dhsegment Documentation

Sofia ARES OLIVEIRA, Benoit SEGUIN

Aug 14, 2019

CONTENTS

1 Introduction 1

2 Quickstart 9

3 Reference guide 13

4 References 39

5 Changelog 41

6 Indices and tables 43

Bibliography 45

Python Module Index 47

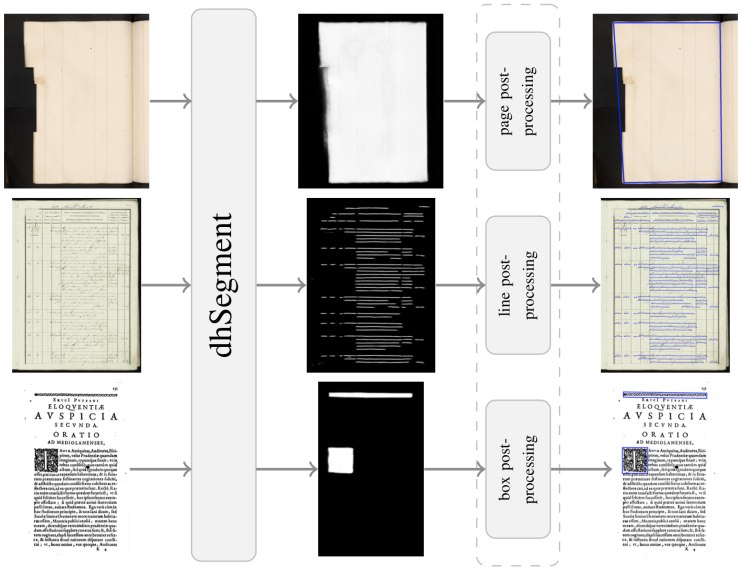
Index 49

CHAPTER

ONE

INTRODUCTION

1.1 What is dhSegment?



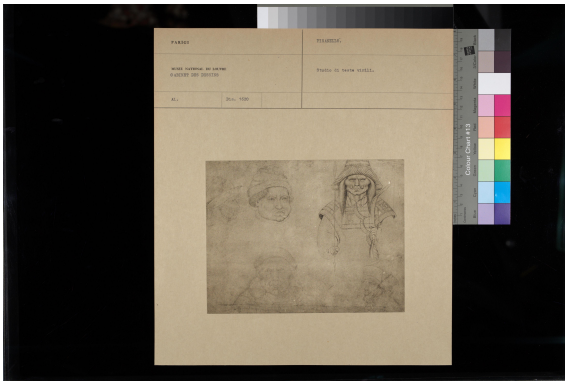
dhSegment is a generic approach for Historical Document Processing. It relies on a Convolutional Neural Network to do the heavy lifting of predicting pixelwise characteristics. Then simple image processing operations are provided to extract the components of interest (boxes, polygons, lines, masks, ...)

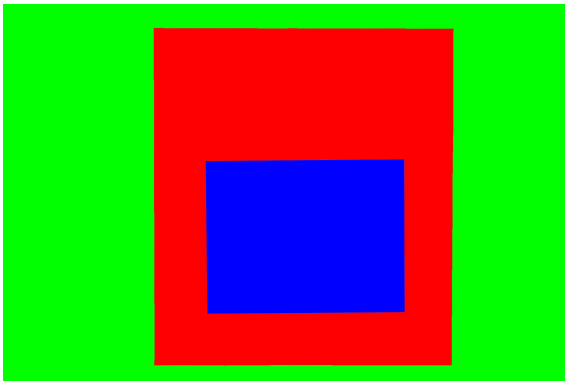
A few key facts:

- You only need to provide a list of images with annotated masks, which can easily be created with an image editing software (Gimp, Photoshop). You only need to draw the elements you care about!
- Allows to classify each pixel across multiple classes, with the possibility of assigning multiple labels per pixel.
- On-the-fly data augmentation, and efficient batching of batches.
- Leverages a state-of-the-art pre-trained network (Resnet50) to lower the need for training data and improve generalization.
- Monitor training on Tensorboard very easily.
- A list of simple image processing operations are already implemented such that the post-processing steps only take a couple of lines.

1.2 What sort of training data do I need?

Each training sample consists in an image of a document and its corresponding parts to be predicted.



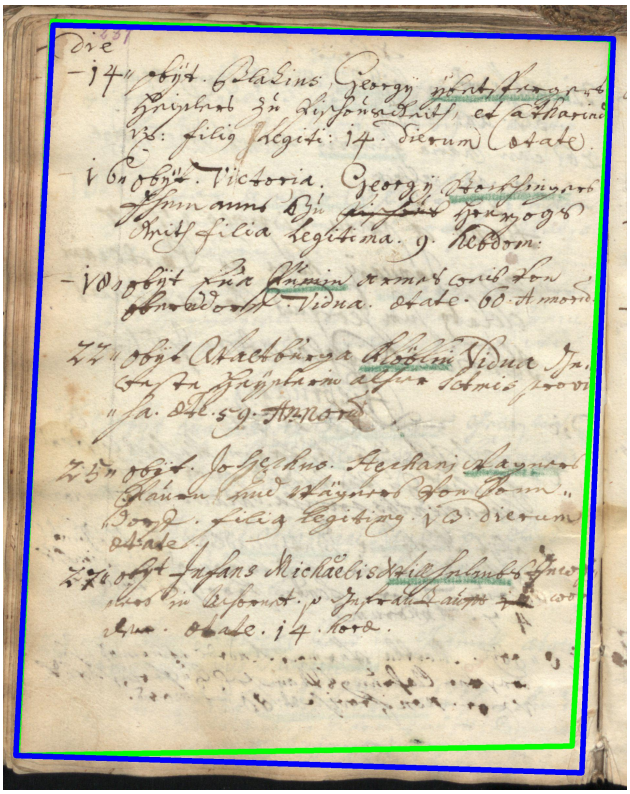


Additionally, a text file encoding the RGB values of the classes needs to be provided. In this case if we want the classes 'background', 'document' and 'photograph' to be respectively classes 0, 1, and 2 we need to encode their color line-by-line:

```
0 255 0
255 0 0
0 0 255
```

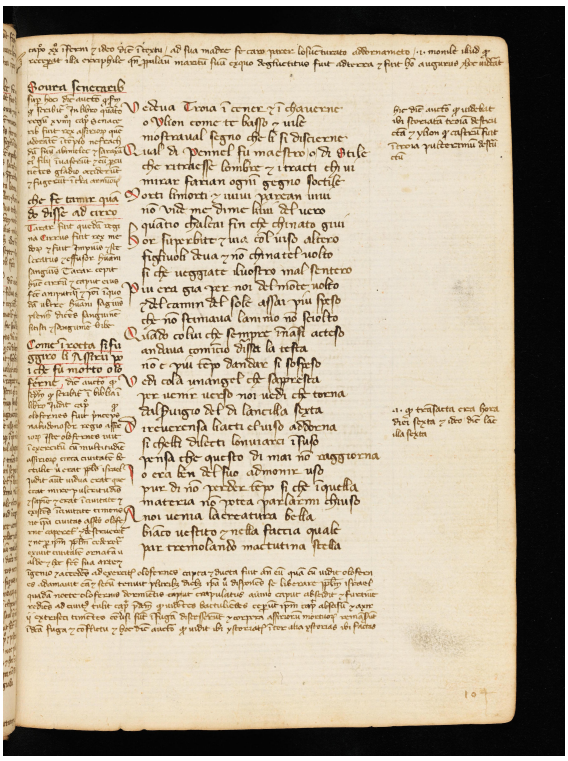
1.3 Use cases

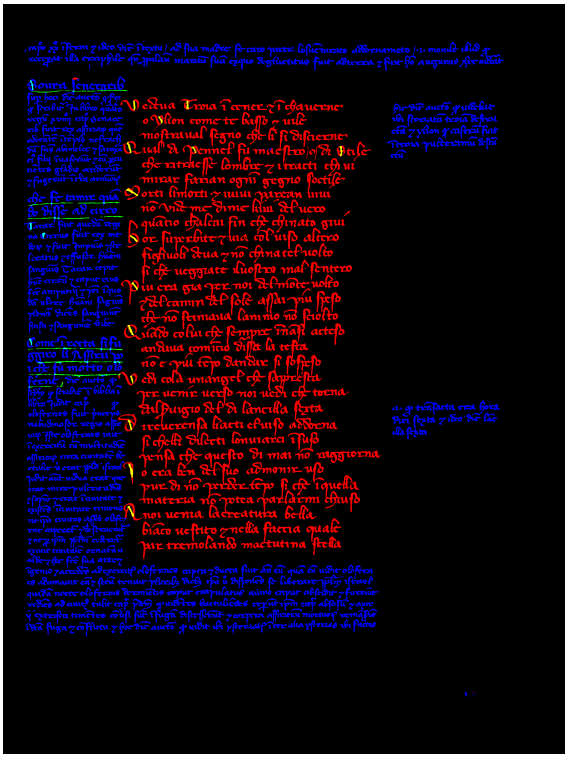
1.3.1 Page Segmentation



Dataset : READ-BAD [GruningLD+18] annotated by [TDW+17].

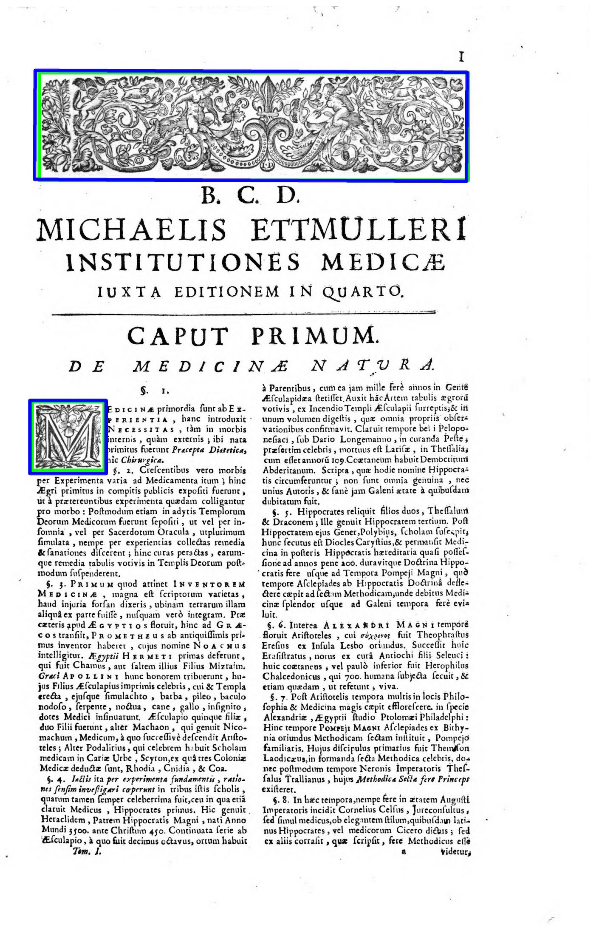
1.3.2 Layout Analysis





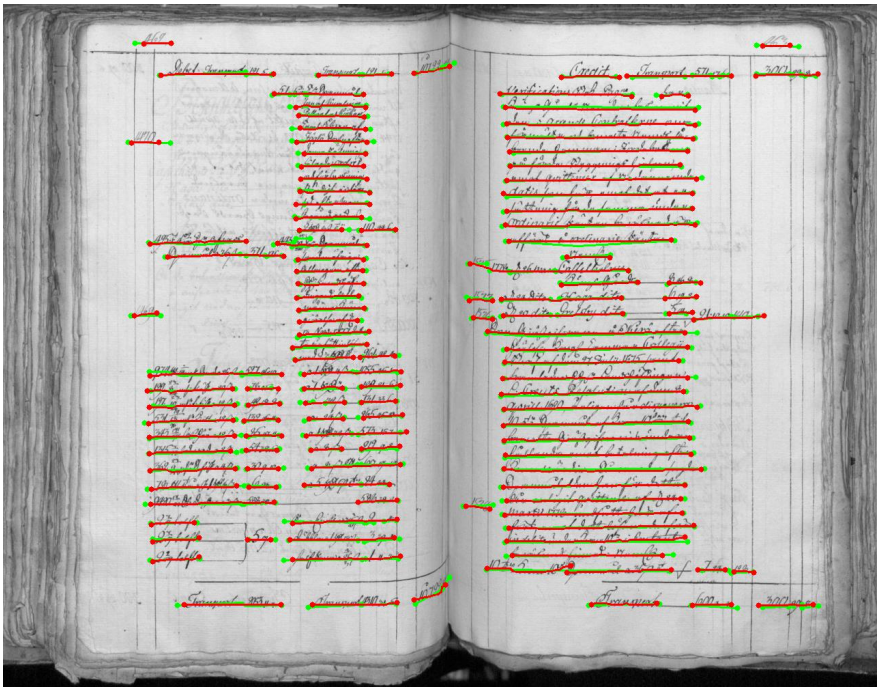
Dataset : DIVA-HisDB [SSE+16].

1.3.3 Ornament Extraction



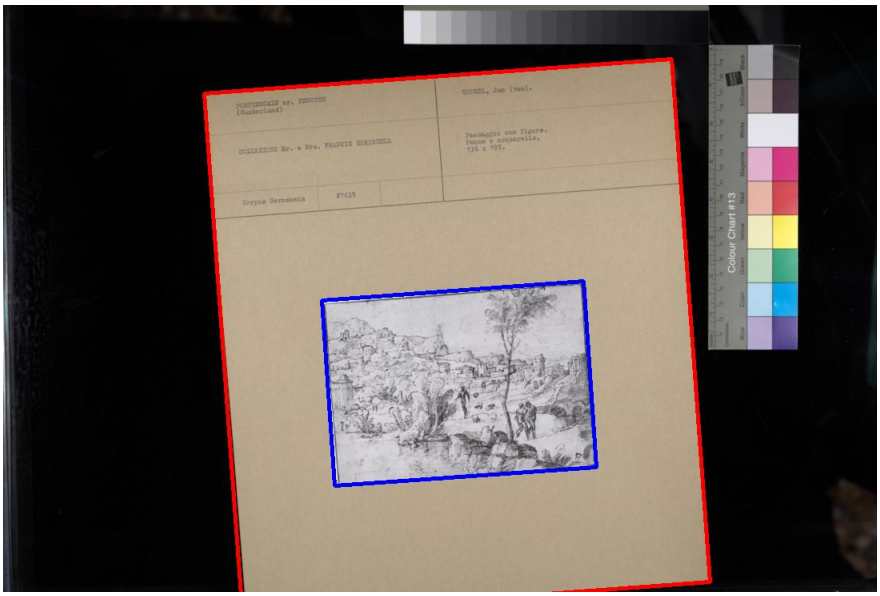
Dataset : BCU collection.

1.3.4 Line Detection



Dataset : READ-BAD [GruningLD+18].

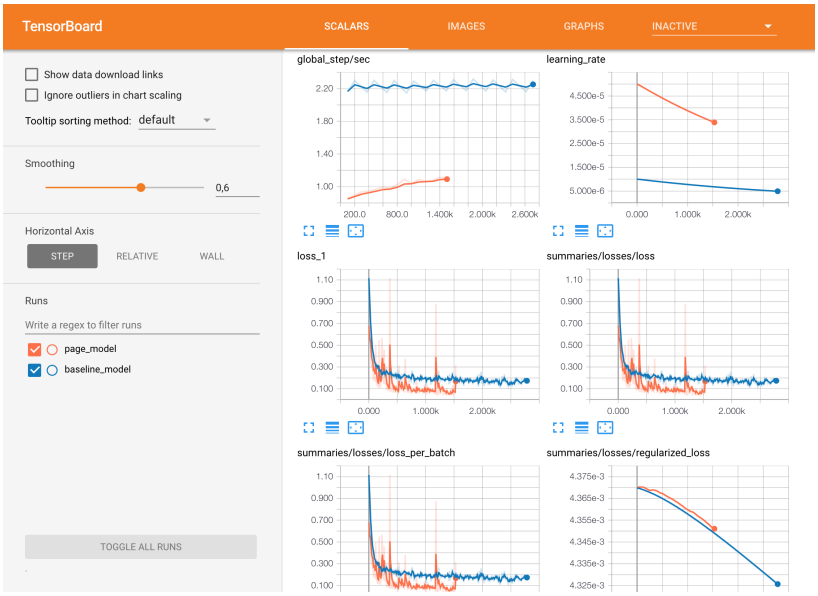
1.3.5 Document Segmentation

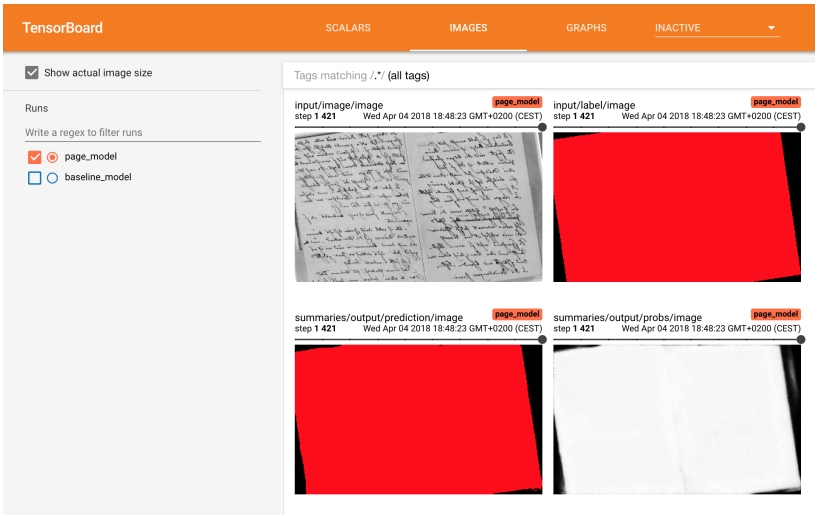


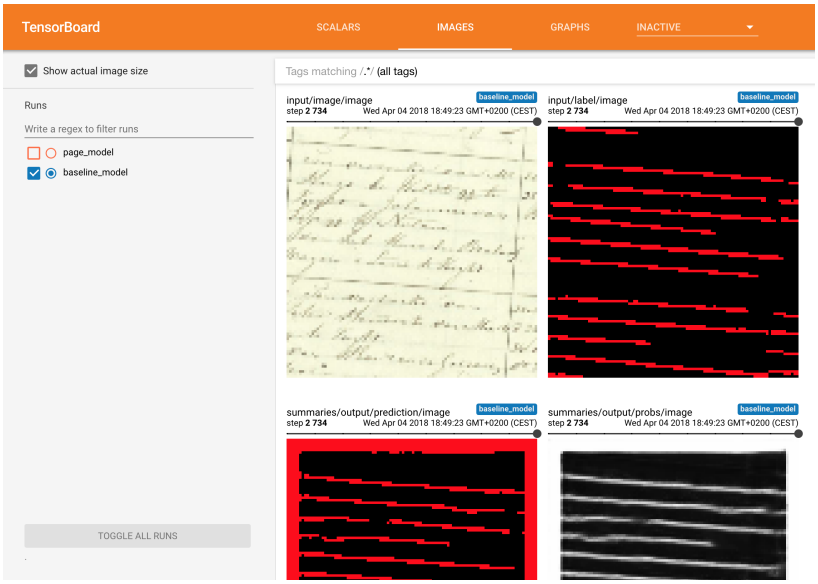
Dataset : Photo-collection from the Cini Foundation.

1.4 Tensorboard Integration

The TensorBoard integration allows to visualize your TensorFlow graph, plot metrics and show the images and predictions during the execution of the graph.






TensorBoard
SCALARS
IMAGES
GRAPHS
INACTIVE
Show actual image size
Runs
Write a regex to filter runs
page_model
baseline_model
TOGGLE ALL RUNS
Tags matching /.*/ (all tags)
input/image/image
baseline_model
step 2 734
Wed Apr 04 2018 18:49:23 GMT+0200 (CEST)
input/label/image
baseline_model
step 2 734
Wed Apr 04 2018 18:49:23 GMT+0200 (CEST)
summaries/output/prediction/image
baseline_model
step 2 734
Wed Apr 04 2018 18:49:23 GMT+0200 (CEST)
summaries/output/probs/image
baseline_model
step 2 734
Wed Apr 04 2018 18:49:23 GMT+0200 (CEST)

CHAPTER

TWO

QUICKSTART

2.1 Installation

It is recommended to install `tensorflow` (or `tensorflow-gpu`) independently using Anaconda distribution, in order to make sure all dependencies are properly installed.

1. Clone the repository using `git clone https://github.com/dhlab-epfl/dhSegment.git`
2. Install Anaconda or Miniconda (installation procedure)
3. Create a virtual environment and activate it

```
conda create -n dh_segment python=3.6
source activate dh_segment
```

4. Install dhSegment dependencies with `pip install git+https://github.com/dhlab-epfl/dhSegment`
5. Install TensorFlow 1.13 with conda `conda install tensorflow-gpu=1.13.1`.

2.2 Creating groundtruth data

2.2.1 Using GIMP or Photoshop

Create directly your masks using your favorite image editor. You just have to draw the regions you want to extract with a different color for each label.

2.2.2 Using VGG Image Annotator (VIA)

VGG Image Annotator (VIA) is an image annotation tool that can be used to define regions in an image and create textual descriptions of those regions. You can either use it online or download the application.

From the exported annotations (in JSON format), you'll have to generate the corresponding image masks. See the *VGG Image Annotator helpers* in the `via` module.

When assigning attributes to your annotated regions, you should favour attributes of type "dropdown", "checkbox" and "radio" and avoid "text" type in order to ease the parsing of the exported file (avoid typos and formatting errors).

Example of how to create individual masks from VIA annotation file

```python
from dh_segment.io import via

collection = 'mycollection'
annotation_file = 'via_sample.json'
masks_dir = '/home/project/generated_masks'
images_dir = './my_images'

# Load all the data in the annotation file
# (the file may be an exported project or an export of the annotations)
via_data = via.load_annotation_data(annotation_file)

# In the case of an exported project file, you can set ``only_img_annotations=True``
# to get only the image annotations
via_annotations = via.load_annotation_data(annotation_file, only_img_annotations=True)

# Collect the annotated regions
working_items = via.collect_working_items(via_annotations, collection, images_dir)

# Collect the attributes and options
if '_via_attributes' in via_data.keys():
    list_attributes = via.parse_via_attributes(via_data['_via_attributes'])
else:
    list_attributes = via.get_via_attributes(via_annotations)

# Create one mask per option per attribute
via.create_masks(masks_dir, working_items, list_attributes, collection)
```

2.3 Training

Note: A good nvidia GPU (6GB RAM at least) is most likely necessary to train your own models. We assume CUDA and cuDNN are installed.

Input data

You need to have your training data in a folder containing `images` folder and `labels` folder. The pairs (images, labels) need to have the same name (it is not mandatory to have the same extension file, however we recommend having the label images as `.png` files).

The annotated images in `label` folder are (usually) RGB images with the regions to segment annotated with a specific color.

Note: It is now also possible to use a *csv* file containing the pairs `original_image_filename`, `label_image_filename` as input data.

To input a `csv` file instead of the two folders `images` and `labels`, the content should be formatted in the following way:

```
mypath/myfolder/original_image_filename1,mypath/myfolder/label_image_filename1
mypath/myfolder/original_image_filename2,mypath/myfolder/label_image_filename2
```

The class.txt file

The file containing the classes has the format shown below, where each row corresponds to one class (including 'negative' or 'background' class) and each row has 3 values for the 3 RGB values. Of course each class needs to have a different code.

```
classes.txt

0 0 0
0 255 0
...
```

Config file with ''sacred''

sacred package is used to deal with experiments and trainings. Have a look at the documentation to use it properly.

In order to train a model, you should run `python train.py with <config.json>`

2.3.1 Multilabel classification training

In case you want to be able to assign multiple labels to elements, the `classes.txt` file must be changed. Besides the color code, you need to add an *attribution* code to each color. The attribution code has length *n_classes* and indicates which classes are assigned to the color.

Take for example 3 classes {A, B, C} and the following possible labelling combinations:

- A (color code `(0 255 0)`) with attribution code `1 0 0`
- B (color code `(255 0 0)`) with attribution code `0 1 0`
- C (color code `(0 0 255)`) with attribution code `0 0 1`
- AB (color code `(128 128 128)`) with attribution code `1 1 0`
- BC (color code `(0 255 255)`) with attribution code `0 1 1`

The attributions code has value `1` when the label is assigned and `0` when it's not. (The attribution code `1 0 1` would mean that the color annotates elements that belong to classes A and C)

In our example the `classes.txt` file would then look like :

```
classes.txt

0 0 0 0 0 0
0 255 0 1 0 0
255 0 0 0 1 0
0 0 255 0 0 1
128 128 128 1 1 0
0 255 255 0 1 1
```

2.4 Demo

This demo shows the usage of dhSegment for page document extraction. It trains a model from scratch (optional) using the READ-BAD dataset [GruningLD+18] and the annotations of Pagenet [TDW+17] (annotator1 is used). In order to limit memory usage, the images in the dataset we provide have been downsized to have 1M pixels each.

How to

0. If you have not yet done so, clone the repository :

```
git clone https://github.com/dhlab-epfl/dhSegment.git
```

1. Get the annotated dataset here, which already contains the folders `images` and `labels` for training, validation and testing set. Unzip it into `demo/pages`.

```
cd demo/
wget https://github.com/dhlab-epfl/dhSegment/releases/download/v0.2/pages.zip
unzip pages.zip
cd ..
```

2. (Only needed if training from scratch) Download the pretrained weights for ResNet :

```
cd pretrained_models/
python download_resnet_pretrained_model.py
cd ..
```

3. You can train the model from scratch with: `python train.py with demo/demo_config.json` but because this takes quite some time, we recommend you to skip this and just download the provided model (download and unzip it in `demo/model`)

```
cd demo/
wget https://github.com/dhlab-epfl/dhSegment/releases/download/v0.2/model.zip
unzip model.zip
cd ..
```

4. (Only if training from scratch) You can visualize the progresses in tensorboard by running `tensorboard --logdir .` in the `demo` folder.

5. Run `python demo.py`

6. Have a look at the results in `demo/processed_images`

CHAPTER

THREE

REFERENCE GUIDE

3.1 Network architecture

Here is the dhsegment architecture definition

---

dh_segment.network.**inference_vgg16**(*images*, *params*, *num_classes*, *use_batch_norm=False*, *weight_decay=0.0*, *is_training=False*)

Return type tensorflow.Tensor

dh_segment.network.**inference_resnet_v1_50**(*images*, *params*, *num_classes*, *use_batch_norm=False*, *weight_decay=0.0*, *is_training=False*)

Return type tensorflow.Tensor

dh_segment.network.**inference_u_net**(*images*, *params*, *num_classes*, *use_batch_norm=False*, *weight_decay=0.0*, *is_training=False*)

Return type tensorflow.Tensor

dh_segment.network.**vgg_16_fn**(*input_tensor*, *scope='vgg_16'*, *blocks=5*, *weight_decay=0.0005*)

Return type (tensorflow.Tensor, <class 'list'>)

dh_segment.network.**resnet_v1_50_fn**(*input_tensor*, *is_training=False*, *blocks=4*, *weight_decay=0.0001*, *renorm=True*, *corrected_version=False*)

Return type tensorflow.Tensor

3.2 Input / Output

The *dh_segment.io* module implements input / output functions and classes.

3.2.1 Input functions for `tf.Estimator`

Input function

input_fn(input_data, params[, ...])	Input_fn for estimator

Data augmentation

data_augmentation_fn(input_image, label_image)	Applies data augmentation to both images and label images.
extract_patches_fn(image, patch_shape, offsets)	Will cut a given image into patches.
rotate_crop(image, rotation[, crop, ...])	Rotates and crops the images.

Resizing function

resize_image(image, size[, interpolation])	Resizes the image
load_and_resize_image(filename, channels[, ...])	Loads an image from its filename and resizes it to the desired output size.

3.2.2 Tensorflow serving functions

serving_input_filename(resized_size)
serving_input_image()

3.2.3 PAGE XML and JSON import / export

PAGE classes

PAGE.Point(y, x)	Point (x,y) class.
PAGE.Text([text_equiv, alternatives, score])	Text entity produced by a transcription system.
PAGE.Border([coords, id])	Region containing the page.
PAGE.TextRegion([id, coords, text_lines, ...])	Region containing text lines.
PAGE.TextLine([id, coords, baseline, text, ...])	Region corresponding to a text line.
PAGE.GraphicRegion([id, coords, ...])	Region containing simple graphics.
PAGE.TableRegion([id, coords, rows, ...])	Tabular data in any form.
PAGE.SeparatorRegion(id[, coords, ...])	Lines separating columns or paragraphs.
PAGE.GroupSegment([id, coords, segment_ids, ...])	Set of regions that make a bigger region (group).
PAGE.Metadata([creator, created, ...])	Metadata information.
PAGE.Page(**kwargs)	Class following PAGE-XML object.

Abstract classes

PAGE.BaseElement	Base page element class.
PAGE.Region([id, coords, custom_attribute])	Region base class.

Parsing and helpers

PAGE.parse_file(filename)	Parses the files to create the corresponding `Page` object.
PAGE.json_serialize(dict_to_serialize[, ...])	Serialize a dictionary in order to export it.

3.2.4 VGG Image Annotator helpers

VIA objects

via.WorkingItem	A container for annotated images.
via.VIAttribute	A container for VIA attributes.

Creating masks with VIA annotations

via.load_annotation_data(via_data_filename)	Load the content of via annotation files.
via.export_annotation_dict(annotation_dict, ...)	Export the annotations to json file.
via.get_annotations_per_file(via_dict, name_file)	From VIA json content, get annotations relative to the given *name_file*.
via.parse_via_attributes(via_attributes)	Parses the VIA attribute dictionary and returns a list of VIAttribute instances
via.get_via_attributes(annotation_dict[, ...])	Gets the attributes of the annotated data and returns a list of *VIAttribute*.
via.collect_working_items(via_annotations, ...)	Given VIA annotation input, collect all info on *WorkingItem* object.
via.create_masks(masks_dir, working_items, ...)	For each annotation, create a corresponding binary mask and resize it (h = 2000).

Formatting in VIA JSON format

via.create_via_region_from_coordinates(...)	Formats coordinates to a VIA region (dict).
via.create_via_annotation_single_image(...)	Returns a dictionary item {key: annotation} in VIA format to further export to .json file

---

dh_segment.io.**input_fn**(*input_data*, *params*, *input_label_dir=None*, *data_augmentation=False*, *batch_size=5*, *make_patches=False*, *num_epochs=1*, *num_threads=4*, *image_summaries=False*)

Input_fn for estimator

Parameters

- **input_data** (Union[str, List[str]]) – input data. It can be a directory containing the images, it can be a list of image filenames, or it can be a path to a csv file.
- **params** (dict) – params from utils.Params object
- **input_label_dir** (Optional[str]) – directory containing the label images
- **data_augmentation** (bool) – boolean, if True will scale, roatate, ... the images
- **batch_size** (int) – size of the bach
- **make_patches** (bool) – bool, whether to make patches (crop image in smaller pieces) or not
- **num_epochs** (int) – number of epochs to cycle trough data (set it to None for infinite repeat)
- **num_threads** (int) – number of thread to use in parallele when usin tf.data.Dataset.map

- **image_summaries** (bool) – boolean, whether to make tf.Summary to watch on tensorboard

Returns fn

dh_segment.io.**serving_input_filename**(*resized_size*)

dh_segment.io.**serving_input_image**()

dh_segment.io.**data_augmentation_fn**(*input_image*, *label_image*, *flip_lr=True*, *flip_ud=True*, *color=True*)

Applies data augmentation to both images and label images. Includes left-right flip, up-down flip and color change.

Parameters

- **input_image** (*tensorflow.Tensor*) – images to be augmented [B, H, W, C]
- **label_image** (*tensorflow.Tensor*) – corresponding label images [B, H, W, C]
- **flip_lr** (bool) – option to flip image in left-right direction
- **flip_ud** (bool) – option to flip image in up-down direction
- **color** (bool) – option to change color of images

Return type (tensorflow.Tensor, tensorflow.Tensor)

Returns the tuple (augmented images, augmented label images) [B, H, W, C]

dh_segment.io.**rotate_crop**(*image, rotation, crop=True, minimum_shape=[0, 0], interpolation='NEAREST'*)

Rotates and crops the images.

Parameters

- **image** (*tensorflow.Tensor*) – image to be rotated and cropped [H, W, C]
- **rotation** (float) – angle of rotation (in radians)
- **crop** (bool) – option to crop rotated image to avoid black borders due to rotation
- **minimum_shape** (Tuple[int, int]) – minimum shape of the rotated image / cropped image
- **interpolation** (str) – which interpolation to use NEAREST or BILINEAR

Return type tensorflow.Tensor

Returns

dh_segment.io.**resize_image**(*image*, *size*, *interpolation='BILINEAR'*)

Resizes the image

Parameters

- **image** (*tensorflow.Tensor*) – image to be resized [H, W, C]
- **size** (int) – size of the resized image (in pixels)
- **interpolation** (str) – which interpolation to use, NEAREST or BILINEAR

Return type tensorflow.Tensor

Returns resized image

dh_segment.io.**load_and_resize_image**(*filename*, *channels*, *size=None*, *interpolation='BILINEAR'*)

Loads an image from its filename and resizes it to the desired output size.

Parameters

- **filename** (str) – string tensor
- **channels** (int) – number of channels for the decoded image
- **size** (Optional[int]) – number of desired pixels in the resized image, tf.Tensor or int (None for no resizing)
- **interpolation** (str) –
- **return_original_shape** – returns the original shape of the image before resizing if this flag is True

Return type tensorflow.Tensor

Returns decoded and resized float32 tensor [h, w, channels],

dh_segment.io.**extract_patches_fn**(*image*, *patch_shape*, *offsets*)

Will cut a given image into patches.

Parameters

- **image** (*tensorflow.Tensor*) – tf.Tensor
- **patch_shape** (Tuple[int, int]) – shape of the extracted patches [h, w]
- **offsets** (Tuple[int, int]) – offset to add to the origin of first patch top-right coordinate, useful during data augmentation to have slighlty different patches each time. This value will be multiplied by [h/2, w/2] (range values [0,1])

Return type tensorflow.Tensor

Returns patches [batch_patches, h, w, c]

dh_segment.io.**local_entropy**(*tf_binary_img*, *sigma=3*)

Parameters

- **tf_binary_img** (*tensorflow.Tensor*) –
- **sigma** (float) –

Return type tensorflow.Tensor

Returns

class dh_segment.io.PAGE.**BaseElement**

Base page element class. (Abstract)

classmethod check_tag(*tag*)

classmethod full_tag()

Return type str

tag = None

class dh_segment.io.PAGE.**Border**(*coords=None*, *id=None*)

Region containing the page. It is the border of the actual page of the document (if the scanned image contains parts not belonging to the page).

Variables **coords** – coordinates of the *Border* region

classmethod from_dict(*dictionary*)

Return type *Border*

classmethod from_xml(*e*)

Return type *Border*

tag = 'Border'

to_dict(*non_serializable_keys=[]*)

Return type dict

to_xml()

Return type Element

class dh_segment.io.PAGE.**GraphicRegion**(*id=None*, *coords=None*, *custom_attribute=None*)

Region containing simple graphics. Company logos for example should be marked as graphic regions.

Variables

- **id** – identifier of the *GraphicRegion*
- **coords** – coordinates of the *GraphicRegion*

classmethod from_dict(*dictionary*)

From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *GraphicRegion*

Returns non serialized dictionary

classmethod from_xml(*e*)

Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** – a xml etree

Return type *GraphicRegion*

Returns a dictionary with keys 'id' and 'coords'

tag = 'GraphicRegion'

to_xml(*name_element='GraphicRegion'*)

Converts a *Region* object to a xml structure

Parameters **name_element** – name of the object (optional)

Return type Element

Returns a etree structure

class dh_segment.io.PAGE.**GroupSegment**(*id=None*, *coords=None*, *segment_ids=None*, *custom_attribute=None*)

Set of regions that make a bigger region (group). *GroupSegment* is a region containing several *TextLine* and that form a bigger region. It is used mainly to make line / column regions. Only for JSON export (no PAGE XML correspondence).

Variables

- **id** – identifier of the *GroupSegment*
- **coords** – coordinates of the *GroupSegment*
- **segment_ids** – list of the regions ids belonging to the group

classmethod from_dict(*dictionary*)

From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *GroupSegment*

Returns non serialized dictionary

class dh_segment.io.PAGE.**Metadata**(*creator=None*, *created=None*, *last_change=None*, *comments=None*)

Metadata information.

Variables

- **creator** – name of the process of person that created the exported file
- **created** – time of creation of the file
- **last_change** – time of last modification of the file
- **comments** – comments on the process

classmethod from_dict(*dictionary*)

Return type *Metadata*

classmethod from_xml(*e*)

Return type *Metadata*

tag = 'Metadata'

to_dict()

to_xml()

Return type Element

class dh_segment.io.PAGE.**Page**(***kwargs*)

Class following PAGE-XML object. This class is used to represent the information of the processed image. It is possible to export this info as PAGE-XML or JSON format.

Variables

- **image_filename** – filename of the image
- **image_width** – width of the original image
- **image_height** – height of the original image
- **text_regions** – list of *TextRegion*
- **graphic_regions** – list of *GraphicRegion*
- **page_border** – *Border* of the page
- **separator_regions** – list of *SeparatorRegion*
- **table_regions** – list of *TableRegion*
- **metadata** – *Metadata* of the image and process
- **line_groups** – list of *GroupSegment* forming lines
- **column_groups** – list of *GroupSegment* forming columns

draw_baselines(*img_canvas*, *color=(255, 0, 0)*, *thickness=2*, *endpoint_radius=4*, *autoscale=True*)

Given an image, draws the TextLines.baselines.

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.

- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **thickness** (int) – the thickness of the line
- **endpoint_radius** (int) – the radius of the endpoints of line s(first and last coordinates of line)
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_column_groups(*img_canvas*, *color=(0*, *255*, *0)*, *fill=False*, *thickness=5*, *autoscale=True*)
It will draw column groups (in case of a table). This is only valid when parsing JSON files.

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **fill** (bool) – either to fill the region (True) of only draw the external contours (False)
- **thickness** (int) – in case fill=False the thickness of the line
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_graphic_regions(*img_canvas*, *color=(255*, *0*, *0)*, *fill=True*, *thickness=3*, *autoscale=True*)
Given an image, draws the GraphicRegions, either fills it (fill=True) or draws the contours (fill=False)

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **fill** (bool) – either to fill the region (True) of only draw the external contours (False)
- **thickness** (int) – in case fill=True the thickness of the line
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_line_groups(*img_canvas*, *color=(0*, *255*, *0)*, *fill=False*, *thickness=5*, *autoscale=True*)
It will draw line groups. This is only valid when parsing JSON files.

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **fill** (bool) – either to fill the region (True) of only draw the external contours (False)
- **thickness** (int) – in case fill=False the thickness of the line
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_lines(*img_canvas*, *color=(255, 0, 0)*, *thickness=2*, *fill=True*, *autoscale=True*)
Given an image, draws the polygons containing text lines, i.e TextLines.coords

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **thickness** (int) – the thickness of the line
- **fill** (bool) – if True fills the polygon
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_page_border(*img_canvas*, *color=(255, 0, 0)*, *fill=True*, *thickness=5*, *autoscale=True*)
Given an image, draws the page border, either fills it (fill=True) or draws the contours (fill=False)

Parameters

- **img_canvas** – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **fill** (bool) – either to fill the region (True) of only draw the external contours (False)
- **thickness** (int) – in case fill=True the thickness of the line
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_separator_lines(*img_canvas*, *color=(0, 255, 0)*, *thickness=3*, *filter_by_id=''*, *autoscale=True*)
Given an image, draws the SeparatorRegion.

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **thickness** (int) – thickness of the line
- **filter_by_id** (str) – string to filter the lines by id. For example vertical/horizontal lines can be filtered if 'vertical' or 'horizontal' is mentioned in the id.
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_text(*img_canvas*, *color=(255, 0, 0)*, *thickness=5*, *font=cv2.FONT_HERSHEY_SIMPLEX*, *font_scale=1.0*, *autoscale=True*)
Writes the text of the TextLine on the given image.

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace
- **color** (Tuple[int, int, int]) – (R, G, B) value color

- **thickness** (int) – the thickness of the characters
- **font** – the type of font (cv2 constant)
- **font_scale** (float) – the scale of font
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

draw_text_regions(*img_canvas*, *color=(255*, *0*, *0)*, *fill=True*, *thickness=3*, *autoscale=True*)
Given an image, draws the TextRegions, either fills it (fill=True) or draws the contours (fill=False)

Parameters

- **img_canvas** (ndarray) – 3 channel image in which the region will be drawn. The image is modified inplace.
- **color** (Tuple[int, int, int]) – (R, G, B) value color
- **fill** (bool) – either to fill the region (True) of only draw the external contours (False)
- **thickness** (int) – in case fill=True the thickness of the line
- **autoscale** (bool) – whether to scale the coordinates to the size of img_canvas. If True, it will use the dimensions provided in Page.image_width and Page.image_height to compute the scaling ratio

classmethod from_dict(*dictionary*)

Return type *Page*

classmethod from_xml(*e*)

Return type *Page*

tag = 'Page'

to_json()

Return type dict

to_xml()

Return type Element

write_to_file(*filename*, *creator_name='dhSegment'*, *comments=''*)
Export Page object to json or page-xml format. Will assume the format based on the extension of the filename, if there is no extension will export as an xml file.

Parameters

- **filename** (str) – filename of the file to be exported
- **creator_name** (str) – name of the creator (process or person) creating the file
- **comments** (str) – optionnal comment to add to the metadata of the file.

Return type None

class dh_segment.io.PAGE.**Point**(*y*, *x*)
Point (x,y) class.

Variables

- **y** – vertical coordinate
- **x** – horizontal coordinate

classmethod array_to_list(*array*)
Converts an *np.array* to a list of coordinates

Parameters **array** (ndarray) – an array of coordinates. Must be of shape (N, 2)

Return type list

Returns list of coordinates, shape (N,2)

classmethod array_to_point(*array*)
Converts an *np.array* to a list of *Point*

Parameters **array** (ndarray) – an array of coordinates. Must be of shape (N, 2)

Return type list

Returns list of *Point*

classmethod cv2_to_point_list(*cv2_array*)
Converts an opencv-formatted set of coordinates to a list of *Point*

Parameters **cv2_array** (ndarray) – opencv-formatted set of coordinates, shape (N,1,2)

Return type List[*Point*]

Returns list of *Point*

classmethod list_from_xml(*etree_elem*)
Converts a PAGEXML-formatted set of coordinates to a list of *Point*

Parameters **etree_elem** (Element) – etree XML element containing a set of coordinates

Return type List[*Point*]

Returns a list of coordinates as *Point*

classmethod list_point_to_string(*list_points*)
Converts a list of *Point* to a string 'x,y'

Parameters **list_points** (List[*Point*]) – list of coordinates with *Point* format

Return type str

Returns a string with the coordinates

classmethod list_to_cv2poly(*list_points*)
Converts a list of *Point* to opencv format set of coordinates

Parameters **list_points** (List[*Point*]) – set of coordinates

Return type ndarray

Returns opencv-formatted set of points, shape (N,1,2)

classmethod list_to_point(*list_coords*)
Converts a list of coordinates to a list of *Point*

Parameters **list_coords** (list) – list of coordinates, shape (N, 2)

Return type List[*Point*]

Returns list of *Point*

classmethod point_to_list(*points*)
Converts a list of *Point* to a list of coordinates

Parameters **points** (List[*Point*]) – list of Points

Return type list

Returns list of shape (N,2)

to_dict()

class dh_segment.io.PAGE.**Region**(*id=None*, *coords=None*, *custom_attribute=None*)
Region base class. (Abstract) This is the superclass for all the extracted regions

Variables

- **id** – identifier of the *Region*
- **coords** – coordinates of the *Region*
- **custom_attribute** – Any custom attribute that may be linked with the region (usually this is added in PAGEXML files, not in JSON files)

classmethod from_dict(*dictionary*)
From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type dict

Returns non serialized dictionary

classmethod from_xml(*etree_element*)
Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** (Element) – a xml etree

Return type dict

Returns a dictionary with keys 'id' and 'coords'

tag = 'Region'

to_dict(*non_serializable_keys=[]*)
Converts a *Region* object to a dictionary.

Parameters **non_serializable_keys** (List[str]) – list of keys that can't be directly serialized and that need some internal serialization

Return type dict

Returns a dictionary with the atributes of the object serialized

to_xml(*name_element=None*)
Converts a *Region* object to a xml structure

Parameters **name_element** (Optional[str]) – name of the object (optional)

Return type Element

Returns a etree structure

class dh_segment.io.PAGE.**SeparatorRegion**(*id*, *coords=None*, *custom_attribute=None*)
Lines separating columns or paragraphs. Separators are lines that lie between columns and paragraphs and can be used to logically separate different articles from each other.

Variables

- **id** – identifier of the *SeparatorRegion*
- **coords** – coordinates of the *SeparatorRegion*

classmethod from_dict(*dictionary*)
From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *SeparatorRegion*

Returns non serialized dictionary

classmethod from_xml(*e*)
Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** – a xml etree

Return type *SeparatorRegion*

Returns a dictionary with keys 'id' and 'coords'

tag = 'SeparatorRegion'

to_xml(*name_element='SeparatorRegion'*)
Converts a *Region* object to a xml structure

Parameters **name_element** – name of the object (optional)

Return type Element

Returns a etree structure

class dh_segment.io.PAGE.**TableRegion**(*id=None*, *coords=None*, *rows=None*, *columns=None*, *embedded_text=None*, *custom_attribute=None*)
Tabular data in any form. Tabular data is represented with a table region. Rows and columns may or may not have separator lines; these lines are not separator regions.

Variables

- **id** – identifier of the *TableRegion*
- **coords** – coordinates of the *TableRegion*
- **rows** – number of rows in the table
- **columns** – number of columns in the table
- **embedded_text** – if text is embedded in the table

classmethod from_dict(*dictionary*)
From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *TableRegion*

Returns non serialized dictionary

classmethod from_xml(*e*)
Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** – a xml etree

Return type *TableRegion*

Returns a dictionary with keys 'id' and 'coords'

tag = 'TableRegion'

to_xml(*name_element='TableRegion'*)
Converts a *Region* object to a xml structure

Parameters **name_element** – name of the object (optional)

Return type Element

Returns a etree structure

class dh_segment.io.PAGE.**Text**(*text_equiv=None*, *alternatives=None*, *score=None*)
Text entity produced by a transcription system.

Variables

- **text_equiv** – the transcription of the text
- **alternatives** – alternative transcriptions
- **score** – the confidence of the transcription output by the transcription system

to_dict()

Return type dict

class dh_segment.io.PAGE.**TextLine**(*id=None*, *coords=None*, *baseline=None*, *text=None*, *line_group_id=None*, *column_group_id=None*, *custom_attribute=None*)
Region corresponding to a text line.

Variables

- **id** – identifier of the *TextLine*
- **coords** – coordinates of the *Texline* line
- **baseline** – coordinates of the *Texline* baseline
- **text** – *Text* class containing the transcription of the *TextLine*
- **line_group_id** – identifier of the line group the instance belongs to
- **column_group_id** – identifier of the column group the instance belongs to
- **custom_attribute** – Any custom attribute that may be linked with the region (usually this is added in PAGEXML files, not in JSON files)

classmethod from_array(*cv2_coords=None*, *baseline_coords=None*, *text_equiv=None*, *id=None*)

classmethod from_dict(*dictionary*)
From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *TextLine*

Returns non serialized dictionary

classmethod from_xml(*etree_element*)
Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** (Element) – a xml etree

Return type *TextLine*

Returns a dictionary with keys 'id' and 'coords'

scale_baseline_points(*ratio*)
Scales the points of the baseline by a factor *ratio*.

Parameters **ratio** (float) – factor to rescale the baseline coordinates

tag = 'TextLine'

to_dict(*non_serializable_keys=[]*)
Converts a *Region* object to a dictionary.

Parameters **non_serializable_keys** (List[str]) – list of keys that can't be directly serialized and that need some internal serialization

Returns a dictionary with the atributes of the object serialized

to_xml(*name_element='TextLine'*)
Converts a *Region* object to a xml structure

Parameters **name_element** – name of the object (optional)

Return type Element

Returns a etree structure

class dh_segment.io.PAGE.**TextRegion**(*id=None*, *coords=None*, *text_lines=None*, *text_equiv=''*, *region_type=None*, *custom_attribute=None*)
Region containing text lines. It can represent a paragraph or a page for instance.

Variables

- **id** – identifier of the *TextRegion*
- **coords** – coordinates of the *TextRegion*
- **text_equiv** – the resulting text of the *Text* contained in the *TextLines*
- **text_lines** – a list of *TextLine* objects
- **region_type** – the type of a TextRegion (can be any string). Example : header, paragraph, page-number...
- **custom_attribute** – Any custom attribute that may be linked with the region (usually this is added in PAGEXML files, not in JSON files)

classmethod from_dict(*dictionary*)
From a seralized dictionary creates a dictionary of the atributes (non serialized)

Parameters **dictionary** (dict) – serialized dictionary

Return type *TextRegion*

Returns non serialized dictionary

classmethod from_xml(*e*)
Creates a dictionary from a XML structure in order to create the inherited objects

Parameters **etree_element** – a xml etree

Return type *TextRegion*

Returns a dictionary with keys 'id' and 'coords'

sort_text_lines(*top_to_bottom=True*)
Sorts TextLine from top to bottom according to their mean y coordinate (centroid)

Parameters **top_to_bottom** (bool) – order lines from top to bottom of image, default=True

Return type None

tag = 'TextRegion'

to_dict(*non_serializable_keys=[]*)
Converts a *Region* object to a dictionary.

Parameters **non_serializable_keys** (List[str]) – list of keys that can't be directly serialized and that need some internal serialization

Returns a dictionary with the atributes of the object serialized

to_xml (*name_element='TextRegion'*)
Converts a *Region* object to a xml structure

Parameters **name_element** – name of the object (optional)

Return type Element

Returns a etree structure

dh_segment.io.PAGE.**get_unique_tags_from_xml_text_regions** (*xml_filename*, *tag_pattern='{type:.*;}'*)
Get a list of all the values of labels/tags

Parameters

- **xml_filename** (str) – filename of the xml file
- **tag_pattern** (str) – regular expression pattern to look for in *TextRegion.custom_attribute*

Returns

dh_segment.io.PAGE.**json_serialize** (*dict_to_serialize*, *non_serializable_keys=[]*)
Serialize a dictionary in order to export it.

Parameters

- **dict_to_serialize** (dict) – dictionary to serialize
- **non_serializable_keys** (List[str]) – keys that are not directly seriazable sucha as python objects

Return type dict

Returns the serialized dictionnary

dh_segment.io.PAGE.**parse_file** (*filename*)
Parses the files to create the corresponding Page object. The files can be a .xml or a .json.

Parameters **filename** (str) – file to parse (either json of page xml)

Return type *Page*

Returns Page object containing all the parsed elements

dh_segment.io.PAGE.**save_baselines** (*filename*, *baselines*, *ratio=(1, 1)*, *predictions_shape=None*)

Parameters

- **filename** (str) – filename to save baselines to
- **baselines** – list of baselines
- **ratio** (Tuple[int, int]) – ratio of prediction shape over original shape
- **predictions_shape** (Optional[Tuple[int, int]]) – shape of the masks output by the network

Return type *Page*

Returns

class dh_segment.io.via.**VIAttribute**
A container for VIA attributes.

Parameters

- **name** (*str*) – The name of attribute

- **type** (*str*) – The type of the annotation (dropdown, markbox, . . .)
- **options** (*list*) – The options / labels possible for this attribute.

property name
Alias for field number 0

property options
Alias for field number 2

property type
Alias for field number 1

class dh_segment.io.via.**WorkingItem**
A container for annotated images.

Parameters

- **collection** (*str*) – name of the collection
- **image_name** (*str*) – name of the image
- **original_x** (*int*) – original image x size (width)
- **original_y** (*int*) – original image y size (height)
- **reduced_x** (*int*) – resized x size
- **reduced_y** (*int*) – resized y size
- **iiif** (*str*) – iiif url
- **annotations** (*dict*) – VIA 'region_attributes'

property annotations
Alias for field number 7

property collection
Alias for field number 0

property iiif
Alias for field number 6

property image_name
Alias for field number 1

property original_x
Alias for field number 2

property original_y
Alias for field number 3

property reduced_x
Alias for field number 4

property reduced_y
Alias for field number 5

dh_segment.io.via.**collect_working_items**(*via_annotations*, *collection_name*, *images_dir=None*, *via_version=2*)
Given VIA annotation input, collect all info on *WorkingItem* object. This function will take care of separating images from local files and images from IIIF urls.

Parameters

- **via_annotations** (dict) – via annotations ('regions' field)

- **images_dir** (Optional[str]) – directory where to find the images
- **collection_name** (str) – name of the collection
- **via_version** (int) – version of the VIA tool used to produce the annotations (1 or 2)

Return type List[*WorkingItem*]

Returns list of *WorkingItem*

dh_segment.io.via.**convert_via_region_page_text_region**(*working_item*, *structure_label*)

Parameters

- **working_item** (*WorkingItem*) –
- **structure_label** (str) –

Return type *Page*

Returns

dh_segment.io.via.**create_masks**(*masks_dir*, *working_items*, *via_attributes*, *collection*, *contours_only=False*)

For each annotation, create a corresponding binary mask and resize it (h = 2000). Only valid for VIA 2.0. Several annotations of the same class on the same image produce one image with several masks.

Parameters

- **masks_dir** (str) – where to output the masks
- **working_items** (List[*WorkingItem*]) – infos to work with
- **via_attributes** (List[*VIAttribute*]) – VIAttributes computed by get_via_attributes function.
- **collection** (str) – name of the nollection
- **contours_only** (bool) – creates the binary masks only for the contours of the object (thickness of contours : 20 px)

Return type dict

Returns annotation_summary, a dictionary containing a list of labels per image

dh_segment.io.via.**create_via_annotation_single_image**(*img_filename*, *via_regions*, *file_attributes=None*)

Returns a dictionary item {key: annotation} in VIA format to further export to .json file

Parameters

- **img_filename** (str) – path to the image
- **via_regions** (List[dict]) – regions in VIA format (output from create_via_region_from_coordinates)
- **file_attributes** (Optional[dict]) – file attributes (usually None)

Return type Dict[str, dict]

Returns dictionary item with key and annotations in VIA format

dh_segment.io.via.**create_via_region_from_coordinates**(*coordinates*, *region_attributes*, *type_region*)

Formats coordinates to a VIA region (dict).

Parameters

- **coordinates** (*<built-in function array>*) – (N, 2) coordinates (x, y)
- **region_attributes** (dict) – dictionary with keys : name of labels, values : values of labels
- **type_region** (str) – via region annotation type ('rect', 'polygon')

Return type dict

Returns a region in VIA style (dict/json)

dh_segment.io.via.**export_annotation_dict**(*annotation_dict*, *filename*)
Export the annotations to json file.

Parameters

- **annotation_dict** (dict) – VIA annotations
- **filename** (str) – filename to export the data (json file)

Return type None

Returns

dh_segment.io.via.**get_annotations_per_file**(*via_dict*, *name_file*)
From VIA json content, get annotations relative to the given *name_file*.

Parameters

- **via_dict** (dict) – VIA annotations content (originally json)
- **name_file** (str) – the file to look for (it can be a iiif path or a file path)

Return type dict

Returns dict

dh_segment.io.via.**get_via_attributes**(*annotation_dict*, *via_version=2*)
Gets the attributes of the annotated data and returns a list of *VIAttribute*.

Parameters

- **annotation_dict** (dict) – json content of the VIA exported file
- **via_version** (int) – either 1 or 2 (for VIA v 1.0 or VIA v 2.0)

Return type List[*VIAttribute*]

Returns A list containing VIAttributes

dh_segment.io.via.**load_annotation_data**(*via_data_filename*, *only_img_annotations=False*, *via_version=2*)
Load the content of via annotation files.

Parameters

- **via_data_filename** (str) – via annotations json file
- **only_img_annotations** (bool) – load only the images annotations ('_via_img_metadata' field)
- **via_version** (int) –

Return type dict

Returns the content of json file containing the region annotated

dh_segment.io.via.**parse_via_attributes**(*via_attributes*)
Parses the VIA attribute dictionary and returns a list of VIAttribute instances

Parameters **via_attributes** (dict) – attributes from VIA annotation ('_via_attributes' field)

Return type List[*VIAttribute*]

Returns list of VIAttribute

3.3 Inference

The *dh_segment.inference* module implements the function related to the usage of a dhSegment model, for instance to use a trained model to inference on new data.

3.3.1 Loading a model

LoadedModel(model_base_dir[, predict_mode, ...])	Loads an exported dhSegment model

class dh_segment.inference.**LoadedModel**(*model_base_dir*, *predict_mode='filename'*, *num_parallel_predictions=2*)

Loads an exported dhSegment model

Parameters

- **model_base_dir** – the model directory i.e. containing *saved_model.{pb|pbtxt}*. If not, it is assumed to be a TF exporter directory, and the latest export directory will be automatically selected.
- **predict_mode** – defines the input/output format of the prediction output (see *.predict()*)
- **num_parallel_predictions** – limits the number of conccurent calls of *predict* to avoid Out-Of-Memory issues if predicting on GPU

predict(*input_tensor*, *prediction_key=None*)

Performs the prediction from the loaded model according to the prediction mode.

Prediction modes:

prediction_mode	*input_tensor*	Output prediction dictionnary	Comment
filename	Single filename string	*labels*, *probs*, *original_shape*	Loads the image, resizes it, and predicts
filename_original_shape	Single filename string	*labels*, *probs*	Loads the image, resizes it, predicts and scale the output to the original resolution of the file
image	Single input image [1,H,W,3] float32 (0..255)	*labels*, *probs*, *original_shape*	Resizes the image, and predicts
image_original_shape	Single input image [1,H,W,3] float32 (0..255)	*labels*, *probs*	Resizes the image, predicts, and scale the output to the original resolution of the input
image_resized	Single input image [1,H,W,3] float32 (0..255)	*labels*, *probs*	Predicts from the image input directly

Parameters

- **input_tensor** – a single input whose format should match the prediction mode
- **prediction_key** – if not *None*, will returns the value of the corresponding key of the output dictionnary instead of the full dictionnary

Returns the prediction output

predict_with_tiles(*filename*, *resized_size=None*, *tile_size=500*, *min_overlap=0.2*, *linear_interpolation=True*)

3.4 Post processing

The *dh_segment.post_processing* module contains functions to post-process probability maps.

Binarization

thresholding(probs[, threshold])	Computes the binary mask of the detected Page from the probabilities output by network.
cleaning_binary(mask[, kernel_size])	Uses mathematical morphology to clean and remove small elements from binary images.

Detection

find_boxes(boxes_mask[, mode, min_area, ...])	Finds the coordinates of the box in the binary image *boxes_mask*.
find_polygonal_regions(image_mask[, ...])	Finds the shapes in a binary mask and returns their coordinates as polygons.

Vectorization

find_lines(lines_mask)	Finds the longest central line for each connected component in the given binary mask.

dh_segment.post_processing.**thresholding**(*probs*, *threshold=-1*)

Computes the binary mask of the detected Page from the probabilities output by network.

Parameters

- **probs** (ndarray) – array in range [0, 1] of shape HxWx2
- **threshold** (float) – threshold between [0 and 1], if negative Otsu's adaptive threshold will be used

Return type ndarray

Returns binary mask

dh_segment.post_processing.**cleaning_binary**(*mask*, *kernel_size=5*)

Uses mathematical morphology to clean and remove small elements from binary images.

Parameters

- **mask** (ndarray) – the binary image to clean

- **kernel_size** (int) – size of the kernel

Return type ndarray

Returns the cleaned mask

dh_segment.post_processing.**find_boxes**(*boxes_mask*, *mode='min_rectangle'*, *min_area=0.2*, *p_arc_length=0.01*, *n_max_boxes=inf*)

Finds the coordinates of the box in the binary image *boxes_mask*.

Parameters

- **boxes_mask** (ndarray) – Binary image: the mask of the box to find. uint8, 2D array
- **mode** (str) – 'min_rectangle' : minimum enclosing rectangle, can be rotated 'rectangle' : minimum enclosing rectangle, not rotated 'quadrilateral' : minimum polygon approximated by a quadrilateral
- **min_area** (float) – minimum area of the box to be found. A value in percentage of the total area of the image.
- **p_arc_length** (float) – used to compute the epsilon value to approximate the polygon with a quadrilateral. Only used when 'quadrilateral' mode is chosen.
- **n_max_boxes** – maximum number of boxes that can be found (default inf). This will select n_max_boxes with largest area.

Return type list

Returns list of length n_max_boxes containing boxes with 4 corners [[x1,y1], . . . , [x4,y4]]

dh_segment.post_processing.**find_polygonal_regions**(*image_mask*, *min_area=0.1*, *n_max_polygons=inf*)

Finds the shapes in a binary mask and returns their coordinates as polygons.

Parameters

- **image_mask** (ndarray) – Uint8 binary 2D array
- **min_area** (float) – minimum area the polygon should have in order to be considered as valid (value within [0,1] representing a percent of the total size of the image)
- **n_max_polygons** (int) – maximum number of boxes that can be found (default inf). This will select n_max_boxes with largest area.

Return type list

Returns list of length n_max_polygons containing polygon's n coordinates [[x1, y1], . . . [xn, yn]]

dh_segment.post_processing.**find_lines**(*lines_mask*)

Finds the longest central line for each connected component in the given binary mask.

Parameters **lines_mask** (ndarray) – Binary mask of the detected line-areas

Return type list

Returns a list of Opencv-style polygonal lines (each contour encoded as [N,1,2] elements where each tuple is (x,y))

3.5 Utilities

The *dh_segment.utils* module contains the parameters for config with sacred package, image label vizualization functions and miscelleanous helpers.

3.5.1 Parameters

ModelParams(**kwargs)	Parameters related to the model
TrainingParams(**kwargs)	Parameters to configure training process

3.5.2 Label image helpers

label_image_to_class(label_image, classes_file) **rtype** tensorflow.Tensor	
class_to_label_image(class_label, classes_file) **rtype** tensorflow.Tensor	
multilabel_image_to_class(label_image, ...)	Combines image annotations with classes info of the txt file to create the input label for the training.
multiclass_to_label_image(...) **rtype** tensorflow.Tensor	
get_classes_color_from_file(classes_file) **rtype** ndarray	
get_n_classes_from_file(classes_file) **rtype** int	
get_classes_color_from_file_multilabel(...)	Get classes and code labels from txt file.
get_n_classes_from_file_multilabel(classes_file) **rtype** int	

3.5.3 Evaluation utils

Metrics()	
intersection_over_union(cnt1, cnt2, shape_mask)	

3.5.4 Miscellaneous helpers

parse_json(filename)	
dump_json(filename, dict)	
load_pickle(filename)	
dump_pickle(filename, obj)	
hash_dict(params)	

class dh_segment.utils.**PredictionType**

Variables

- ***CLASSIFICATION*** –

- *REGRESSION* –
- *MULTILABEL* –

CLASSIFICATION = 'CLASSIFICATION'

MULTILABEL = 'MULTILABEL'

REGRESSION = 'REGRESSION'

classmethod parse(*prediction_type*)

class dh_segment.utils.**VGG16ModelParams**

CORRECTED_VERSION = None

INTERMEDIATE_CONV = [[(256, 3)]]

PRETRAINED_MODEL_FILE = 'pretrained_models/vgg_16.ckpt'

SELECTED_LAYERS_UPSCALING = [True, True, True, True, False, False]

UPSCALE_PARAMS = [[(32, 3)], [(64, 3)], [(128, 3)], [(256, 3)], [(512, 3)], [(512, 3)]

class dh_segment.utils.**ResNetModelParams**

CORRECT_VERSION = False

INTERMEDIATE_CONV = None

PRETRAINED_MODEL_FILE = 'pretrained_models/resnet_v1_50.ckpt'

SELECTED_LAYERS_UPSCALING = [True, True, True, True, True]

UPSCALE_PARAMS = [(32, 0), (64, 0), (128, 0), (256, 0), (512, 0)]

class dh_segment.utils.**UNetModelParams**

CORRECT_VERSION = False

INTERMEDIATE_CONV = None

PRETRAINED_MODEL_FILE = None

SELECTED_LAYERS_UPSCALING = None

UPSCALE_PARAMS = None

class dh_segment.utils.**ModelParams**(***kwargs*)

Parameters related to the model

check_params()

class dh_segment.utils.**TrainingParams**(***kwargs*)

Parameters to configure training process

Variables

- **n_epochs** (*int*) – number of epoch for training
- **evaluate_every_epoch** (*int*) – the model will be evaluated every *n* epochs
- **learning_rate** (*float*) – the starting learning rate value
- **exponential_learning** (*bool*) – option to use exponential learning rate
- **batch_size** (*int*) – size of batch

- **data_augmentation** (*bool*) – option to use data augmentation (by default is set to False)
- **data_augmentation_flip_lr** (*bool*) – option to use image flipping in right-left direction
- **data_augmentation_flip_ud** (*bool*) – option to use image flipping in up down direction
- **data_augmentation_color** (*bool*) – option to use data augmentation with color
- **data_augmentation_max_rotation** (*float*) – maximum angle of rotation (in radians) for data augmentation
- **data_augmentation_max_scaling** (*float*) – maximum scale of zooming during data augmentation (range: [0,1])
- **make_patches** (*bool*) – option to crop image into patches. This will cut the entire image in several patches
- **patch_shape** (*tuple*) – shape of the patches
- **input_resized_size** (*int*) – size (in pixel) of the image after resizing. The original ratio is kept. If no resizing is wanted, set it to -1
- **weights_labels** (*list*) – weight given to each label. Should be a list of length = number of classes
- **training_margin** (*int*) – size of the margin to add to the images. This is particularly useful when training with patches
- **local_entropy_ratio** (*float*) –
- **local_entropy_sigma** (*float*) –
- **focal_loss_gamma** (*float*) – value of gamma for the focal loss. See paper : https://arxiv.org/abs/1708.02002

check_params()

Checks if there is no parameter inconsistency

Return type None

dh_segment.utils.**label_image_to_class**(*label_image*, *classes_file*)

Return type tensorflow.Tensor

dh_segment.utils.**class_to_label_image**(*class_label*, *classes_file*)

Return type tensorflow.Tensor

dh_segment.utils.**multilabel_image_to_class**(*label_image*, *classes_file*)

Combines image annotations with classes info of the txt file to create the input label for the training.

Parameters

- **label_image** (*tensorflow.Tensor*) – annotated image [H,W,Ch] or [B,H,W,Ch] (Ch = color channels)
- **classes_file** (*str*) – the filename of the txt file containing the class info

Return type tensorflow.Tensor

Returns [H,W,Cl] or [B,H,W,Cl] (Cl = number of classes)

dh_segment.utils.**multiclass_to_label_image**(*class_label_tensor*, *classes_file*)

Return type tensorflow.Tensor

dh_segment.utils.**get_classes_color_from_file**(*classes_file*)

Return type ndarray

dh_segment.utils.**get_n_classes_from_file**(*classes_file*)

Return type int

dh_segment.utils.**get_classes_color_from_file_multilabel**(*classes_file*)

Get classes and code labels from txt file. This function deals with the case of elements with multiple labels.

Parameters **classes_file** (str) – file containing the classes (usually named *classes.txt*)

Return type Tuple[ndarray, <built-in function array>]

Returns for each class the RGB color (array size [N, 3]); and the label's code (array size [N, C]), with N the number of combinations and C the number of classes

dh_segment.utils.**get_n_classes_from_file_multilabel**(*classes_file*)

Return type int

dh_segment.utils.**parse_json**(*filename*)

dh_segment.utils.**dump_json**(*filename*, *dict*)

dh_segment.utils.**load_pickle**(*filename*)

dh_segment.utils.**dump_pickle**(*filename*, *obj*)

dh_segment.utils.**hash_dict**(*params*)

class dh_segment.utils.**Metrics**

compute_accuracy()

compute_iu()

compute_miou()

compute_mse()

compute_prf(*beta=1*)

compute_psnr()

save_to_json(*json_filename*)

Return type None

dh_segment.utils.**intersection_over_union**(*cnt1*, *cnt2*, *shape_mask*)

CHAPTER

FOUR

REFERENCES

CHAPTER

FIVE

CHANGELOG

5.1 0.5.0 - 2019-08-14

5.1.1 Added

- `https` can now be used for PAGEXML schema.
- All the `PAGE` objects can now have `custom_attribute` (this is mainly for tagging purposes).

5.1.2 Changed

- The `exps` folders contains now only two examples that can be used as demos. The other experiments have been removed.
- Installation of `dh_segment` package is now done via pip (using `setup.py`) except for `tensorflow` package which is installed with anaconda.
- `setup.py` has more flexible package versions.
- Forced integer conversion when exporting coordinates to XML format.

5.1.3 Fixed

- In page `demo.py` a empty `Border` is now created if no region has been detected.

5.1.4 Removed

- Experiments in `exps` folder have been removed, except for `page` and `cbad`.

5.2 0.4.0 - 2019-04-10

5.2.1 Added

- Input data can be a .csv file with format `<filename-image>,<filename-label>`.
- `dh_segment.io.via` helper functions to generate/export groundtruth from/to VGG Image Annotation tool.
- `Point.array_to_point` to export a `np.array` into a list of `Point`.
- PAGEXML Regions can now contain a custom attribute (Transkribus output of region annotation)

- `Page.to_json()` method for json formatting.

5.2.2 Changed

- `tensorflow` v1.13 and `opencv` v4.0 are now used.
- mIOU metric for evaluation during training (instead of accuracy).
- TextLines are sorted according to their mean *y* coordinate when exported.

5.2.3 Fixed

- Variable names typos in `input.py` and `train.py`.
- Documentation of the quickstart demo.

5.2.4 Removed

dhSegment is a tool for Historical Document Processing. Its generic approach allows to segment regions and extract content from different type of documents. See some example of applications in the *Use cases* section.

The complete description of the system can be found in the corresponding paper [AOSK18] .

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

6.1 Acknowledgement

This work has been partly funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943.

BIBLIOGRAPHY

[AOSK18] Sofia Ares Oliveira, Benoit Seguin, and Frederic Kaplan. Dhsegment: a generic deep-learning approach for document segmentation. In *Frontiers in Handwriting Recognition (ICFHR), 2018 16th International Conference on*, 7–12. IEEE, 2018.

[GruningLD+18] Tobias Grüning, Roger Labahn, Markus Diem, Florian Kleber, and Stefan Fiel. Read-bad: a new dataset and evaluation scheme for baseline detection in archival documents. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 351–356. IEEE, 2018.

[SSE+16] Foteini Simistira, Mathias Seuret, Nicole Eichenberger, Angelika Garz, Marcus Liwicki, and Rolf Ingold. Diva-hisdb: a precisely annotated large dataset of challenging medieval manuscripts. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, 471–476. IEEE, 2016.

[TDW+17] Chris Tensmeyer, Brian Davis, Curtis Wigington, Iain Lee, and Bill Barrett. Pagenet: page boundary extraction in historical handwritten documents. In *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*, 59–64. ACM, 2017.

PYTHON MODULE INDEX

d

dh_segment, 13
dh_segment.inference, 32
dh_segment.io, 13
dh_segment.io.PAGE, 17
dh_segment.io.via, 28
dh_segment.network, 13
dh_segment.post_processing, 33
dh_segment.utils, 34

INDEX

A

annotations() (*dh_segment.io.via.WorkingItem property*), 29
array_to_list() (*dh_segment.io.PAGE.Point class method*), 22
array_to_point() (*dh_segment.io.PAGE.Point class method*), 23

B

BaseElement (*class in dh_segment.io.PAGE*), 17
Border (*class in dh_segment.io.PAGE*), 17

C

check_params() (*dh_segment.utils.ModelParams method*), 36
check_params() (*dh_segment.utils.TrainingParams method*), 37
check_tag() (*dh_segment.io.PAGE.BaseElement class method*), 17
class_to_label_image() (*in module dh_segment.utils*), 37
CLASSIFICATION (*dh_segment.utils.PredictionType attribute*), 36
cleaning_binary() (*in module dh_segment.post_processing*), 33
collect_working_items() (*in module dh_segment.io.via*), 29
collection() (*dh_segment.io.via.WorkingItem property*), 29
compute_accuracy() (*dh_segment.utils.Metrics method*), 38
compute_iu() (*dh_segment.utils.Metrics method*), 38
compute_miou() (*dh_segment.utils.Metrics method*), 38
compute_mse() (*dh_segment.utils.Metrics method*), 38
compute_prf() (*dh_segment.utils.Metrics method*), 38
compute_psnr() (*dh_segment.utils.Metrics method*), 38
convert_via_region_page_text_region() (*in module dh_segment.io.via*), 30
CORRECT_VERSION (*dh_segment.utils.ResNetModelParams attribute*), 36
CORRECT_VERSION (*dh_segment.utils.UNetModelParams attribute*), 36
CORRECTED_VERSION (*dh_segment.utils.VGG16ModelParams attribute*), 36
create_masks() (*in module dh_segment.io.via*), 30
create_via_annotation_single_image() (*in module dh_segment.io.via*), 30
create_via_region_from_coordinates() (*in module dh_segment.io.via*), 30
cv2_to_point_list() (*dh_segment.io.PAGE.Point class method*), 23

D

data_augmentation_fn() (*in module dh_segment.io*), 16
dh_segment (*module*), 13
dh_segment.inference (*module*), 32
dh_segment.io (*module*), 13
dh_segment.io.PAGE (*module*), 17
dh_segment.io.via (*module*), 28
dh_segment.network (*module*), 13
dh_segment.post_processing (*module*), 33
dh_segment.utils (*module*), 34
draw_baselines() (*dh_segment.io.PAGE.Page method*), 19
draw_column_groups() (*dh_segment.io.PAGE.Page method*), 20
draw_graphic_regions() (*dh_segment.io.PAGE.Page method*), 20
draw_line_groups() (*dh_segment.io.PAGE.Page method*), 20
draw_lines() (*dh_segment.io.PAGE.Page method*), 20
draw_page_border() (*dh_segment.io.PAGE.Page method*), 21
draw_separator_lines() (*dh_segment.io.PAGE.Page method*), 21
draw_text() (*dh_segment.io.PAGE.Page method*), 21
draw_text_regions() (*dh_segment.io.PAGE.Page*

method), 22
dump_json() (*in module dh_segment.utils*), 38
dump_pickle() (*in module dh_segment.utils*), 38

E

export_annotation_dict() (*in module dh_segment.io.via*), 31
extract_patches_fn() (*in module dh_segment.io*), 17

F

find_boxes() (*in module dh_segment.post_processing*), 34
find_lines() (*in module dh_segment.post_processing*), 34
find_polygonal_regions() (*in module dh_segment.post_processing*), 34
from_array() (*dh_segment.io.PAGE.TextLine class method*), 26
from_dict() (*dh_segment.io.PAGE.Border class method*), 17
from_dict() (*dh_segment.io.PAGE.GraphicRegion class method*), 18
from_dict() (*dh_segment.io.PAGE.GroupSegment class method*), 18
from_dict() (*dh_segment.io.PAGE.Metadata class method*), 19
from_dict() (*dh_segment.io.PAGE.Page class method*), 22
from_dict() (*dh_segment.io.PAGE.Region class method*), 24
from_dict() (*dh_segment.io.PAGE.SeparatorRegion class method*), 24
from_dict() (*dh_segment.io.PAGE.TableRegion class method*), 25
from_dict() (*dh_segment.io.PAGE.TextLine class method*), 26
from_dict() (*dh_segment.io.PAGE.TextRegion class method*), 27
from_xml() (*dh_segment.io.PAGE.Border class method*), 17
from_xml() (*dh_segment.io.PAGE.GraphicRegion class method*), 18
from_xml() (*dh_segment.io.PAGE.Metadata class method*), 19
from_xml() (*dh_segment.io.PAGE.Page class method*), 22
from_xml() (*dh_segment.io.PAGE.Region class method*), 24
from_xml() (*dh_segment.io.PAGE.SeparatorRegion class method*), 25
from_xml() (*dh_segment.io.PAGE.TableRegion class method*), 25
from_xml() (*dh_segment.io.PAGE.TextLine class method*), 26
from_xml() (*dh_segment.io.PAGE.TextRegion class method*), 27
full_tag() (*dh_segment.io.PAGE.BaseElement class method*), 17

G

get_annotations_per_file() (*in module dh_segment.io.via*), 31
get_classes_color_from_file() (*in module dh_segment.utils*), 38
get_classes_color_from_file_multilabel() (*in module dh_segment.utils*), 38
get_n_classes_from_file() (*in module dh_segment.utils*), 38
get_n_classes_from_file_multilabel() (*in module dh_segment.utils*), 38
get_unique_tags_from_xml_text_regions() (*in module dh_segment.io.PAGE*), 28
get_via_attributes() (*in module dh_segment.io.via*), 31
GraphicRegion (*class in dh_segment.io.PAGE*), 18
GroupSegment (*class in dh_segment.io.PAGE*), 18

H

hash_dict() (*in module dh_segment.utils*), 38

I

iiif() (*dh_segment.io.via.WorkingItem property*), 29
image_name() (*dh_segment.io.via.WorkingItem property*), 29
inference_resnet_v1_50() (*in module dh_segment.network*), 13
inference_u_net() (*in module dh_segment.network*), 13
inference_vgg16() (*in module dh_segment.network*), 13
input_fn() (*in module dh_segment.io*), 15
INTERMEDIATE_CONV (*dh_segment.utils.ResNetModelParams attribute*), 36
INTERMEDIATE_CONV (*dh_segment.utils.UNetModelParams attribute*), 36
INTERMEDIATE_CONV (*dh_segment.utils.VGG16ModelParams attribute*), 36
intersection_over_union() (*in module dh_segment.utils*), 38

J

json_serialize() (*in module dh_segment.io.PAGE*), 28

L

label_image_to_class() (*in module dh_segment.utils*), 37
list_from_xml() (*dh_segment.io.PAGE.Point class method*), 23
list_point_to_string() (*dh_segment.io.PAGE.Point class method*), 23
list_to_cv2poly() (*dh_segment.io.PAGE.Point class method*), 23
list_to_point() (*dh_segment.io.PAGE.Point class method*), 23
load_and_resize_image() (*in module dh_segment.io*), 16
load_annotation_data() (*in module dh_segment.io.via*), 31
load_pickle() (*in module dh_segment.utils*), 38
LoadedModel (*class in dh_segment.inference*), 32
local_entropy() (*in module dh_segment.io*), 17

M

Metadata (*class in dh_segment.io.PAGE*), 19
Metrics (*class in dh_segment.utils*), 38
ModelParams (*class in dh_segment.utils*), 36
multiclass_to_label_image() (*in module dh_segment.utils*), 37
MULTILABEL (*dh_segment.utils.PredictionType attribute*), 36
multilabel_image_to_class() (*in module dh_segment.utils*), 37

N

name() (*dh_segment.io.via.VIAttribute property*), 29

O

options() (*dh_segment.io.via.VIAttribute property*), 29
original_x() (*dh_segment.io.via.WorkingItem property*), 29
original_y() (*dh_segment.io.via.WorkingItem property*), 29

P

Page (*class in dh_segment.io.PAGE*), 19
parse() (*dh_segment.utils.PredictionType class method*), 36
parse_file() (*in module dh_segment.io.PAGE*), 28
parse_json() (*in module dh_segment.utils*), 38
parse_via_attributes() (*in module dh_segment.io.via*), 31
Point (*class in dh_segment.io.PAGE*), 22
point_to_list() (*dh_segment.io.PAGE.Point class method*), 23
predict() (*dh_segment.inference.LoadedModel method*), 32
predict_with_tiles() (*dh_segment.inference.LoadedModel method*), 33
PredictionType (*class in dh_segment.utils*), 35
PRETRAINED_MODEL_FILE (*dh_segment.utils.ResNetModelParams attribute*), 36
PRETRAINED_MODEL_FILE (*dh_segment.utils.UNetModelParams attribute*), 36
PRETRAINED_MODEL_FILE (*dh_segment.utils.VGG16ModelParams attribute*), 36

R

reduced_x() (*dh_segment.io.via.WorkingItem property*), 29
reduced_y() (*dh_segment.io.via.WorkingItem property*), 29
Region (*class in dh_segment.io.PAGE*), 24
REGRESSION (*dh_segment.utils.PredictionType attribute*), 36
resize_image() (*in module dh_segment.io*), 16
resnet_v1_50_fn() (*in module dh_segment.network*), 13
ResNetModelParams (*class in dh_segment.utils*), 36
rotate_crop() (*in module dh_segment.io*), 16

S

save_baselines() (*in module dh_segment.io.PAGE*), 28
save_to_json() (*dh_segment.utils.Metrics method*), 38
scale_baseline_points() (*dh_segment.io.PAGE.TextLine method*), 26
SELECTED_LAYERS_UPSCALING (*dh_segment.utils.ResNetModelParams attribute*), 36
SELECTED_LAYERS_UPSCALING (*dh_segment.utils.UNetModelParams attribute*), 36
SELECTED_LAYERS_UPSCALING (*dh_segment.utils.VGG16ModelParams attribute*), 36
SeparatorRegion (*class in dh_segment.io.PAGE*), 24
serving_input_filename() (*in module dh_segment.io*), 16
serving_input_image() (*in module dh_segment.io*), 16
sort_text_lines() (*dh_segment.io.PAGE.TextRegion method*),

27

T

TableRegion (*class in dh_segment.io.PAGE*), 25
tag (*dh_segment.io.PAGE.BaseElement attribute*), 17
tag (*dh_segment.io.PAGE.Border attribute*), 18
tag (*dh_segment.io.PAGE.GraphicRegion attribute*), 18
tag (*dh_segment.io.PAGE.Metadata attribute*), 19
tag (*dh_segment.io.PAGE.Page attribute*), 22
tag (*dh_segment.io.PAGE.Region attribute*), 24
tag (*dh_segment.io.PAGE.SeparatorRegion attribute*), 25
tag (*dh_segment.io.PAGE.TableRegion attribute*), 25
tag (*dh_segment.io.PAGE.TextLine attribute*), 26
tag (*dh_segment.io.PAGE.TextRegion attribute*), 27
Text (*class in dh_segment.io.PAGE*), 26
TextLine (*class in dh_segment.io.PAGE*), 26
TextRegion (*class in dh_segment.io.PAGE*), 27
thresholding() (*in module dh_segment.post_processing*), 33
to_dict() (*dh_segment.io.PAGE.Border method*), 18
to_dict() (*dh_segment.io.PAGE.Metadata method*), 19
to_dict() (*dh_segment.io.PAGE.Point method*), 24
to_dict() (*dh_segment.io.PAGE.Region method*), 24
to_dict() (*dh_segment.io.PAGE.Text method*), 26
to_dict() (*dh_segment.io.PAGE.TextLine method*), 26
to_dict() (*dh_segment.io.PAGE.TextRegion method*), 27
to_json() (*dh_segment.io.PAGE.Page method*), 22
to_xml() (*dh_segment.io.PAGE.Border method*), 18
to_xml() (*dh_segment.io.PAGE.GraphicRegion method*), 18
to_xml() (*dh_segment.io.PAGE.Metadata method*), 19
to_xml() (*dh_segment.io.PAGE.Page method*), 22
to_xml() (*dh_segment.io.PAGE.Region method*), 24
to_xml() (*dh_segment.io.PAGE.SeparatorRegion method*), 25
to_xml() (*dh_segment.io.PAGE.TableRegion method*), 25
to_xml() (*dh_segment.io.PAGE.TextLine method*), 27
to_xml() (*dh_segment.io.PAGE.TextRegion method*), 27
TrainingParams (*class in dh_segment.utils*), 36
type() (*dh_segment.io.via.VIAttribute property*), 29

U

UNetModelParams (*class in dh_segment.utils*), 36
UPSCALE_PARAMS (*dh_segment.utils.ResNetModelParams attribute*), 36
UPSCALE_PARAMS (*dh_segment.utils.UNetModelParams attribute*), 36
UPSCALE_PARAMS (*dh_segment.utils.VGG16ModelParams attribute*), 36

V

VGG16ModelParams (*class in dh_segment.utils*), 36
vgg_16_fn() (*in module dh_segment.network*), 13
VIAttribute (*class in dh_segment.io.via*), 28

W

WorkingItem (*class in dh_segment.io.via*), 29
write_to_file() (*dh_segment.io.PAGE.Page method*), 22