Phenopix - Exposure extraction

G. Filippa

December 2, 2015

Based on images retrieved from stardot cameras, we defined a suite of functions that perform a simplified OCR procedure to extract Exposure values from images. Exposure will then be used to calculate camera NDVI following (cit.).

1 Install the phenopix package

The package `phenopix` is hosted in the r-forge repository and can be installed via the following command:

```
> install.packages("phenopix", repos="http://R-Forge.R-project.org")
```

Note that by running this command you will likely be asked to install the dependencies, which are available via the usual command:

```
> install.packages('package.name')
```

2 The steps

The suite of functions necessary to perform the OCR based Exposure recognition are:

- `getCoords()` to isolate the string with exposure which will be used by OCR
- `trainOCR()` to get a sample of numbers that work as training sample for the OCR procedure
- `getCoords()` to isolate the string with exposure which will be used by OCR
- `getExposure()` to apply ROI coordinates to an image of different size

Hidden behind each of the steps there is an image preprocessing which consists in converting each image into black and white reversed image (a binary

0-1 matrix) after transforming images in greyscale. Black and whites are reverted simply to have text strings written in black, with white background. This binary-conversion was formerly a separate function of the package (before v 2.0.2). Now the function is embedded in the other functions so that the user does not need to create a separate folder with all images converted to binary (a side effect is that the exposure extraction command is slightly slower now).
Before proceeding further, we load the package.

```
> library(phenopix)
```

3 Train OCR process

The first step, the most important one, is the training of the dataset. The important features that R must be able to recognize are 11: the ten numbers (0-9) and the capital letter E of Exposure. The function designed to perform this task is `trainOCR()`. Thier arguments are `image.path`, the path to your images, and `nsamples`, that decides the maximum number of images to be used in the training. `nsamples` is set to 100 but usually 11 to 15 images are enough to recognize all numbers. This is done by running:

```
> trainOCR(image.path='RGB/', nsamples=100)
```

In fig. 1 steps to train one number are illustrated. The function first prints a full image choosen randomly within the folder. `locator()` is automatically called. You must first left-click on topleft and bottomright corner of the crop you want to choose. In this example cropping points are rpresented by red dots. It is suggested to include the whole text string in the first crop. Then you will have to close your locator, based on which is your default graphic device (if device is X11, you will be able to close locator with any mouse button other than left, whereas on quarts device you have to press ESC key). After closing locator the plot in topright panel shows up, i.e. the crop you have identified before. Here, the plot title remembers you which numbers you still have to identify. In this example we choose to identify number 2. Crop again around your number (see red points for reference) and you will get a third crop, gridded in blue to highlight each pixel (bottomleft plot). You have to crop this last image as close as possible to topleft and bottomright margin of the choosen number. When you then close this last locator, the function `ask()` will ask you which number you have just identified, type 2 and press enter. A new image is then sampled from the folder and the procedure starts again from topleft panel if fig. 1. When

you will be done with all numbers and letter E, the process will be closed and the object you assigned to `trainOCR()` will be a named list described below.

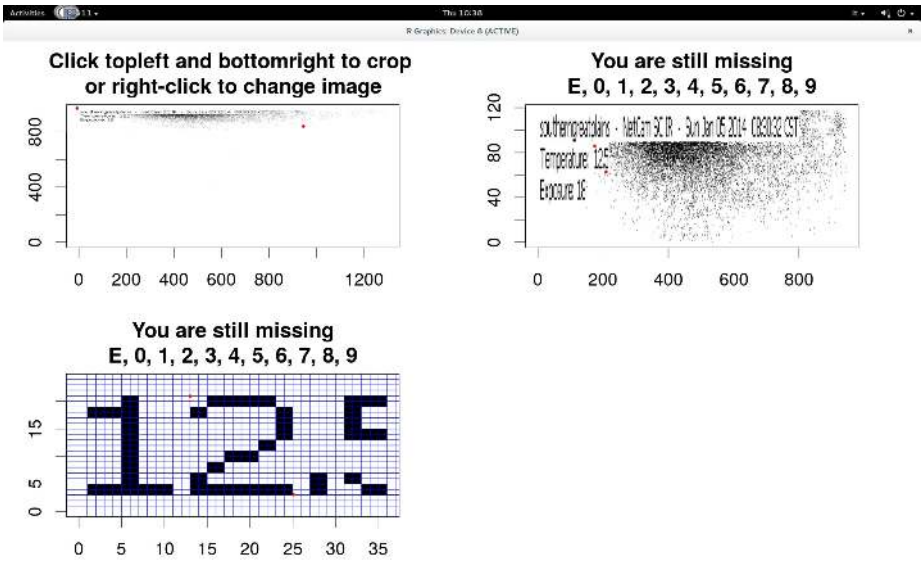

Figure 1: A screenshot of the procedure steps to train OCR function.

```
> load('south.numbers.RData')
> names(south.numbers)

 [1] "E" "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"

> str(south.numbers)

List of 11
 $ E: num [1:18, 1:12] 0 0 0 0 0 0 0 0 0 0 ...
 $ 0: num [1:18, 1:12] 1 1 0 0 0 0 0 0 0 0 ...
 $ 1: num [1:18, 1:10] 1 1 0 0 1 1 1 1 1 1 ...
 $ 2: num [1:18, 1:12] 1 1 0 0 1 1 1 1 1 1 ...
 $ 3: num [1:18, 1:12] 1 1 0 0 1 1 1 1 1 1 ...
 $ 4: num [1:18, 1:12] 1 1 1 1 1 1 1 1 0 0 ...
 $ 5: num [1:18, 1:12] 0 0 0 0 0 0 0 0 1 1 ...
 $ 6: num [1:18, 1:12] 1 1 1 1 0 0 0 0 0 0 ...
 $ 7: num [1:18, 1:12] 0 0 1 1 1 1 1 1 1 1 ...
 $ 8: num [1:18, 1:12] 1 1 0 0 0 0 0 0 1 1 ...
 $ 9: num [1:18, 1:12] 1 1 0 0 0 0 0 0 0 0 ...

> sapply(south.numbers, class) # a list of matrices
```

```
       E        0        1        2        3        4        5        6
"matrix" "matrix" "matrix" "matrix" "matrix" "matrix" "matrix" "matrix"
       7        8        9
"matrix" "matrix" "matrix"
```

```
> sapply(south.numbers, nrow) # check number of rows and columns
```

```
 E  0  1  2  3  4  5  6  7  8  9
18 18 18 18 18 18 18 18 18 18 18
```

```
> sapply(south.numbers, ncol) # check number of rows and columns
```

```
 E  0  1  2  3  4  5  6  7  8  9
12 12 10 12 12 12 12 12 12 12 12
```

Object is a named list with numbers and Exposure. It consists of 0-1 matrices. It is important to check that all matrices show the same row number, which means that number height is consistent, whereas it is normal that number 1 for instance has less columns than other numbers. Most stardot images will show exactly the number of columns and rows of this example, but it happens that with images of different size, the number of pixel of the shapes may change. Next figure shows how topleft and bottomright locator points must be set on each shape to get the best OCR recognition.

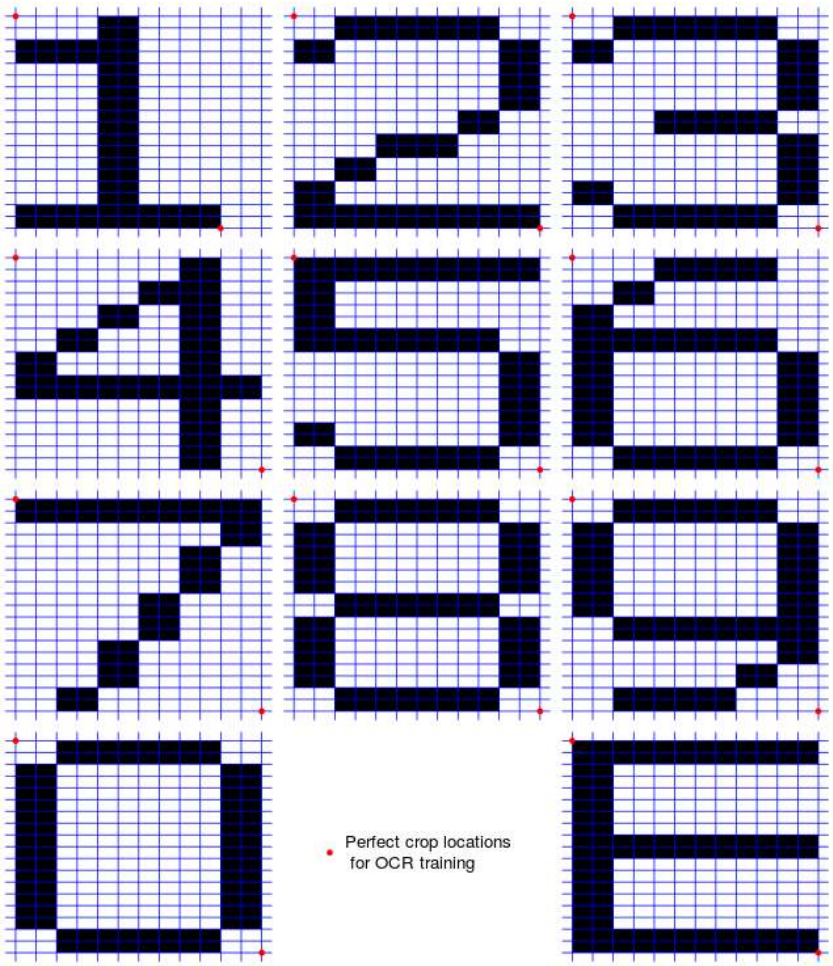

Figure 2: Examples of training shapes and the perfect crop locations (red dots) for each of them.

The named list obtained above (`south.numbers`) is now ready to enter the function described in the next section.

4 Extract coordinates of the Exposure string

This is another key point of the OCR procedure. The Exposure string may be located in different areas of the image. Usually, the text string is on the topleft corner of the images. Exposure can be on the first line as in Kamuela (fig. 3) or on the second line (fig. 4). It may be of apporximately 18 pixels height or

half sized (fig. 5).

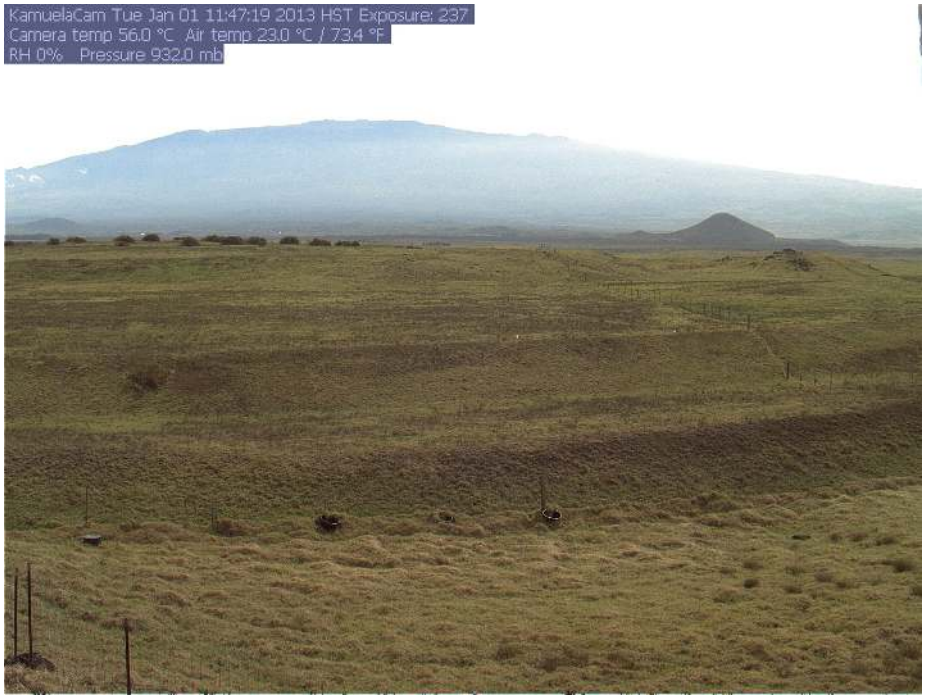

Figure 3: Example image from Kamuela site. Exposure is on the first line.

Figure 4: Example image from Niwot Ridge site. Exposure is on the second line.

Figure 5: Example image from Las Majadas site. Text string is smaller than in the previous examples.

Exposure can shift left and right according to the text size that preceeds exposure. The exposure number itself can be constituted by 1 to 5 digits and therefore it is important to define the coordinates of the cropping area with caution. Keep a reasonable amount of space either on the left and on the right side of the targeted text string to be sure to include it completely in all images. Up-downward shifting of the text string, instead, is very unlikely, so crop the image as close as possible to the text upper and lower margin. The `getCoords()` function is again based on `locator()` and works as follows.

```
> getCoords(image='RGB/southerngreatplainsIR_2014_01_04_123031.jpg')
```

It gets in input a complete path to a given target image that you will use to define the Exposure string position. Explore your images and get a rough idea of the range of exposures you might get. If your maximum exposure is a four digit number, then use one of those images to define your target. The figure below illustrates the steps to recognize coordinates. In the first step you only click the bottomright corner of the image you want to crop first. Crop to the right of exposure by some 100 pixels. Then your crop is plotted. Here click on topleft and bottomright margins. Be sure to include some 30-40 pixels on the

left of Exposure and 50-80 pixels on the right. If the Exposure string is aligned to the left of the image (as in fig. 4) you don't need extra space on the left. Titles in the various plots help you (fig. 6).

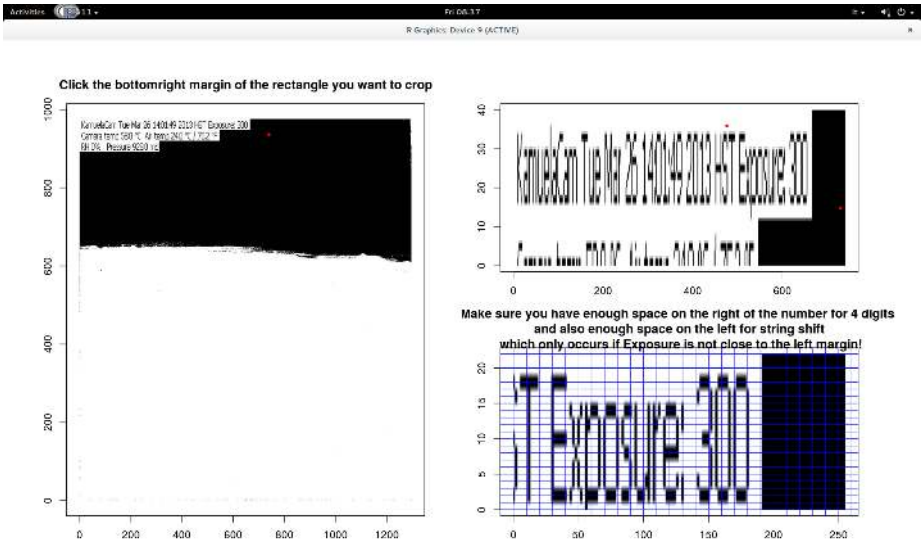

Figure 6: Steps of the function `getCoords()`. Red dots represent crop points used in `locator()`.

5 Extracting the exposure

Now that we have trained the OCR, and extracted target coordinates we are ready to extract exposure from our set of images. The function `getExposure` preforms this task. Arguments to the function are `ipath`, the path to your binary converted images, `coords`, the four coordinates in a named vector, `train.data`, the training dataset, `date.code`, the format of date included in each jpg file, which is used to extract a POSIX date for each exposure. An additional argument `sample` allows to run the function on a limited number of images for testing purposes. Now let's see if everything works. With the following code we run the procedure.

```
> ## manually set coordinates in this example
> south.coords <- c(x1=4, x2=172, y1=60, y2=81)
> load('south.numbers.RData')
> exposure.south <- getExposure('RGB/', south.coords,
+         south.numbers, 'yyyy_mm_dd_HHMMSS')
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
[1] 25
[1] 26
[1] 27
```

```
> class(exposure.south)
```

```
[1] "data.frame"
```

```
> summary(exposure.south)
```

```
                                          image       exposure
 southerngreatplainsIR_2014_01_03_120033.jpg: 1   Min.   :   2.0
 southerngreatplainsIR_2014_01_04_080031.jpg: 1   1st Qu.:   6.0
 southerngreatplainsIR_2014_01_04_083032.jpg: 1   Median :   8.0
 southerngreatplainsIR_2014_01_04_090031.jpg: 1   Mean   : 138.4
 southerngreatplainsIR_2014_01_04_093031.jpg: 1   3rd Qu.:  20.5
```

```
 southerngreatplainsIR_2014_01_04_100032.jpg: 1   Max.   :2400.0
 (Other)                                     :21
   timestamp
 Min.   :2014-01-03 12:00:00
 1st Qu.:2014-01-04 10:45:00
 Median :2014-01-04 14:00:00
 Mean   :2014-01-04 15:48:53
 3rd Qu.:2014-01-04 17:15:00
 Max.   :2014-01-05 11:00:00
```

The procedure ended up successfully. The printed index indicates that all 27 images sampled in this example were processed, and by visual checking we were 100% successfull in recognizing exposures. Now we will explore how the procedure is sensitive to changes in cropping coordinates. In the following example we expand the lower margin of the image by 8 pixels. In this way we will likely include a portion of the image below the text string. This portion will be, in many cases, a black band in the b/w image, and thereby a source of problem. We envelope the `getExposure()` command within a try statement in case it could fail.

```
> south.coords2 <- c(x1=4, x2=172, y1=60, y2=89)
> exposure.south2 <- try(getExposure('RGB/', south.coords2,
+         south.numbers, 'yyyy_mm_dd_HHMMSS'))

[1] 1

> class(exposure.south2)

[1] "try-error"
```

As expected, the procedure stops. Much importance must be given to upper and lower limits of the cropped image. It is useless to include extra space above or below the Exposure string. Let's see what happens if we include too much space on the right side of the image.

```
> south.coords3 <- c(x1=4, x2=180, y1=60, y2=81)
> exposure.south3 <- try(getExposure('RGB/', south.coords3,
+         south.numbers, 'yyyy_mm_dd_HHMMSS'))

[1] 1
[1] 2
[1] 3
```

```
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
[1] 25
[1] 26
[1] 27
```

```
> class(exposure.south3)
```

```
[1] "data.frame"
```

```
> table(exposure.south$exposure==exposure.south3$exposure)
```

```
TRUE
  27
```

Same results as in the first try. This indicates that we can move right as much as we need without affecting the computation. This is important because exposure can have up to 4 or 5 digits and we want to be sure to include them all. Be careful that this is true only if no other text is printed on the right of the exposure string. So if you have to set up the text string of your stardot (or camera in general) while programming it, make sure that Exposure be not followed by any text in the same line or, best, to keep Exposure in a separate,

single line. In summary, two rules are important: (1) Keep upper (lower) crops only few pixels above (below) the Exposure string; (2) Keep enough room on the right and left of Exposure to be sure to include all digits in the string.

Finally, when a set of images is processed we will likely do not want to check them all. We will therefore write a simple function that allows to check a sample of images in a fast way. Basically, a random sample of images is choosen and for each of them a plot is generated, which puts together the number you extract with `getExposure()` and the original string on which the extaction was performed. This function is not included in the package, so if you want to use it copy and paste it from this document.

```
> checkExposure <- function(data, ipath, coords, nsamples, opath) {
+         .plotImage <- function(image, ...) {
+                 ncols <- ncol(image)
+                 nrows <- nrow(image)
+                 suppressWarnings(plot(0,
+                         type='n', xlim=c(0, ncols),
+                         ylim=c(0, nrows), ...))
+                 suppressWarnings(rasterImage(image,
+                         xleft=0, ybottom=0, xright=ncols,
+                         ytop=nrows, ...))
+         }
+         to.sample <- length(list.files(ipath, full.names=TRUE))
+         the.sample <- sample(to.sample, nsamples)
+         all.jpeg.files.full <- list.files(ipath,
+                 full.names=TRUE)[the.sample]
+         all.jpeg.files <- list.files(ipath)[the.sample]
+         data.subset <- data[the.sample,]
+         for (a in 1:length(the.sample)) {
+                 jpegname <- paste0(opath, all.jpeg.files[a],'.png')
+                 .binaryConvert <- function(img) {
+                         grey.image <- 0.2126*img[,,1] +
+                         0.7152*img[,,2] + 0.0722*img[,,3]
+                         binary <- round(grey.image, 0)
+                         rev.binary <- ifelse(binary==1, 0, 1)
+                         return(rev.binary)
+                 }
+                 image.target <- readJPEG(all.jpeg.files.full[a])
+                 image.target <- .binaryConvert(image.target)
```

```
+                 cut.image <- image.target[coords['y1']:coords['y2'],
+                 coords['x1']:coords['x2']]
+                 cut.image.binary <- round(cut.image)
+                 png(jpegname, width=500, height=500)
+                 par(mfrow=c(2,1))
+                 .plotImage(cut.image.binary)
+                 plot(0, type='n')
+                 act.value <- data.subset[a,'exposure']
+                 text(1,0, act.value, cex=5)
+                 dev.off()
+         }
+ }
```

Arguments are `data`, the dataframe of exposures you have extracted (`exposure.south` in our example); `ipath`, the path of binary images, `coords`, the cropping coordinates; `nsamples` is the number of samples you want to check, `opath`, an output folder where to save the generated plots.

6 Summary

A procedure is illustrated, that allows to automatically extract exposure values from a set of images that display this number as a printed text string. Procedure was built to work with stardot images, and not tested on other type of images. However it should be flexible enough to adapt to other image size, text fonts, etc. The suite of R functions presented here are part of the `phenopix` package downloadable from the R forge (r-forge.r-project.org/projects/phenopix/). I am available for further information, debug, receive and provide suggestions at `gian.filippa@gmail.com`