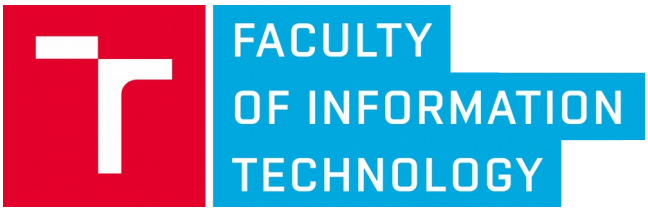

Manuál k Software pro adaptabilní rozpoznávání textu starých tisků

Michal Hradiš, Martin Kišš, Oldřich Kodym, Jan Kohút, Karel Beneš, Petr Buchal

Vysoké učení technické v Brně

Brno 2020

Tento dokument byl vytvořen s finanční podporou MK ČR v rámci programu NAKI II v projektu DG18P02OVV055 (Pokročilá extrakce a rozpoznávání obsahu tištěných a rukou psaných digitalizátů pro zvýšení jejich přístupnosti a využitelnosti).

Číslo a název projektu:

DG18P02OVV055	Pokročilá extrakce a rozpoznávání obsahu tištěných a rukou psaných digitalizátů pro zvýšení jejich přístupnosti a využitelnosti

Název a popis dílčího výstupu:

Manuál k Software pro adaptabilní rozpoznávání textu starých tisků
Tento dokument popisuje funkčnost a použití software pro automatický přepis textu tištěných dokumentů.

Jazyk dokumentu

Angličtina

Organizace a řešitel

Vysoké učení technické v Brně	Doc. RNDr. PAVEL SMRŽ Ph.D.

Availability

The software module is available from https://github.com/DCGM/pero-ocr.
Python module https://pypi.org/project/pero-ocr/, install as “pip install pero-ocr”
This OCR module is used py publicly available pero-ocr web application http://pero-ocr.fit.vutbr.cz/ .

License

BSD 3-Clause License

Usage

The package provides a full OCR pipeline including text paragraph detection, text line detection, text transcription, and text refinement using a language model.
The package can be used as a command line application or as a python package which provides a document processing class and a class which represents document page content.

Requirements

Linux/Windows
Python 3.6/3.7, numpy, numba, scikit-learn, scikit-image, OpenCV, tensorflow 1.15, PyTorch, shapely, pyamg, imgaug,
For faster processing: Cuda capable GPU with at least 4 GB RAM and CUDA toolkit.

Publicly available pretrained OCR models

Pretrained models can be downloaded from https://www.fit.vut.cz/~ihradis/pero/pero_eu_cz_print_newspapers_2020-10-09.tar.gz.
This package contains a layout analysis module which is suitable for most printed and handwritten documents together with OCR suitable for most european printed documents. The OCR module is specialized for low-quality czech newspapers digitized from microfilms, but it provides very good results for other poor-quality black/white documents and perfect text recognition for good quality documents in major european languages typeset in Antiqua fonts.

Command line application

Command line application is ./user_scripts/parse_folder.py. It is able to process images in a

directory using an OCR engine. It can render detected lines in an image and provide document content in Page XML and ALTO XML formats. Additionally, it is able to crop all text lines as rectangular regions of normalized size and save them into separate image files.

Command line parameters of parse_folder.py:

-c CONFIG, --config CONFIG	Path to config file which specifies OCR engine and other parameters of processing. The exact format will be described below.
-s, --skip-processed	Do not overwrite existing outputs.
--input-image-path INPUT_IMAGE_PATH	Path to a directory of images which should be processed.
-x INPUT_XML_PATH, --input-xml-path INPUT_XML_PATH	The tool allows users to process documents in separate steps, use the result of a previous processing step and only update some information. In such cases the previous results are stored as Page XML files and this option specifies a path to those files.
--output-xml-path	Directory where output Page XML should be stored.
--output-render-path	Directory where images with rendered text lines and paragraphs should be stored. This option is useful for fast and easy visual verification that the processing is configured correctly.
--output-line-path	Directory where images of cropped text lines should be stored.
--output-logit-path	Directory where logits (probabilities of characters) should be stored. This output is used only in advanced usage of the tool.
--output-alto-path	
--set-gpu	Sets the ID of a GPU which should be used by the tool. This is optional.

Configuration file

Configuration file has multiple sections, where each section generally defines a single step of a processing pipeline and section [PAGE_PARSER] defines which of the steps of the pipeline should be computed. In case that a processing stage is missing some needed inputs the processing exits with an error. Processing stages can be skipped only when the same information was computed previously and is loaded from an existing Page XML file. An example

of a configuration file with explanation follows:

```
[PAGE_PARSER]
# Which processing steps should be computed. The order of the steps is fixed. Options are yes/no.
RUN_LAYOUT_PARSER = yes  # Detection of paragraphs and text lines
RUN_LINE_CROPPER = yes     # Crop text lines. This is needed for OCR engine processing.
RUN_OCR = yes    # Run OCR.
RUN_DECODER = no      # Run optional decoding with a language model which improves

[PARSE_FOLDER]
# This section is used by parse_folder.py and can be used to specify paths to input and output directories. However, it is advisable to specify the directories using command line parameters. If the engine is used directly without parse_folder.py, this section can be omitted.

# Layout detection can be specified in multiple stages
# [LAYOUT_PARSER_X] where X specifies the order of processing.
[LAYOUT_PARSER_1]
METHOD = LAYOUT_CNN    # This method uses neural network to detect lines and paragraph.
DETECT_LINES = yes   # Should the method detect lines
DETECT_REGIONS = yes # Should the method detect text regions. This option can be set to “no” when text regions are defined in input Page XML files.
MERGE_LINES = no    # Optionally merges lines with similar horizontal positions inside a text region. This is usually not needed.
ADJUST_HEIGHTS = no # Adjust height of existing lines. This can be used only when text lines are specified in input Page XML files.
MODEL_PATH = ./ParseNet.pb # Path to a Pytorch network which processes images and detects lines and paragraph.
MAX_MEGAPIXELS = 5   # Maximum resolution of image which can be processed. The resolution is dynamic and adapts to text size in an image. This option effectively limits processing scale. 5 MPx fits into 5GB of GPU memory.
GPU_FRACTION = 0.5 # Fraction of GPU memory which should be allocated to this processing step.
USE_CPU = no  # Set this option to yes if you want to avoid using GPU. CPU processing can be 2-5x slower depending on the CPU type, GPU type and page size.
DOWNSAMPLE = 4 # Initial image downsampling factor for processing. 4 is generally a good option for most documents and if it is not optimal, the engine changes this downsampling factor based on text
```

```
size. If your text size if very big, you can increase the this number. If your text is tiny, you can try to decrease it
PAD = 52   # Do not change this value
DETECTION_THRESHOLD = 0.2   # Higher values result in more detected text lines. Lower values result in less detected text line or possibly in broken text lines. Generally, there should be no need to adjust this parameter

[LAYOUT_PARSER_2]
METHOD = REGION_SORTER_SMART  # This method orders paragraphs base on simple rules.

[LINE_CROPPER]
# This stage crops text lines for OCR. The parameters have to match the expectation of the OCR engine and generally should not be changed if you are using an OCR engine. You may want to change the values if you want to crop text lines and use them outside per-ocr.
INTERP = 2  # Base lines are interpolated by polynom of this order.
LINE_SCALE = 1.25   # This scale changes the height of the cropped regions in the source image.
LINE_HEIGHT = 40    # Resulting “normalized” height in pixels of the cropped text line images.

[OCR]
# This stage reads text from each cropped text line.
METHOD = pytorch_ocr   # This is the only supported method at the moment.
OCR_JSON = ./ocr_engine.json # This is a path to an OCR configuration file. The content of the OCR configuration file should not be changed.
```

Integration of the pero-ocr python module

This example shows how to directly use the OCR pipeline provided by pero-ocr package which can be used to integrate pero-ocr into other applications. Class `PageLayout` represents content of a single document page and can be loaded from Page XMl and exported to Page XML and ALTO XML formats. The OCR pipeline is represented by the PageParser class.

```
import os
import configparser
import cv2
from pero_ocr.document_ocr.layout import PageLayout
from pero_ocr.document_ocr.page_parser import PageParser

# Read config file.
```

```
config_path = "./config_file.ini"
config = configparser.ConfigParser()
config.read(config_path)

# Init the OCR pipeline.
# You have to specify config_path to be able to use relative paths
# inside the config file.
Page_parser = PageParser(config,
    config_path=os.path.dirname(config_path))

# Read the document page image.
input_image_path = "page_image.jpg"
image = cv2.imread(input_image_path, 1)

# Init empty page content.
# This object will be updated by the ocr pipeline. id can be any
string and it is used to identify the page.
page_layout = PageLayout(id=input_image_path,
     page_size=(image.shape[0], image.shape[1]))

# Process the image by the OCR pipeline
page_layout = page_parser.process_page(input_image_path, page_layout)

page_layout.to_pagexml('output_page.xml') # Save results as Page XML.
page_layout.to_altoxml('output_ALTO.xml') # Save results as ALTO XML.

# Render detected text regions and text lines into the image and
# save it into a file.
page_layout.render_to_image(image)
cv2.imwrite('page_image_render.jpg')

# Save each cropped text line in a separate .jpg file.
for region in page_layout.regions:
  for line in region.lines:
     cv2.imwrite(f'file_id-{line.id}.jpg', line.crop.astype(np.uint8))
```