Optical Character Recognition (OCR) with ABBYY FineReader & Kraken

Institute for Computational Linguistics, 2020

Abstract: This paper briefly summarizes the applied OCR methods as well as the (optional) file splitting in the digitalization process of 10'012 file cards, given as pages in one large pdf file (778.8 MB).

1 Preprocessing

Breaking the input into multiple pieces is not necessarily needed, however, «divide et impera» appears to be a recommendable strategy: A preceding splitting bewares from reprocessing everything in case of an error, and might, facilitating parallelization, speed up the process significantly.

```
1 from PyPDF2 import PdfFileWriter, PdfFileReader
2 input_pdf = PdfFileReader(open("/path/to/input/file.pdf", "rb"))
3 for i in range(input_pdf.numPages):
4     with open("file_"+(5-len(str(i+1)))*'0'+str(i+1)+".pdf", "wb") as output:
5         PdfFileWriter().addPage(input_pdf.getPage(i)).write(output)
```

Code Snipped: A Python-script to save each page of a pdf file as a new pdf file.

The names of the output files should be of uniform length (depending on the particular naming system). Otherwise, *abc*-sort of certain file explorers might destroy the order, which eventually has disadvantageous effects on further processing steps (e.g. required id-extraction from file names).

2 ABBYY FineReader

This section documents the process of applying OCR on multiple files by the commercially proprietary software «FineReader» by ABBYY, which involves the creation of different directories as well as some configuration steps.

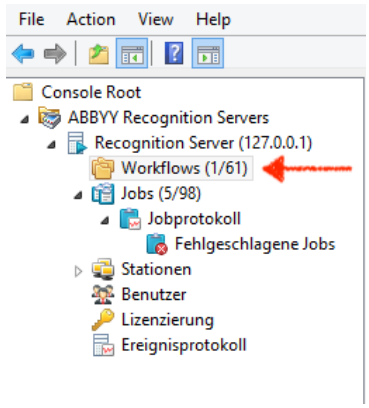
2.1 Directories (Input/Output)

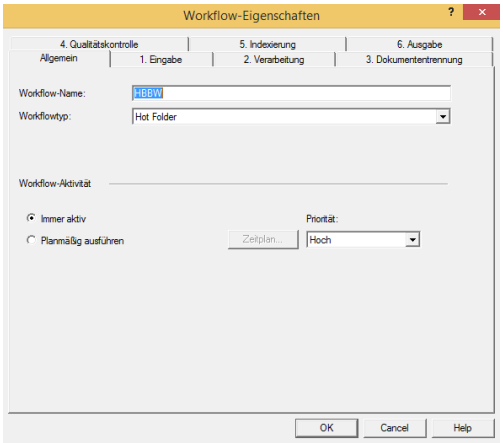
FineReader requires up to 5 directories (R/W rights required):

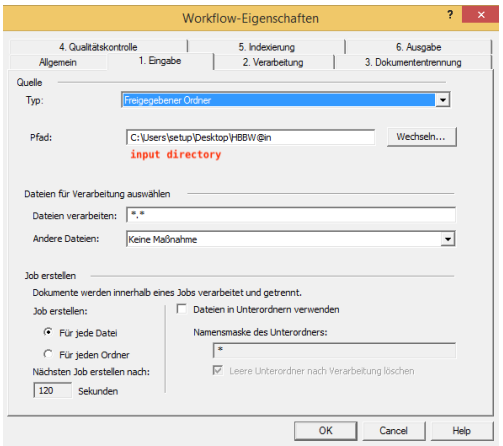
- Input («Hot Folder»)
- Output (the OCR output in its desired format)
- Errors (corrupt input files and corresponding log files)
- Log files (optional)
- Images (optional; the input after passing the «Hot Folder»)

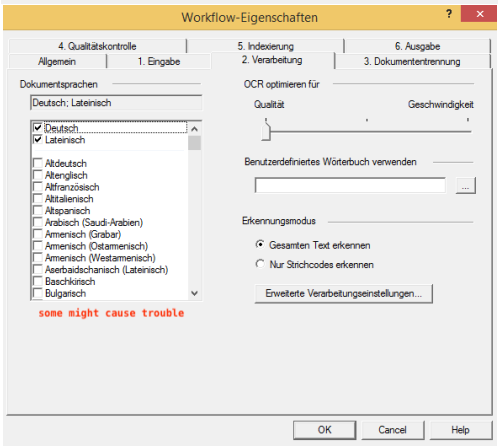
2.2 Configuration

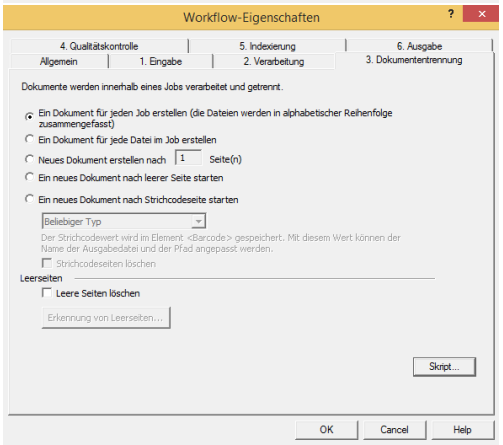
The entire process can be defined in a single «Workflow»:

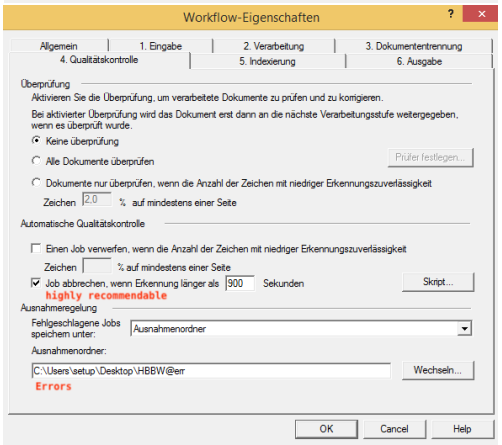


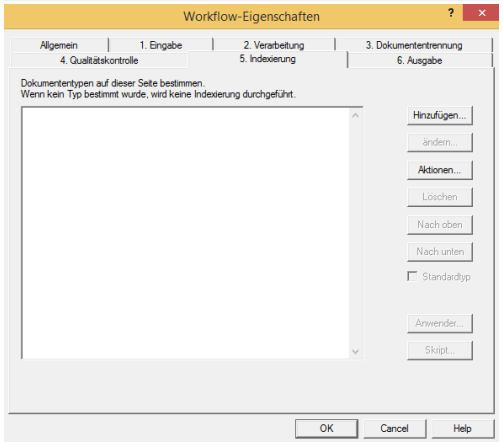


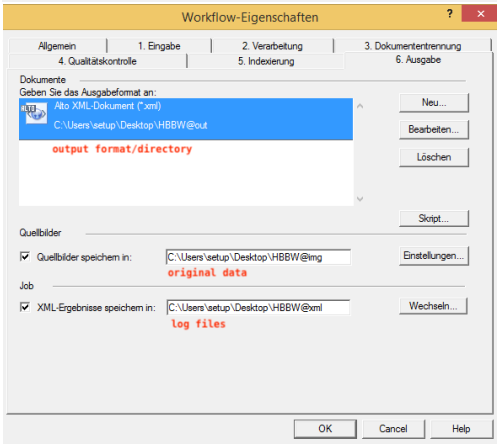












2.3 Execution

After completing the configuration above (by clicking on «ok»), the input directory becomes active. Every file in this directory appears to magically disappear, and (eventually) reappear again in the image directory - together with data in the other directories, as indicated by their names. If there are no configuration problems, the output follows immediately. Otherwise, the overviews «Jobs», «Job Protokoll» and «Fehlgeschlagene Jobs» (see Workflow) might reveal some information about the problem.

With the settings described, FineReader needed 3h and 49m to process our 10'012 files, thats is, 1.37s per file (80 KB) on average.

3 Kraken

The OCR-Tool «Kraken» needs to be downloaded and installed as described on the official website or on GitHub. Since Kraken runs on images, our pdf files had to be transformed:

```python
from pdf2image import convert_from_path
for page in convert_from_path("path/to/input_file.pdf", 600):  # dpi
    page.save("path/to/output_file.png", 'PNG')
```

Code Snipped (Python): pdf → png

```python
import subprocess, psutil
def run_kraken(input_path_png, output_path):
    cmd, url_model = "binarize segment ocr —a —m", "path/to/language_best.mlmodel"
    cmd = ' '.join(["kraken —i", input_path_png, output_path, cmd, url_model])
    sub_process = subprocess.Popen([cmd], shell=True)
    p = psutil.Process(sub_process.pid)
    try: p.wait(timeout=100)
    except psutil.TimeoutExpired: p.kill()
```

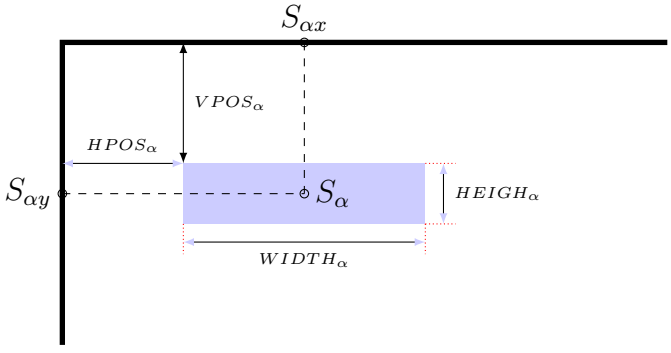
Code Snipped (Python): png → xml (alto)

4 Postprocessing

4.1 Scaling

The size of a page (length times width [px]) differs from file to file. Consequently, a reliable detection of contents based on absolute positions requires to normalize all coordinates.

- width $B_i \propto S_{i\alpha x}(\text{WIDTH}_{i\alpha x}, \text{HPOS}_{i\alpha x})$
- height $H_i \propto S_{i\alpha y}(\text{HEIGHT}_{i\alpha y}, \text{VPOS}_{i\alpha y})$



$$\text{Scaling factors:} \quad x_s(B_i) = \frac{\hat{B}}{B_i}, \quad y_s(H_i) = \frac{\hat{H}}{H_i} \quad (\hat{B} := \text{norm width}, \hat{H} := \text{norm height}).$$

4.2 Parsing

```
import xml.sax
import pandas as pd
from xml.sax.handler import ContentHandler

class AltoParser(ContentHandler):

    """ [String, HPOS, VPOS, HEIGHT, WIDTH] ---> [str, x, y, baseline] """

    # filter
    NAMES = ["Datum", "Absender", "Empfaenger", "Autograph", "Kopie", "Photokopie",
             "Standort", "Bull.", "Corr.", "Sign.", "Abschrift", "Umfang", "Sprache",
             "Literatur", "Gedruckt", "Bemerkungen"]s

    def __init__(self, scaling, inverse=False):
        super(AltoParser, self).__init__()
        self.data = pd.DataFrame({'Value': [], 'x': [], 'y': [], "Baseline": []})
        self.current_baseline = 0
        self.scaling = scaling
        self.inverse = inverse

    def startElement(self, name, attributes):
        if name == "TextLine":
            for a in attributes.getNames():
                if a == "BASELINE":
                    self.current_baseline = attributes.getValue(a)

        if name == "String" and "STYLE" not in attributes.getNames():
            d = dict()
            d["BASELINE"] = self.current_baseline
            for a in attributes.getNames():
                d[a] = attributes.getValue(a)
            if self.inverse:
                if d["CONTENT"] not in AltoParser.NAMES:
                    self.append(d)
            else:
                if d["CONTENT"] in AltoParser.NAMES:
                    self.append(d)

    def append(self, d):
        hpos, vpos = int(d["HPOS"]), int(d["VPOS"])
        width, height = int(d["WIDTH"]), int(d["HEIGHT"])
        x, y = AltoParser.get_mass_point(hpos, vpos, width, height, self.scaling)
        value = ''
        if ("SUBS_TYPE" in d) and ("SUBS_CONTENT" in d):
```

```
            if d["SUBS_TYPE"] == "HypPart1":
                value = d["SUBS_CONTENT"]
        else: value = d["CONTENT"]
        df = pd.DataFrame({
            'Value': [value],
            'x': [x], 'y': [y],
            'Baseline': [self.current_baseline]
        })
        self.data = pd.concat([self.data, df])

    @staticmethod
    def get_mass_point(hpos, vpos, width, height, scaling):
        """ text center coordinates """
        x, y = hpos + 0.5 * width, vpos + 0.5 * height
        return int(scaling[0]*x), int(scaling[1]*y)

    @staticmethod
    def extract_data(path, scaling, inverse=False):
        try:
            parser = xml.sax.make_parser()
            counter = AltoParser(scaling, inverse=inverse)
            parser.setContentHandler(counter)
            parser.parse(path)
            return counter.data
        except:
            print("*** Warning: alto-parser failed on", path)
            return pd.DataFrame({'Value': [], 'x': [], 'y': [], "Baseline": []})
```