COLOR/$j.jpg > ./ALASKA_v2_JPG_512_QF90_GrayScale/$j.jpg
done
```

6 Contact

We have tried to comment each and every script, such that it should be self-explained and easily modifiable. If, however, you have some troubles using of modifying those script, feel free to contact us at: alaska@utt.fr

Please, keep in mind that we have organized the ALASKA steganalysis challenge during our "spare time" and that we have worked hard to produce this follow-up dataset and this improved conversion scripts.

Eventually, let us recall that we provide those scripts without any warranty. Though we have done our best and may update it according to correction that may be applied, those scripts are provided without

any obligation to provide any accompanying service or maintenance. In no event, the organizers of the ALASKA challenge may be liable for any consequences arising out of the use of the provided scripts.

References

1. R. Cogranne, Q. Giboulot, and P. Bas. The alaska steganalysis challenge: A first step towards steganalysis. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'19, pages 125–137, New York, NY, USA, 2019. ACM.

2. R. Cogranne, Q. Giboulot, and P. Bas. Documentation of alaskav2 dataset scripts: A hint moving towards steganography and steganalysis into the wild. In *Arxiv Preprint*, 2019.

3. A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10):1737–1754, Oct 2008.

4. C. F. Tsang and J. Fridrich. Steganalyzing images of arbitrary size with cnns. volume 2018, pages 1–8. Society for Imaging Science and Technology, 2018.