KID
Handling Scanned Files

Fabian Blasch

09.03.2022

Tesseract

Tesseract is an optical character recognition engine, it can be used to convert pictures into machine readable characters. The sourcecode is available on GitHub and fortunately the API was implemented in an R package.

First Steps

Tesseract offers the possibility to alter the engine based on the problem at hand. In the case of KIDs, we want to extract the position of the scale of each SRRI entry. Thus one parameter of great importance is the character white list. It allows us to narrow the classification into the characters or in this case digits handed over to tesseract via the white list.

To eliminate noise we first look at a snippet taken from one of Security's KIDs that only contains the scale and no text, this allows for the identification of problems related to the extraction of the scale without the surrounding noise.

```
# load KiDs
setwd("./../KIDs")
devtools::load_all()
```

```
## i Loading KIDs
```

```
# example pdf
setwd("./../../../KIDs/Security")
pd <- list.files(pattern = ".pdf")[1]

# first convert to png so we can use tesseract directly
pdftools::pdf_convert(pd, pages = 1, filenames = c("testscanread.png"), dpi = 600)
```

```
## Converting page 1 to testscanread.png... done!
```

```
## [1] "testscanread.png"
```

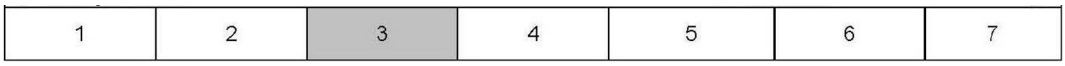
```
# set whitelist to digits 1-7 and whitespace
eng_spec <- tesseract::tesseract(options = list(tessedit_char_whitelist = " 1234567"))

# image that only contains the scale and no text
tesseract::ocr_data("Scale_notext.jpeg", engine = eng_spec) |> knitr::kable()
```

word	confidence	bbox
1	0.000000	97,5,165,84
2	88.425575	342,5,402,85
4	7.978073	816,5,869,84
5	60.284004	1061,5,1121,84
6	77.367195	1313,5,1366,85
7	0.000000	1528,5,1574,85

The readout worked good for all digits on white background, next we may use other engine parameters as well as image pre-processing to obtain the scale in its entirety. First off, we will try to convert to black and white to correctly identify the entire scale. The R package magick offers powerful tools for image processing.

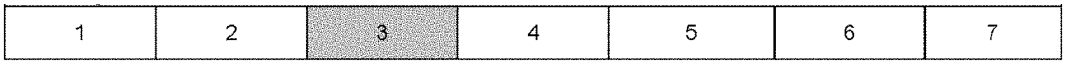
```
# reset
setwd("./../../../KIDs/Security")

# image
img_m <- magick::image_read("Scale_notext.jpeg")
```

```
# open image for a closer look
# magick::image_browse(img_m)

# plot image
par(mar = c(1, 4, 1, 2))
plot(img_m)
```



Now to switch from grayscale to a true black and white there are two different options. We can either use the function image_convert() or image_threshold().

```
# convert
img_m_conv <- magick::image_convert(img_m, type = "Bilevel")

# display
# plot image
par(mar = c(1, 4, 1, 2))
plot(img_m_conv)
```



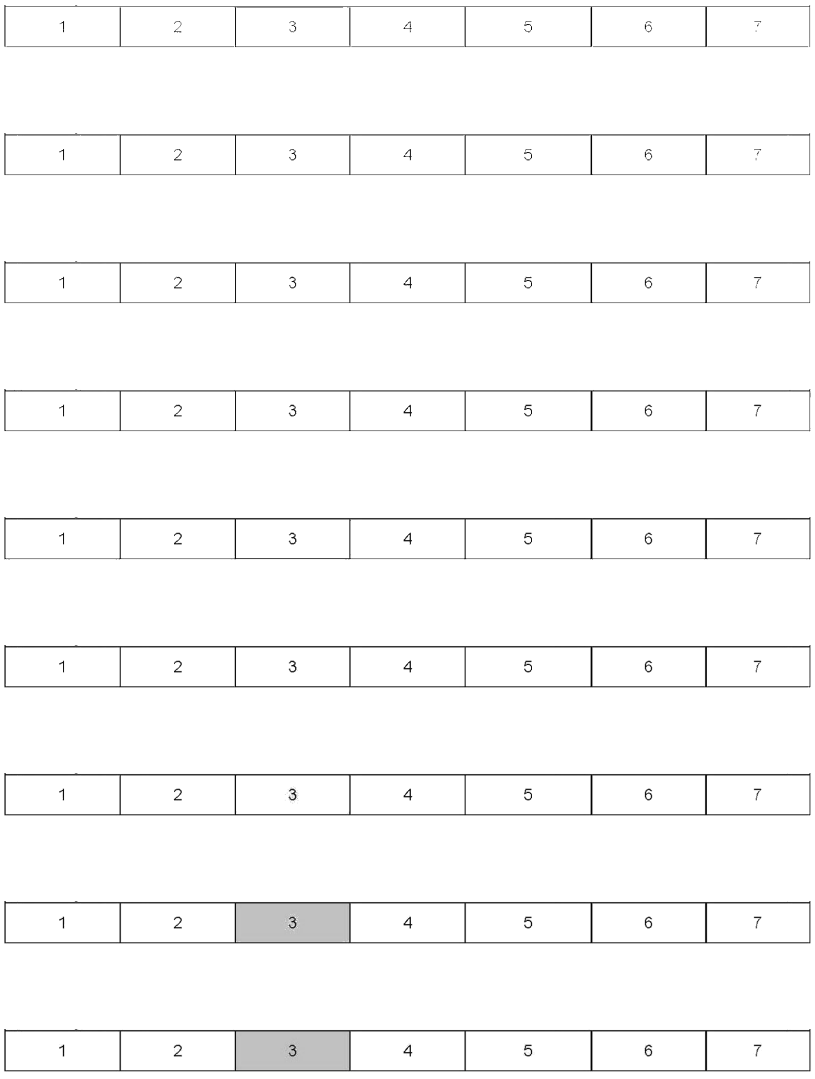

```
# repeat ocr
tesseract::ocr_data(img_m_conv, engine = eng_spec)|> knitr::kable()
```

word	confidence	bbox
1	44.12727	106,5,159,85
2	88.78406	338,5,406,85
3	37.03163	578,5,631,85
4	53.32304	818,5,871,85
5	73.43266	1058,5,1119,85
6	82.94523	1313,5,1366,85
7	0.00000	1523,5,1569,85

As visible from the table we are now able to detect the entire scale, on top of that the classification confidence increased significantly. Lets see if we can increase the confidence by using different thresholds when converting

to black and white.

```
# alter with varying thresholds
img_m_thresh <- lapply(paste0(seq(10, 90, 10), "%"),
                       \(x) magick::image_threshold(img_m, threshold = x, type = "white"))
# align
par(mfrow = c(9, 1), mar = c(1, 4, 1, 2))

# plot
lapply(img_m_thresh, \(x) plot(x)) |> invisible()
```



For the plots above the threshold starts art 10% and increases to 90% in increments of 10%.In theory we want to choose a threshold that keeps the digits as legible as possible while converting the shade to white. By observing the plots we know that this threshold has to lie somewhere between 70% and 80%. We have to keep in mind however, that this will vary across different KIDs, thus this parameter will have to be determined via cross validation at a later stage, should it perform better than bi-level processing.

```
lapply(img_m_thresh,
         \(x) tesseract::ocr_data(x, engine = eng_spec))
```

```
## [[1]]
## [1] word       confidence bbox
## <0 Zeilen> (oder row.names mit Länge 0)
##
## [[2]]
##   word confidence           bbox
## 1    1   37.47584    98,4,159,85
## 2    2   90.69081   338,5,399,85
## 3    3   66.06633   571,4,624,85
## 4    4   71.54774   811,4,871,85
## 5    5   67.67908 1058,4,1119,85
## 6    6    0.00000 1306,4,1366,85
##
## [[3]]
##   word confidence           bbox
## 1    1    0.00000    90,5,157,85
## 2    2   88.71307   334,5,402,85
## 3    3   63.00784   579,5,632,85
## 4    4   86.19589   816,5,877,85
## 5    6   67.12558 1061,5,1121,85
## 6    6   90.62632 1313,5,1366,85
## 7    7    0.00000 1521,5,1574,85
##
## [[4]]
##   word confidence           bbox
## 1    1   9.509064    97,5,165,85
## 2    2  77.638237   342,5,402,85
## 3    3  74.381256   579,5,632,85
## 4    4  71.474670   816,5,877,85
## 5    6  47.728615 1061,5,1121,85
## 6    6  82.772720 1313,5,1366,85
## 7    7   0.000000 1484,5,1574,85
##
## [[5]]
##   word confidence         bbox
## 1    1   11.63204  97,5,165,85
## 2    2   59.56064 342,5,402,85
## 3    3    0.00000 572,5,639,84
##
## [[6]]
##   word confidence           bbox
## 1    1    0.00000    97,5,157,84
## 2    2   90.05758   334,5,402,85
## 3    3   82.89348   572,5,632,84
## 4    4   83.86511   816,5,877,84
## 5    6   72.44370 1061,5,1121,84
## 6    6   90.63850 1313,5,1366,85
## 7    7    0.00000 1521,5,1581,85
##
## [[7]]
##   word confidence           bbox
```

```
## 1    1   36.78156    98,5,166,84
## 2    2   57.74221   328,5,403,85
## 3    3   22.47987   565,5,633,84
## 4    4    0.00000   810,5,870,84
## 5    6   13.63799 1307,5,1367,84
##
## [[8]]
##   word confidence           bbox
## 1    1    0.00000    90,5,157,84
## 2    2   90.30231   334,5,402,85
## 3    3   80.03242   572,5,632,84
## 4    4   54.35625   816,5,869,84
## 5   56   63.68980 1061,1,1114,90
## 6    6   75.67296 1313,5,1366,85
## 7    7    0.00000 1521,5,1581,85
##
## [[9]]
##   word confidence           bbox
## 1    1   39.07653    97,5,157,84
## 2    2   91.19086   334,5,402,85
## 3    3   58.06713   579,5,632,84
## 4    4   56.51816   816,5,869,84
## 5   56   59.64135 1061,1,1114,90
## 6    6   81.46667 1313,5,1366,85
## 7    7    0.00000 1521,5,1581,85
```