Calamari OCR

The Calamari OCR authors

Aug 18, 2023

DOCUMENTATION

CHAPTER

ONE

INSTALL

Calamari requires:

- Python3.7 or later including the development packages.
- (optional) cuda/cudnn libs for GPU support, see tensorflow for the versions which are required/compatible.

Calamari was tested on Linux, but should be flawlessly usable on windows or mac.

See also the install instructions for tfaip and tensorflow.

1.1 Installation using Pip

The suggested method is to install calamari into a virtual environment using pip:

```
virtualenv -p python3 PATH_TO_VENV_DIR # (e.g. virtualenv -p python3 calamari_venv)
source PATH_TO_VENV_DIR/bin/activate
pip install calamari-ocr
```

which will install Calamari and all of its dependencies.

To install the package without a virtual environment simply run

1.2 Installation from Source

To install the package from its source, download the source code and install it. Optionally (but recommended) install in a virtual env.

```
git clone https://github.com/calamari-OCR/calamari
cd calamari
python setup.py install
```

Conda users can alternatively call

```
conda env create -f environment_master.yml
```

1.3 Development Setup

Calamari uses black code styling. It is recommended to integrate running black as pre-commit hook. The following guide helps to setup everything.

The requirements `pre-commit` and `black` are already part of the requirements. Setup `pre-commit` by calling:

```
pre-commit install
```

To upgrade the pre-commit packages call

```
pre-commit autoupdate
```

CHAPTER

TWO

COMMAND-LINE USAGE

The easiest way to use Calamari is the command-line interface. It allows to apply existent models on text lines but also to train new models.

2.1 calamari-predict

The `calamari-predict` scripts allows to transcribe an image of a text line into the written text. Note, that currently only OCR on lines is supported. Segmenting pages into lines (and the preceding preprocessing steps), other software solutions as provided by OCRopus, Kraken, Tesseract, etc, are required. For users (especially less technical ones) in need of an all-in-one package OCR4all might be worth a look.

2.1.1 Example Usage

The script at least requires the model (`path_to_model.ckpt`) to apply and the paths to the images (`your_images.*.png`) to transcribe:

```
calamari-predict --checkpoint path_to_model.ckpt --data.images your_images.*.png
```

2.1.2 Voting

Calamari supports confidence voting to different predictions of different models. To enable voting simply pass several models to the `--checkpoint` argument:

```
calamari-predict --checkpoint path_to_model_1.ckpt path_to_model_2.ckpt ... --data.
↪images your_images.*.png
```

2.1.3 Parameters

In the following is the full list of arguments (`calamari-predict --help`):

```
usage: calamari-predict [--version] [--show] [-h]

optional arguments:
  --version             show program's version number and exit
  --show                show the parsed parameters
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  --checkpoint [CHECKPOINT [CHECKPOINT ...]]
                        Path to the checkpoint without file extension (default: None)
  --data DATA
  --verbose VERBOSE     Print the prediction result to the log (default: True)
  --extended_prediction_data EXTENDED_PREDICTION_DATA
                        Write: Predicted string, labels; position, probabilities and␣
↪alternatives of chars to a .pred file (default: False)
  --extended_prediction_data_format EXTENDED_PREDICTION_DATA_FORMAT
                        Extension format: Either pred or json. Note that json will not␣
↪print logits. (default: json)
  --no_progress_bars NO_PROGRESS_BARS
                        Do not show any progress bars (default: False)
  --voter VOTER
  --output_dir OUTPUT_DIR
                        By default the prediction files will be written to the same␣
↪directory as the given files. You can use this argument to specify a specific output␣
↪dir for the prediction files. (default: None)
  --pipeline.batch_size PIPELINE.BATCH_SIZE
                        Batch size (default: 16)
  --pipeline.limit PIPELINE.LIMIT
                        Limit the number of examples produced by the generator. Note, if␣
↪GeneratingDataProcessors are present in the data pipeline, the number of examples␣
↪produced by the generator can differ. (default: -1)
  --pipeline.prefetch PIPELINE.PREFETCH
                        Prefetching data. -1 default to max(num_processes * 2 by default,
↪ 2 * batch size) (default: -1)
  --pipeline.num_processes PIPELINE.NUM_PROCESSES
                        Number of processes for data loading. (default: 4)
  --pipeline.batch_drop_remainder PIPELINE.BATCH_DROP_REMAINDER
                        Drop remainder parameter of padded_batch. Drop batch if it is␣
↪smaller than batch size. (default: False)
  --pipeline.shuffle_buffer_size PIPELINE.SHUFFLE_BUFFER_SIZE
                        Size of the shuffle buffer required for randomizing data (if␣
↪required). Disabled by default. (default: -1)
  --pipeline.bucket_boundaries [PIPELINE.BUCKET_BOUNDARIES [PIPELINE.BUCKET_BOUNDARIES ..
↪.]]
                        Elements of the Dataset are grouped together by length and then␣
↪are padded and batched. See tf.data.experimental.bucket_by_sequence_length (default:␣
↪[])
  --pipeline.bucket_batch_sizes [PIPELINE.BUCKET_BATCH_SIZES [PIPELINE.BUCKET_BATCH_
↪SIZES ...]]
                        Batch sizes of the buckets. By default, batch_size * (len(bucked_
↪boundaries) + 1). (default: None)
  --predictor.progress_bar PREDICTOR.PROGRESS_BAR
                        Render a progress bar during prediction. (default: True)
  --predictor.run_eagerly PREDICTOR.RUN_EAGERLY
                        Run the prediction model in eager mode. Use for debug only.␣
↪(default: False)
  --data.skip_invalid DATA.SKIP_INVALID
                        Missing help string (default: False)
  --data.non_existing_as_empty DATA.NON_EXISTING_AS_EMPTY
```

(continues on next page)

(continued from previous page)

```
                        Missing help string (default: False)
  --data.preload DATA.PRELOAD
                        Instead of preloading all data, load the data on the fly. This␣
↪is slower, but might be required for limited RAM or large dataset (default: True)
  --data.images [DATA.IMAGES [DATA.IMAGES ...]]
                        List all image files that shall be processed. Ground truth files␣
↪with the same base name but with '.gt.txt' as extension are required at the same␣
↪location (default: [])
  --data.texts [DATA.TEXTS [DATA.TEXTS ...]]
                        List the text files (default: [])
  --data.gt_extension DATA.GT_EXTENSION
                        Extension of the gt files (expected to exist in same dir)␣
↪(default: .gt.txt)
  --data.pred_extension DATA.PRED_EXTENSION
                        Extension of prediction text files (default: .pred.txt)
  --voter.type {SequenceVoter,ConfidenceVoterDefaultCTC,sequence_voter,confidence_voter_
↪default_ctc}
                        Missing help string (default: VoterType.
↪ConfidenceVoterDefaultCTC)
  --voter.blank_index VOTER.BLANK_INDEX
                        Missing help string (default: 0)
  --predictor.device.gpus [PREDICTOR.DEVICE.GPUS [PREDICTOR.DEVICE.GPUS ...]]
                        List of the GPUs to use. (default: None)
  --predictor.device.gpu_auto_tune PREDICTOR.DEVICE.GPU_AUTO_TUNE
                        Enable auto tuning of the GPUs (default: False)
  --predictor.device.gpu_memory PREDICTOR.DEVICE.GPU_MEMORY
                        Limit the per GPU memory in MB. By default the memory will grow␣
↪automatically (default: None)
  --predictor.device.soft_device_placement PREDICTOR.DEVICE.SOFT_DEVICE_PLACEMENT
                        Set up soft device placement is enabled (default: True)
  --predictor.device.dist_strategy {DEFAULT,CENTRAL_STORAGE,MIRROR,default,central_
↪storage,mirror}
                        Distribution strategy for multi GPU, select 'mirror' or 'central_
↪storage' (default: DistributionStrategy.DEFAULT)
```

2.2 calamari-train

Calamari allows to train new models using the `calamari-train`-script which produces a single model.

2.2.1 Selected Parameters

The following list highlights the most common parameters to adapt training. A full list is shown below.

- `--trainer.output_dir`: A path where to store checkpoints
- `--trainer.epochs`: The maximum number of training iterations (batches) for training. Note: this is the upper boundary if you use early stopping.
- `--trainer.samples_per_epoch`: The number of samples to process per epoch (by default the size of the dataset)
- `--early_stopping.frequency=1`: How often to check for early stopping on the validation dataset.

- `--early_stopping.n_to_go=5`: How many successive models must be worse than the current best model to break the training loop
- `--warmstart.model`: Load network weights from a given pretrained model. Note that the codec will probably change its size to match the codec of the provided ground truth files. To enforce that some characters may not be deleted use a –whitelist.
- `--n_augmentations=0`: Data augmentation on the training set.

2.2.2 Validation

Calamari requires to pass validation data to detect and store the best model during training. There are multiple ways to define the validation data. The type of validation is defined by the `--trainer.gen` param which defaults to `TrainVal`, a separate train and validations set.

Separate Train and Validation Set

The default is to define separate training and validation lists, e.g.:

```
calamari-train --train.images TRAIN/FILES.png --val.images VAL/FILES.png
```

Automatic Split of Files into Train and Val

Alternatively, Calamari allows to split provided data automatically into train and val by a given ratio. The following example splits the provided `train.images` into the actual training data (80%) and validation data (20%).

```
calamari-train --trainer.gen SplitTrain --trainer.gen.validation_split_ratio=0.2 --train.
↪images TRAIN/FILES.png
```

Use Training Data also for Validation

Finally, Calamari can determine the best model also on the training data which will most likely result into a highly overfit model, though.

```
calamari-train --trainer.gen TrainOnly --train.images TRAIN/FILES.png
```

2.2.3 Data Origin

The type of data to process is adapted by specifying the type of `--train` and `--val`, see also *here*, e.g. use

```
calamari-train --train PageXML --train.images TRAIN/FILES.png --val PageXML --val.images␣
↪VAL/FILES.png
```

to train and validate on PageXML files.

2.2.4 Training Duration

The training duration is adapted by the `--trainer.epochs`, and `-early_stopping` parameters.

2.2.5 Data Augmentation

Calamari supports automatic data augmentation. The ratio of real and augmented data can be adapted by the `--n_augmentations` parameter. `--n_augmentations=5` means that for every real line there are five augmented lines.

2.2.6 Color mode

By default, Calamari converts all images to grayscale. Any color (or RGBA) image is converted to grayscale using OpenCVs convert function. This can be changed to a simple average on RGB by setting `--train.to_gray_method=avg`.

To train a model on color images, if present in the images, set `--data.input_channels=3`.

2.2.7 Warm-Starting with a Pretrained model

Provide a path to `--warmstart.model` to preload from this model. Loading a model modify the codec of the model by keeping the weights of known characters that are present in the loaded model and the target alphabet. By default characters that are not present in the new alphabet are erased, this can be adapted by setting `--codec.keep_loaded True` which will produce the union of both alphabets (loaded and target).

By default all weights of the loaded model can be trained, i.e. receive weight updates. To disable the behaviour specify `--trainer.no_train_scope` which expects a regular expression to match with the layers to not train.

2.2.8 Preloading Data / Load Data on the fly

Calamari allows both to load the complete data into the RAM before training which can considerably speed up training. Large datasets can not be stored completely into RAM, tough, which is why the data can also be loaded on the fly. The default is to preload the data into the RAM, modify by

```
calamari-train --train.preload False --val.preload False
```

2.2.9 Training with GPU

By default, Calamari does not use GPUs for training even if present. To enable training on a GPU pass the GPU device id to `--device.gpus`, e.g.:

```
calamari-train --device.gpus 0
```

which will use *GPU0* for training. This is the parameter if only one GPU is present in a system.

2.2.10 Network Architecture

Calamari allows to fully modify the network architecture in two exclusive ways. Following layers are supported: `BiLSTM`, `Concat`, `Conv2D`, `DilatedBlock`, `Dropout`, `Pool2D`, `TransposedConv2D`.

Predefined

Calamari provides several predefined network architectures, that can be passed to the *–network* argument.

- `def`: The default Calamari network with conv, max-pool, conv, max-pool, and one BiLSTM layer.
- `deep3`: The default Calamari network with conv, max-pool, conv, max-pool, conv, and three BiLSTM layer.
- `htr+`: The default network architecture of Transkribus (see, e.g., Michael et al. (2019)). Note that this network should/must be applied on a larger line height (recommended is 64: `--data.line_height=64`)

Simple

The easiest way to modify the network architecture is to pass the `--network` argument. The default architecture of Calamari can be expressed as

```
calamari-train --network=conv=40:3x3,pool=2x2,conv=60:3x3,pool=2x2,lstm=200,dropout=0.5
```

which are two convolution and max-pooling blocks, followed by an bidirectional lstm and dropout.

The following adds an additional lstm layer with only 100 nodes and also a dropout of 0.5.

```
calamari-train --network=conv=40:3x3,pool=2x2,conv=60:3x3,pool=2x2,lstm=200,dropout=0.5,
↪lstm=100,dropout=0.5
```

Advanced

The advanced setup allows to modify more parameters of the network architecture, e.g. including the activation functions.

```
calamari-train \
    # Define the overall structure
    --model.layers Conv Pool conv Pool BiLSTM Dropout \
    # Set the parameters of the first layer (Conv)
    --model.layers.0.filters 40 \
    --model.layers.0.stride 3 3 \
    # Set the parameters of the next layer (Pool)
    --model.layers.1.pool_size 2 2 \
    # Set the parameters of the next layer (Conv)
    --model.layers.2.filters 60 \
    --model.layers.2.stride 3 3 \
    # Set the parameters of the next layer (Pool)
    --model.layers.3.pool_size 2 2 \
    # Set the parameters of the next layer (BiLSTM)
    --model.layers.4.hidden 200 \
    # Set the parameters of the next layer (Dropout)
    --model.layers.5.rate 0.5
```

For a full set of parameters of the different layers, have a look at the model parameters.

2.2.11 Learning Rate

The learning rate can be modified by `--learning_rate.lr`, the complete schedule by `--learning_rate` The default schedule is a constant learning rate `--learning_rate Const --learning_rate 0.001`.

2.2.12 Codec

By default the codec, that is the alphabet to detect, is automatically computed based on the Ground Truth files of both the training and validation sets.

There are additional parameters that modify this behaviour:

- `--codec.auto_compute` can be set to `False` while passing either
- `--codec.include` to specify a list of characters or
- `--codec.include_files` to set a path to a file with the list of characters.

An example usage is the combination with *training on the fly*:

```
calamari-train --train.preload False --val.preload False \
    --codec.auto_compute False \
    --codec.include a b c d e f g ... \  # Either list all characters, or
    --codec.include_files alphabet.txt   # pass a list
```

2.2.13 Optimizer

The default optimizer is `--optimizer Adam` but can be adapted to e.g. `SGD`, `RMSProp`, or `AdaBelief`.

Gradient Clipping

Any optimizer supports gradient clipping by either passing `--optimizer.clip_norm`, `--optimizer.clip_value`, or `--optimizer.clip_global_norm`.

2.2.14 All Parameters

```
usage: calamari-train [--version] [--show] [-h]

optional arguments:
  --version             show program's version number and exit
  --show                show the parsed parameters
  -h, --help            show this help message and exit

optional arguments:
  --trainer TRAINER
  --trainer.epochs TRAINER.EPOCHS
                        The number of training epochs. (default: 100)
  --trainer.current_epoch TRAINER.CURRENT_EPOCH
                        The epoch to start with. Usually 0, but can be overwritten for␣
↪resume training. (default: 0)
  --trainer.samples_per_epoch TRAINER.SAMPLES_PER_EPOCH
                        The number of samples (not batches!) to process per epoch. By␣
```

(continues on next page)

(continued from previous page)

```
↪default (-1) the size of the training dataset. (default: -1)
  --trainer.scale_epoch_size TRAINER.SCALE_EPOCH_SIZE
                        Multiply the number of samples per epoch by this factor. This is␣
↪useful when using the dataset size as samples per epoch (--samples_per_epoch=-1, the␣
↪default), but if you desire to set it e.g. to the half dataset size
                        (--scale_epoch_size=0.5) (default: 1)
  --trainer.train_accum_steps TRAINER.TRAIN_ACCUM_STEPS
                        Artificially increase the batch size by accumulating the␣
↪gradients of n_steps(=batches) before applying them. This factor has to be multiplied␣
↪with data_params.train_batch_size to compute the "actual" batch size (default: 1)
  --trainer.progress_bar_mode TRAINER.PROGRESS_BAR_MODE
                        Verbose level of the progress bar. (default: 1)
  --trainer.progbar_delta_time TRAINER.PROGBAR_DELTA_TIME
                        If verbose=2 the interval after which to output the current␣
↪progress (default: 5)
  --trainer.tf_cpp_min_log_level TRAINER.TF_CPP_MIN_LOG_LEVEL
                        The log level for tensorflow cpp code. (default: 2)
  --trainer.force_eager TRAINER.FORCE_EAGER
                        Activate eager execution of the graph. See also --scenario debug_
↪graph_construction (default: False)
  --trainer.skip_model_load_test TRAINER.SKIP_MODEL_LOAD_TEST
                        By default, the trainer checks initially whether the prediction␣
↪model can be saved and loaded. This may take some time. Thus for debugging you should␣
↪skip this by setting it to True (default: False)
  --trainer.val_every_n TRAINER.VAL_EVERY_N
                        Rate at which to test the model on the validation data (--data_
↪params validation_list) (default: 1)
  --trainer.lav_every_n TRAINER.LAV_EVERY_N
                        Rate at which to LAV the model during training (similar to test,␣
↪however on the actual prediction model).LAV uses --data_params lav_lists (default: 0)
  --trainer.output_dir TRAINER.OUTPUT_DIR
                        Dictionary to use to write checkpoints, logging files, and␣
↪export of best and last model. (default: None)
  --trainer.write_checkpoints TRAINER.WRITE_CHECKPOINTS
                        Write checkpoints to output_dir during training. Checkpoints are␣
↪obligatory if you want support to resume the training (see tfaip-resume-training␣
↪script) (default: True)
  --trainer.export_best TRAINER.EXPORT_BEST
                        Continuously export the best model during testing to output_dir/
↪best. (default: None)
  --trainer.export_final TRAINER.EXPORT_FINAL
                        Export the final model after training to output_dir/export.␣
↪(default: True)
  --trainer.no_train_scope TRAINER.NO_TRAIN_SCOPE
                        Regex to match with layer names to exclude from training, i.e.␣
↪the weights of these layers will not receive updates (default: None)
  --trainer.ema_decay TRAINER.EMA_DECAY
                        Calculate ema weights by decaying the current training weights␣
↪with the given factor. These weights are exported as best or final (prediction model).␣
↪0.0 means OFF, greater zero uses this value directly, less than zero calculates ema␣
↪decay
                        value dynamically. Values greater equals 1 are not supported.␣
```

(continues on next page)

(continued from previous page)

```
↪(default: 0.0)
  --trainer.random_seed TRAINER.RANDOM_SEED
                        Random seed for all random generators. Use this to obtain␣
↪reproducible results (at least on CPU) (default: None)
  --trainer.profile TRAINER.PROFILE
                        Enable profiling for tensorboard, profiling batch 10 to 20,␣
↪initial setup:pip install -U tensorboard_plugin_profileLD_LIBRARY_PATH=:/usr/local/
↪cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64options nvidia
                        "NVreg_RestrictProfilingToAdminUsers=0" to /etc/modprobe.d/
↪nvidia-kernel-common.confreboot system (default: False)
  --device DEVICE
  --optimizer OPTIMIZER
  --learning_rate LEARNING_RATE
  --scenario SCENARIO
  --warmstart WARMSTART
  --early_stopping EARLY_STOPPING
  --trainer.gen TRAINER.GEN
  --trainer.version TRAINER.VERSION
                        Missing help string (default: 4)
  --trainer.data_aug_retrain_on_original TRAINER.DATA_AUG_RETRAIN_ON_ORIGINAL
                        When training with augmentations usually the model is retrained␣
↪in a second run with only the non augmented data. This will take longer. Use this flag␣
↪to disable this behavior. (default: True)
  --trainer.current_stage TRAINER.CURRENT_STAGE
                        Missing help string (default: 0)
  --trainer.progress_bar TRAINER.PROGRESS_BAR
                        Missing help string (default: True)
  --trainer.auto_upgrade_checkpoints TRAINER.AUTO_UPGRADE_CHECKPOINTS
                        Missing help string (default: True)
  --codec CODEC
  --trainer.best_model_prefix TRAINER.BEST_MODEL_PREFIX
                        The prefix of the best model using early stopping (default: best)
  --network NETWORK     Pass a network configuration to construct a simple graph.␣
↪Defaults to: --network=cnn=40:3x3,pool=2x2,cnn=60:3x3,pool=2x2,lstm=200,dropout=0.5␣
↪(default: None)
  --device.gpus [DEVICE.GPUS [DEVICE.GPUS ...]]
                        List of the GPUs to use. (default: [])
  --device.gpu_auto_tune DEVICE.GPU_AUTO_TUNE
                        Enable auto tuning of the GPUs (default: False)
  --device.gpu_memory DEVICE.GPU_MEMORY
                        Limit the per GPU memory in MB. By default the memory will grow␣
↪automatically (default: None)
  --device.soft_device_placement DEVICE.SOFT_DEVICE_PLACEMENT
                        Set up soft device placement is enabled (default: True)
  --device.dist_strategy {DEFAULT,CENTRAL_STORAGE,MIRROR,default,central_storage,mirror}
                        Distribution strategy for multi GPU, select 'mirror' or 'central_
↪storage' (default: DistributionStrategy.DEFAULT)
  --optimizer.clip_norm OPTIMIZER.CLIP_NORM
                        float or None. If set, clips gradients to a maximum norm.␣
↪(default: None)
  --optimizer.clip_value OPTIMIZER.CLIP_VALUE
                        float or None. If set, clips gradients to a maximum value.␣
```

(continues on next page)

(continued from previous page)

```
↪(default: None)
  --optimizer.global_clip_norm OPTIMIZER.GLOBAL_CLIP_NORM
                        float or None. If set, the gradient of all weights is clipped so␣
↪that their global norm is no higher than this value. (default: None)
  --optimizer.beta_1 OPTIMIZER.BETA_1
                        Missing help string (default: 0.9)
  --optimizer.beta_2 OPTIMIZER.BETA_2
                        Missing help string (default: 0.999)
  --optimizer.epsilon OPTIMIZER.EPSILON
                        Missing help string (default: 1e-07)
  --optimizer.weight_decay OPTIMIZER.WEIGHT_DECAY
                        Missing help string (default: 0.0)
  --learning_rate.lr LEARNING_RATE.LR
                        The learning rate. (default: 0.001)
  --learning_rate.step_function LEARNING_RATE.STEP_FUNCTION
                        (type dependent) Step function of exponential decay. (default:␣
↪True)
  --learning_rate.offset_epochs LEARNING_RATE.OFFSET_EPOCHS
                        Offset to subtract from the current training epoch (if the total␣
↪is negative it will be capped at 0, and i.e., if < 0 the total epoch is greater than␣
↪the training epoch). Can be used to reset the learning rate schedule when resuming
                        training. (default: 0)
  --scenario.debug_graph_construction SCENARIO.DEBUG_GRAPH_CONSTRUCTION
                        Build the graph in pure eager mode to debug the graph␣
↪construction on real data (default: False)
  --scenario.debug_graph_n_examples SCENARIO.DEBUG_GRAPH_N_EXAMPLES
                        number of examples to take from the validation set for debugging,
↪ -1 = all (default: 1)
  --scenario.print_eval_limit SCENARIO.PRINT_EVAL_LIMIT
                        Number of evaluation examples to print per evaluation, use -1 to␣
↪print all (default: 10)
  --scenario.tensorboard_logger_history_size SCENARIO.TENSORBOARD_LOGGER_HISTORY_SIZE
                        Number of instances to store for outputting into tensorboard.␣
↪Default (last n=5) (default: 5)
  --scenario.export_serve SCENARIO.EXPORT_SERVE
                        Export the serving model (saved model format) (default: True)
  --model MODEL
  --data DATA
  --evaluator EVALUATOR
  --warmstart.model WARMSTART.MODEL
                        Path to the saved model or checkpoint to load the weights from.␣
↪(default: None)
  --warmstart.allow_partial WARMSTART.ALLOW_PARTIAL
                        Allow that not all weights can be matched. (default: False)
  --warmstart.trim_graph_name WARMSTART.TRIM_GRAPH_NAME
                        Remove the graph name from the loaded model and the target model.
↪ This is useful if the model name changed (default: True)
  --warmstart.rename [WARMSTART.RENAME [WARMSTART.RENAME ...]]
                        A list of renaming rules to perform on the loaded weights.␣
↪Format: FROM->TO FROM->TO ... (default: [])
  --warmstart.rename_targets [WARMSTART.RENAME_TARGETS [WARMSTART.RENAME_TARGETS ...]]
                        A list of renaming rules to perform on the target weights.␣
```

(continues on next page)

(continued from previous page)

```
↪Format: FROM->TO FROM->TO ... (default: [])
  --warmstart.exclude WARMSTART.EXCLUDE
                        A regex applied on the loaded weights to ignore from loading.␣
↪(default: None)
  --warmstart.include WARMSTART.INCLUDE
                        A regex applied on the loaded weights to include from loading.␣
↪(default: None)
  --warmstart.auto_remove_numbers_for [WARMSTART.AUTO_REMOVE_NUMBERS_FOR [WARMSTART.AUTO_
↪REMOVE_NUMBERS_FOR ...]]
                        Missing help string (default: ['lstm_cell'])
  --early_stopping.best_model_output_dir EARLY_STOPPING.BEST_MODEL_OUTPUT_DIR
                        Override the default output_dir of the best model. (default:␣
↪None)
  --early_stopping.best_model_name EARLY_STOPPING.BEST_MODEL_NAME
                        Name of the best model. (default: best)
  --early_stopping.frequency EARLY_STOPPING.FREQUENCY
                        Frequency in terms of epochs when to test for a new best model.␣
↪Defaults to 1, i.e. after each epoch. (default: 1)
  --early_stopping.n_to_go EARLY_STOPPING.N_TO_GO
                        Set to a value > 0 to enable early stopping, i.e. if not better␣
↪model was found after n_to_go epochs (modify by frequency), training is stopped.␣
↪(default: -1)
  --early_stopping.lower_threshold EARLY_STOPPING.LOWER_THRESHOLD
                        Threshold that must be reached at least (if mode=max) to count␣
↪for early stopping, or stop training immediately (if mode=min) if the monitored value␣
↪is lower. E.g. 0 for an accuracy. (default: -1e+100)
  --early_stopping.upper_threshold EARLY_STOPPING.UPPER_THRESHOLD
                        If mode=min the monitored value must be lower to count for early␣
↪stopping, or if mode=max and the threshold is exceeded training is stopped immediately.
↪ E.g. 1 for an accuracy. (default: 1e+100)
  --train.batch_size TRAIN.BATCH_SIZE
                        Batch size (default: 16)
  --train.limit TRAIN.LIMIT
                        Limit the number of examples produced by the generator. Note, if␣
↪GeneratingDataProcessors are present in the data pipeline, the number of examples␣
↪produced by the generator can differ. (default: -1)
  --train.prefetch TRAIN.PREFETCH
                        Prefetching data. -1 default to max(num_processes * 2 by default,
↪ 2 * batch size) (default: -1)
  --train.num_processes TRAIN.NUM_PROCESSES
                        Number of processes for data loading. (default: 4)
  --train.batch_drop_remainder TRAIN.BATCH_DROP_REMAINDER
                        Drop remainder parameter of padded_batch. Drop batch if it is␣
↪smaller than batch size. (default: False)
  --train.shuffle_buffer_size TRAIN.SHUFFLE_BUFFER_SIZE
                        Size of the shuffle buffer required for randomizing data (if␣
↪required). Disabled by default. (default: -1)
  --train.bucket_boundaries [TRAIN.BUCKET_BOUNDARIES [TRAIN.BUCKET_BOUNDARIES ...]]
                        Elements of the Dataset are grouped together by length and then␣
↪are padded and batched. See tf.data.experimental.bucket_by_sequence_length (default:␣
↪[])
  --train.bucket_batch_sizes [TRAIN.BUCKET_BATCH_SIZES [TRAIN.BUCKET_BATCH_SIZES ...]]
```

(continues on next page)

(continued from previous page)

```
                        Batch sizes of the buckets. By default, batch_size * (len(bucked_
↪boundaries) + 1). (default: None)
  --val.batch_size VAL.BATCH_SIZE
                        Batch size (default: 16)
  --val.limit VAL.LIMIT
                        Limit the number of examples produced by the generator. Note, if␣
↪GeneratingDataProcessors are present in the data pipeline, the number of examples␣
↪produced by the generator can differ. (default: -1)
  --val.prefetch VAL.PREFETCH
                        Prefetching data. -1 default to max(num_processes * 2 by default,
↪ 2 * batch size) (default: -1)
  --val.num_processes VAL.NUM_PROCESSES
                        Number of processes for data loading. (default: 4)
  --val.batch_drop_remainder VAL.BATCH_DROP_REMAINDER
                        Drop remainder parameter of padded_batch. Drop batch if it is␣
↪smaller than batch size. (default: False)
  --val.shuffle_buffer_size VAL.SHUFFLE_BUFFER_SIZE
                        Size of the shuffle buffer required for randomizing data (if␣
↪required). Disabled by default. (default: -1)
  --val.bucket_boundaries [VAL.BUCKET_BOUNDARIES [VAL.BUCKET_BOUNDARIES ...]]
                        Elements of the Dataset are grouped together by length and then␣
↪are padded and batched. See tf.data.experimental.bucket_by_sequence_length (default:␣
↪[])
  --val.bucket_batch_sizes [VAL.BUCKET_BATCH_SIZES [VAL.BUCKET_BATCH_SIZES ...]]
                        Batch sizes of the buckets. By default, batch_size * (len(bucked_
↪boundaries) + 1). (default: None)
  --train TRAIN
  --val VAL
  --codec.keep_loaded CODEC.KEEP_LOADED
                        Fully include the codec of the loaded model to the new codec␣
↪(default: True)
  --codec.auto_compute CODEC.AUTO_COMPUTE
                        Compute the codec automatically. See also include. (default:␣
↪True)
  --codec.include [CODEC.INCLUDE [CODEC.INCLUDE ...]]
                        Whitelist of characters that may not be removed on restoring a␣
↪model. For large dataset you can use this to skip the automatic codec computation (see␣
↪auto_compute) (default: [])
  --codec.include_files [CODEC.INCLUDE_FILES [CODEC.INCLUDE_FILES ...]]
                        Whitelist of txt files that may not be removed on restoring a␣
↪model (default: [])
  --model.layers [MODEL.LAYERS [MODEL.LAYERS ...]]
  --model.classes MODEL.CLASSES
                        Missing help string (default: -1)
  --model.ctc_merge_repeated MODEL.CTC_MERGE_REPEATED
                        Missing help string (default: True)
  --model.ensemble MODEL.ENSEMBLE
                        Missing help string (default: 0)
  --model.masking_mode MODEL.MASKING_MODE
                        Missing help string (default: False)
  --data.pre_proc DATA.PRE_PROC
  --data.post_proc DATA.POST_PROC
```

(continues on next page)

(continued from previous page)

```
  --data.skip_invalid_gt DATA.SKIP_INVALID_GT
                        Missing help string (default: True)
  --data.input_channels DATA.INPUT_CHANNELS
                        Missing help string (default: 1)
  --data.line_height DATA.LINE_HEIGHT
                        The line height (default: 48)
  --train.skip_invalid TRAIN.SKIP_INVALID
                        Missing help string (default: False)
  --train.non_existing_as_empty TRAIN.NON_EXISTING_AS_EMPTY
                        Missing help string (default: False)
  --train.preload TRAIN.PRELOAD
                        Instead of preloading all data, load the data on the fly. This␣
↪is slower, but might be required for limited RAM or large dataset (default: True)
  --train.images [TRAIN.IMAGES [TRAIN.IMAGES ...]]
                        List all image files that shall be processed. Ground truth files␣
↪with the same base name but with '.gt.txt' as extension are required at the same␣
↪location (default: [])
  --train.texts [TRAIN.TEXTS [TRAIN.TEXTS ...]]
                        List the text files (default: [])
  --train.gt_extension TRAIN.GT_EXTENSION
                        Extension of the gt files (expected to exist in same dir)␣
↪(default: .gt.txt)
  --train.pred_extension TRAIN.PRED_EXTENSION
                        Extension of prediction text files (default: .pred.txt)
  --val.skip_invalid VAL.SKIP_INVALID
                        Missing help string (default: False)
  --val.non_existing_as_empty VAL.NON_EXISTING_AS_EMPTY
                        Missing help string (default: False)
  --val.preload VAL.PRELOAD
                        Instead of preloading all data, load the data on the fly. This␣
↪is slower, but might be required for limited RAM or large dataset (default: True)
  --val.images [VAL.IMAGES [VAL.IMAGES ...]]
                        List all image files that shall be processed. Ground truth files␣
↪with the same base name but with '.gt.txt' as extension are required at the same␣
↪location (default: [])
  --val.texts [VAL.TEXTS [VAL.TEXTS ...]]
                        List the text files (default: [])
  --val.gt_extension VAL.GT_EXTENSION
                        Extension of the gt files (expected to exist in same dir)␣
↪(default: .gt.txt)
  --val.pred_extension VAL.PRED_EXTENSION
                        Extension of prediction text files (default: .pred.txt)
  --model.layers.0.name MODEL.LAYERS.0.NAME
                        Missing help string (default: None)
  --model.layers.0.filters MODEL.LAYERS.0.FILTERS
                        Missing help string (default: 40)
  --model.layers.0.kernel_size MODEL.LAYERS.0.KERNEL_SIZE
  --model.layers.0.strides MODEL.LAYERS.0.STRIDES
  --model.layers.0.padding MODEL.LAYERS.0.PADDING
                        Missing help string (default: same)
  --model.layers.0.activation MODEL.LAYERS.0.ACTIVATION
                        Missing help string (default: relu)
```

(continues on next page)

(continued from previous page)

```
  --model.layers.1.name MODEL.LAYERS.1.NAME
                        Missing help string (default: None)
  --model.layers.1.pool_size MODEL.LAYERS.1.POOL_SIZE
  --model.layers.1.strides MODEL.LAYERS.1.STRIDES
  --model.layers.1.padding MODEL.LAYERS.1.PADDING
                        Missing help string (default: same)
  --model.layers.2.name MODEL.LAYERS.2.NAME
                        Missing help string (default: None)
  --model.layers.2.filters MODEL.LAYERS.2.FILTERS
                        Missing help string (default: 40)
  --model.layers.2.kernel_size MODEL.LAYERS.2.KERNEL_SIZE
  --model.layers.2.strides MODEL.LAYERS.2.STRIDES
  --model.layers.2.padding MODEL.LAYERS.2.PADDING
                        Missing help string (default: same)
  --model.layers.2.activation MODEL.LAYERS.2.ACTIVATION
                        Missing help string (default: relu)
  --model.layers.3.name MODEL.LAYERS.3.NAME
                        Missing help string (default: None)
  --model.layers.3.pool_size MODEL.LAYERS.3.POOL_SIZE
  --model.layers.3.strides MODEL.LAYERS.3.STRIDES
  --model.layers.3.padding MODEL.LAYERS.3.PADDING
                        Missing help string (default: same)
  --model.layers.4.name MODEL.LAYERS.4.NAME
                        Missing help string (default: None)
  --model.layers.4.hidden_nodes MODEL.LAYERS.4.HIDDEN_NODES
                        Missing help string (default: 200)
  --model.layers.4.merge_mode MODEL.LAYERS.4.MERGE_MODE
                        Missing help string (default: concat)
  --model.layers.5.name MODEL.LAYERS.5.NAME
                        Missing help string (default: None)
  --model.layers.5.rate MODEL.LAYERS.5.RATE
                        Missing help string (default: 0.5)
  --data.pre_proc.run_parallel DATA.PRE_PROC.RUN_PARALLEL
                        Run this pipeline in parallel. (default: True)
  --data.pre_proc.num_threads DATA.PRE_PROC.NUM_THREADS
                        The number of threads to use for this pipeline. Else use the␣
↪value of the generator params. (default: -1)
  --data.pre_proc.max_tasks_per_process DATA.PRE_PROC.MAX_TASKS_PER_PROCESS
                        Maximum tasks of a child in the preproc pipeline after a child␣
↪is recreated. Higher numbers for better performance but on the drawback if higher␣
↪memory consumption. Only used if the scenario uses a DataPipeline. (default: 250)
  --data.pre_proc.processors [DATA.PRE_PROC.PROCESSORS [DATA.PRE_PROC.PROCESSORS ...]]
  --data.post_proc.run_parallel DATA.POST_PROC.RUN_PARALLEL
                        Run this pipeline in parallel. (default: True)
  --data.post_proc.num_threads DATA.POST_PROC.NUM_THREADS
                        The number of threads to use for this pipeline. Else use the␣
↪value of the generator params. (default: -1)
  --data.post_proc.max_tasks_per_process DATA.POST_PROC.MAX_TASKS_PER_PROCESS
                        Maximum tasks of a child in the preproc pipeline after a child␣
↪is recreated. Higher numbers for better performance but on the drawback if higher␣
↪memory consumption. Only used if the scenario uses a DataPipeline. (default: 250)
  --data.post_proc.processors [DATA.POST_PROC.PROCESSORS [DATA.POST_PROC.PROCESSORS ...]]
```

(continues on next page)

(continued from previous page)

```
  --model.layers.0.kernel_size.x MODEL.LAYERS.0.KERNEL_SIZE.X
                        Missing help string (default: None)
  --model.layers.0.kernel_size.y MODEL.LAYERS.0.KERNEL_SIZE.Y
                        Missing help string (default: None)
  --model.layers.0.strides.x MODEL.LAYERS.0.STRIDES.X
                        Missing help string (default: None)
  --model.layers.0.strides.y MODEL.LAYERS.0.STRIDES.Y
                        Missing help string (default: None)
  --model.layers.1.pool_size.x MODEL.LAYERS.1.POOL_SIZE.X
                        Missing help string (default: None)
  --model.layers.1.pool_size.y MODEL.LAYERS.1.POOL_SIZE.Y
                        Missing help string (default: None)
  --model.layers.2.kernel_size.x MODEL.LAYERS.2.KERNEL_SIZE.X
                        Missing help string (default: None)
  --model.layers.2.kernel_size.y MODEL.LAYERS.2.KERNEL_SIZE.Y
                        Missing help string (default: None)
  --model.layers.2.strides.x MODEL.LAYERS.2.STRIDES.X
                        Missing help string (default: None)
  --model.layers.2.strides.y MODEL.LAYERS.2.STRIDES.Y
                        Missing help string (default: None)
  --model.layers.3.pool_size.x MODEL.LAYERS.3.POOL_SIZE.X
                        Missing help string (default: None)
  --model.layers.3.pool_size.y MODEL.LAYERS.3.POOL_SIZE.Y
                        Missing help string (default: None)
  --data.pre_proc.processors.0.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.1.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.1.extra_params DATA.PRE_PROC.PROCESSORS.1.EXTRA_PARAMS
                        Missing help string (default: (4, 1.0, 0.3))
  --data.pre_proc.processors.1.line_height DATA.PRE_PROC.PROCESSORS.1.LINE_HEIGHT
                        Missing help string (default: -1)
  --data.pre_proc.processors.2.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.2.normalize DATA.PRE_PROC.PROCESSORS.2.NORMALIZE
                        Missing help string (default: True)
  --data.pre_proc.processors.2.invert DATA.PRE_PROC.PROCESSORS.2.INVERT
```

(continues on next page)

(continued from previous page)

```
                        Missing help string (default: True)
  --data.pre_proc.processors.2.transpose DATA.PRE_PROC.PROCESSORS.2.TRANSPOSE
                        Missing help string (default: True)
  --data.pre_proc.processors.2.pad DATA.PRE_PROC.PROCESSORS.2.PAD
                        Padding (left right) of the line (default: 16)
  --data.pre_proc.processors.2.pad_value DATA.PRE_PROC.PROCESSORS.2.PAD_VALUE
                        Missing help string (default: 0)
  --data.pre_proc.processors.3.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.3.bidi_direction {LTR,RTL,AUTO,L,R,auto}
                        The default text direction when preprocessing bidirectional text.
↪ Supported values are 'auto' to automatically detect the direction, 'ltr' and 'rtl'␣
↪for left-to-right and right-to-left, respectively (default: BidiDirection.AUTO)
  --data.pre_proc.processors.4.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.5.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.5.unicode_normalization DATA.PRE_PROC.PROCESSORS.5.UNICODE_
↪NORMALIZATION
                        Unicode text normalization to apply. Defaults to NFC (default:␣
↪NFC)
  --data.pre_proc.processors.6.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.pre_proc.processors.6.replacement_groups [DATA.PRE_PROC.PROCESSORS.6.
↪REPLACEMENT_GROUPS [DATA.PRE_PROC.PROCESSORS.6.REPLACEMENT_GROUPS ...]]
                        Text regularization to apply. (default: ['extended'])
  --data.pre_proc.processors.7.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
```

(continues on next page)

(continued from previous page)

```
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --augmenter AUGMENTER
  --n_augmentations N_AUGMENTATIONS
                        Amount of data augmentation per line (done before training). If␣
↪this number is < 1 the amount is relative. (default: 0)
  --data.pre_proc.processors.8.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.0.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.1.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.1.ctc_decoder_params DATA.POST_PROC.PROCESSORS.1.CTC_
↪DECODER_PARAMS
  --data.post_proc.processors.2.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.2.bidi_direction {LTR,RTL,AUTO,L,R,auto}
                        The default text direction when preprocessing bidirectional text.
↪ Supported values are 'auto' to automatically detect the direction, 'ltr' and 'rtl'␣
↪for left-to-right and right-to-left, respectively (default: BidiDirection.AUTO)
  --data.post_proc.processors.3.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.4.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
```

(continues on next page)

(continued from previous page)

```
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.4.unicode_normalization DATA.POST_PROC.PROCESSORS.4.
↪UNICODE_NORMALIZATION
                        Unicode text normalization to apply. Defaults to NFC (default:␣
↪NFC)
  --data.post_proc.processors.5.modes [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} [{TRAINING,EVALUATION,PREDICTION,TARGETS,training,
↪evaluation,prediction,targets} ...]]
                        The PipelineModes when to apply this DataProcessor (e.g., only␣
↪during PipelineMode.TRAINING) (default: {<PipelineMode.TRAINING: 'training'>,
↪<PipelineMode.PREDICTION: 'prediction'>, <PipelineMode.TARGETS: 'targets'>,
                        <PipelineMode.EVALUATION: 'evaluation'>})
  --data.post_proc.processors.5.replacement_groups [DATA.POST_PROC.PROCESSORS.5.
↪REPLACEMENT_GROUPS [DATA.POST_PROC.PROCESSORS.5.REPLACEMENT_GROUPS ...]]
                        Text regularization to apply. (default: ['extended'])
  --data.post_proc.processors.1.ctc_decoder_params.type {Default,TokenPassing,
↪WordBeamSearch,default,token_passing,word_beam_search}
                        Missing help string (default: CTCDecoderType.Default)
  --data.post_proc.processors.1.ctc_decoder_params.blank_index DATA.POST_PROC.PROCESSORS.
↪1.CTC_DECODER_PARAMS.BLANK_INDEX
                        Missing help string (default: 0)
  --data.post_proc.processors.1.ctc_decoder_params.min_p_threshold DATA.POST_PROC.
↪PROCESSORS.1.CTC_DECODER_PARAMS.MIN_P_THRESHOLD
                        Missing help string (default: 0)
  --data.post_proc.processors.1.ctc_decoder_params.non_word_chars [DATA.POST_PROC.
↪PROCESSORS.1.CTC_DECODER_PARAMS.NON_WORD_CHARS [DATA.POST_PROC.PROCESSORS.1.CTC_
↪DECODER_PARAMS.NON_WORD_CHARS ...]]
                        Missing help string (default: ['0', '1', '2', '3', '4', '5', '6',
↪ '7', '8', '9', '[', ']', '(', ')', '_', '.', ':', ';', '!', '?', '{', '}', '-', "'", '
↪"'])
  --data.post_proc.processors.1.ctc_decoder_params.dictionary [DATA.POST_PROC.PROCESSORS.
↪1.CTC_DECODER_PARAMS.DICTIONARY [DATA.POST_PROC.PROCESSORS.1.CTC_DECODER_PARAMS.
↪DICTIONARY ...]]
                        Missing help string (default: [])
  --data.post_proc.processors.1.ctc_decoder_params.word_separator DATA.POST_PROC.
↪PROCESSORS.1.CTC_DECODER_PARAMS.WORD_SEPARATOR
                        Missing help string (default: )
```

2.3 calamari-resume-training

This script can be used to resume from checkpoints that are written during training in the `checkpoint` directory located within the `--output_dir`. Call

```
calamari-resume-training OUTPUT_DIR/checkpoint/checkpoint_XXX/trainer_params.json
```

to resume training from a certain checkpoint. Modify the `trainer_params.json` to adapt training, e.g., extending the number of epochs or adapting early stopping.

2.4 calamari-cross-fold-train

Calamari allows to train an ensemble of models based on a cross-fold on the provided data. These models can then be passed to `calamari-predict` to obtain an improved prediction by voting.

The default call is

```
calamari-cross-fold-train --data.images your_images*.*.png --best_models_dir some_dir
```

By default, this will train 5 default models using 80%=(n-1)/n of the provided data for training and 20%=1/n for validation. These independent models can then be used to predict lines using a voting mechanism. There are several important parameters to adjust the training. For a full list see `calamari-cross-fold-train --help`.

- Almost parameters of calamari-train can be used to affect the training
- `--n_folds=5`: The number of folds
- `--weights=None`: Specify one or n_folds models to use for pretraining.
- `--best_models_dir=REQUIRED`: Directory where to store the best model determined on the validation data set
- `--best_model_label={id}`: The prefix for each of the best model of each fold. A string that will be formatted. {id} will be replaced by the number of the fold, i.e. 0, . . . , n-1.
- `--temporary_dir=None`: A directory where to store temporary files, e.g. checkpoints of the scripts to train an individual model. By default, a temporary dir using pythons tempfile modules is used.
- `--max_parallel_models=n_folds`: The number of models that shall be run in parallel. By default, all models are trained in parallel.
- `--single_fold=[]`: Use this parameter to train only a subset, e.g. a single fold out of all n_folds.

2.5 calamari-predict-and-eval

2.6 calamari-eval

To compute the performance of a model first predict the evaluation data set (see `calamari-predict`). Afterwards run

```
calamari-eval --gt.texts *.gt.txt
```

on the ground truth files to compute an evaluation measure including the full confusion matrix. By default, the predicted sentences as produced by the `calamari-predict` script end in `.pred.txt`. Change the default behavior of the validation script by the following parameters

- `--gt.texts=REQUIRED`: The ground truth txt files.
- `--gt.pred_extension=.pred.txt`: The suffix of the prediction files if –pred is not specified
- `--n_confusions=-1`: Print only the top n_confusions most common errors.

CHAPTER

THREE

DATASET FORMATS

3.1 File Extensions

File extensions are used to map ground truth and image files (e.g., a `.png` and its corresponding `.gt.txt`). Everything after the first dot (.) in a filename is considered as the files extension. So do not use dots to label files.

Most dataset formats provide a `.gt_extension` and a `.pred_extension` which can be used to modify the loaded and written files.

3.2 Plain files

Files provided as single images of a line (e. g. png or jpg). The ground truth is provided as plain text files in UTF-8 encoding using the same base name as the corresponding image and *.gt.txt* as extension.

Example:

```
+ train
  - 0001.png
  - 0001.gt.txt
  - 0002.png
  - 0002.gt.txt
  - ...
+ test
  - 1001.png
  - 1001.gt.txt
  - 1002.png
  - 1002.gt.txt
  - ...
```

Command-Line Call:

```
calamari-train --train.images train/*.png
calamari-predict --data.images test/*.png
```

3.3 PageXML

Use `--train PageXML` to switch mode. Provide page images as `--train.images` and the corresponding `xml`-files as `--xml_files`

Example structure:

```
+ train
  - 0001.png
  - 0001.xml
  - 0002.png
  - 0002.xml
  - ...
+ test
  - 1001.png
  - 1001.xml
  - 1002.png
  - 1002.xml
  - ...
```

Call:

```
calamari-train --train PageXML --train.images train/*.png
calamari-predict --data PageXML --data.images test/*.png
```

3.4 Abbyy

Use `--train Abbyy` to switch mode. Provide page images as `--train.images`, the corresponding `xml` files must end with `.abbyy.xml`.

Example structure:

```
+ train
  - 0001.png
  - 0001.abbyy.xml
  - 0002.png
  - 0002.abbyy.xml
  - ...
+ test
  - 1001.png
  - 1001.abbyy.xml
  - 1002.png
  - 1002.abbyy.xml
  - ...
```

Call:

```
calamari-train --train Abbyy --train.images train/*.png
calamari-predict --data Abbyy --data.images test/*.png
```

3.5 HDF5

Use `--train Hdf5` to switch mode.

The content of a *h5*-file is:

- `images`: list of raw images
- `images_dims`: the shape of the images (numpy arrays)
- `codec`: integer mapping to decode the `transcriptions` (ASCII)
- `transcripts`: list of encoded transcriptions using the codec

Call

```
calamari-train --train Hdf5 --train.files train.h5
calamari-predict --train Hdf5 --train.files test.h5
```

CHAPTER

FOUR

PREDICTING

First you need to create a predictor object giving an existing model.

4.1 Python API

The predictor must be created once but can then be used for multiple predictions.

4.1.1 Single Model

There are two options for prediction depending on your data and usecase.

If you have already a *list of data* use the code below: create the predictor and call `predict_raw`, `predict_pipeline`, `predict_dataset`. This will setup all internal pipelines and close them afterwards automatically.

```
from calamari_ocr.ocr.predict.predictor import Predictor, PredictorParams
predictor = Predictor.from_checkpoint(
    params=PredictorParams(),
    checkpoint='PATH_TO_THE_MODEL_WITHOUT_EXT')

for sample in predictor.predict_raw(raw_image_generator):
    inputs, prediction, meta = sample.inputs, sample.outputs, sample.meta
    # prediction is usually what you are looking for
```

Whereby a `raw_image_generator` is of type `Iterable[np.ndarray]` for example a list of images:

```
raw_image_generator = [np.zeros(shape=(200, 50))]
```

If instead the *samples (lines) are dynamically created* during the execution and the predictor shall be kept alive use the following:

```
# Create the predictor, and the raw predictor somewhere in your code
from calamari_ocr.ocr.predict.predictor import Predictor, PredictorParams
predictor = Predictor.from_checkpoint(
    params=PredictorParams(),
    checkpoint='PATH_TO_THE_MODEL_WITHOUT_EXT')
raw_predictor = predictor.raw().__enter__()  # you can also wrap the following lines in␣
↪a `with`-block

# somewhere else in your code, just call the raw_predictor with a single image
sample = raw_predictor(raw_image)  # raw_image is e.g. np.zeros(200, 50)
```

(continues on next page)

(continued from previous page)

```
inputs, prediction, meta = sample.inputs, sample.outputs, sample.meta
# prediction is usually what you are looking for
```

Have a look at the prediction tests for some more examples.

4.1.2 Multiple models (voting)

```
from calamari_ocr.ocr.predict.predictor import MultiPredictor, PredictorParams
predictor = MultiPredictor.from_paths(
    checkpoints=['CKPT1', 'CKPT2', ...],
    params=PredictorParams())

for sample in predictor.predict_raw(raw_image_generator):
    inputs, (results, prediction), meta = sample.inputs, sample.outputs, sample.meta
    # prediction (the voted result) is usually what you are looking for
```

Then, apply the `predictor` to any data.

4.2 Prediction Object

Each `prediction` holds a full prediction object which holds the actual outcome (`prediction.sentence`) but also the single character positions and probabilities (`prediction.positions`).

4.3 Extended Prediction Data

A **full prediction** object generated from the `Predictor` yields a lot of information, such as **character probabilities** or **positions**. For a full overview see the dataclass structure *here*. To generate those data you can either

- pass the `--extended_prediction_data` parameter to `calamari-predict` which will create `.json` files with the additional data written (note the huge probability matrix is not included by default), or
- call the predictor in custom python code see *Prediction*

CHAPTER

FIVE

API USAGE

The Calamari API can be used to modify the complete workflow. This document shows how to adapt different typical use-cases.

5.1 Data Generator

5.2 Data Augmentation

CHAPTER

SIX

CALAMARI_OCR.OCR.DATASET

6.1 .datareader.abbyy

class calamari_ocr.ocr.dataset.datareader.abbyy.reader.**Abbyy**(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *images: List[str] = <factory>*, *xml_files: List[str] = <factory>*, *gt_extension: str = '.abbyy.xml'*, *binary: bool = False*, *pred_extension: str = '.abbyy.pred.xml'*)

Bases: CalamariDataGeneratorParams

images: List[str]

xml_files: List[str]

gt_extension: str = '.abbyy.xml'

binary: bool = False

pred_extension: str = '.abbyy.pred.xml'

select(*indices: List[int]*)

to_prediction()

static cls()

prepare_for_mode(*mode: PipelineMode*)

__init__(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *images: ~typing.List[str] = <factory>*, *xml_files: ~typing.List[str] = <factory>*, *gt_extension: str = '.abbyy.xml'*, *binary: bool = False*, *pred_extension: str = '.abbyy.pred.xml'*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*, *include_cls=True*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

6.2 .datareader.file

class calamari_ocr.ocr.dataset.datareader.file.**FileDataParams**(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *images: List[str] = <factory>*, *texts: List[str] = <factory>*, *gt_extension: str = '.gt.txt'*, *pred_extension: str = '.pred.txt'*)

Bases: CalamariDataGeneratorParams

images: List[str]

texts: List[str]

gt_extension: str = '.gt.txt'

pred_extension: str = '.pred.txt'

static cls()

to_prediction()

select(*indices: List[int]*)

prepare_for_mode(*mode: PipelineMode*)

__init__(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *images: ~typing.List[str] = <factory>*, *texts: ~typing.List[str] = <factory>*, *gt_extension: str = '.gt.txt'*, *pred_extension: str = '.pred.txt'*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(*, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*, *include_cls=True*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(*, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

6.3 .datareader.hdf5

class calamari_ocr.ocr.dataset.datareader.hdf5.reader.**Hdf5**(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *files: List[str] = <factory>*, *pred_extension: str = '.pred.h5'*)

Bases: CalamariDataGeneratorParams

files: List[str]

pred_extension: str = '.pred.h5'

to_prediction()

static cls()

prepare_for_mode(*mode: PipelineMode*)

__init__(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *files: ~typing.List[str] = <factory>*, *pred_extension: str = '.pred.h5'*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*, *include_cls=True*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

6.4 .datareader.pagexml

class calamari_ocr.ocr.dataset.datareader.pagexml.reader.**PageXML**(*channels: int = 1, to_gray_method: str = 'cv', skip_invalid: bool = True, non_existing_as_empty: bool = False, n_folds: int = -1, preload: bool = True, images: List[str] = <factory>, xml_files: List[str] = <factory>, gt_extension: str = '.xml', text_index: int = 0, pad: Union[List[int], NoneType] = None, pred_extension: str = '.pred.xml', skip_commented: bool = False, cut_mode: calamari_ocr.ocr.dataset.datareader.pagexml.reader.C = <CutMode.POLYGON: 1>, output_confidences: bool = False, output_glyphs: bool = False, max_glyph_alternatives: int = 1, delete_old_words: bool = True*)

Bases: CalamariDataGeneratorParams

images: List[str]

xml_files: List[str]

gt_extension: str = '.xml'

text_index: int = 0

pad: Optional[List[int]] = None

pred_extension: str = '.pred.xml'

skip_commented: bool = False

cut_mode: CutMode = 1

output_confidences: bool = False

output_glyphs: bool = False

max_glyph_alternatives: int = 1

delete_old_words: bool = True

select(*indices: List[int]*)

to_prediction()

static cls()

prepare_for_mode(*mode: PipelineMode*)

__init__(*channels: int = 1*, *to_gray_method: str = 'cv'*, *skip_invalid: bool = True*, *non_existing_as_empty: bool = False*, *n_folds: int = -1*, *preload: bool = True*, *images: ~typing.List[str] = <factory>*, *xml_files: ~typing.List[str] = <factory>*, *gt_extension: str = '.xml'*, *text_index: int = 0*, *pad: ~typing.Optional[~typing.List[int]] = None*, *pred_extension: str = '.pred.xml'*, *skip_commented: bool = False*, *cut_mode: ~calamari_ocr.ocr.dataset.datareader.pagexml.reader.CutMode = CutMode.POLYGON*, *output_confidences: bool = False*, *output_glyphs: bool = False*, *max_glyph_alternatives: int = 1*, *delete_old_words: bool = True*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*, *include_cls=True*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

CHAPTER

SEVEN

CALAMARI_OCR.OCR.PREDICT

class calamari_ocr.ocr.predict.params.**PredictionCharacter**(*char: str = ''*, *label: int = 0*, *probability: float = 0*)

Bases: object

char: str = ''

label: int = 0

probability: float = 0

__init__(*char: str = ''*, *label: int = 0*, *probability: float = 0*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

class calamari_ocr.ocr.predict.params.**PredictionPosition**(*chars: List[calamari_ocr.ocr.predict.params.PredictionCharacter] = <factory>*, *local_start: int = 0*, *local_end: int = 0*, *global_start: int = 0*, *global_end: int = 0*)

Bases: object

chars: List[*PredictionCharacter*]

local_start: int = 0

local_end: int = 0

global_start: int = 0

global_end: int = 0

__init__(*chars: ~typing.List[~calamari_ocr.ocr.predict.params.PredictionCharacter] = <factory>, local_start: int = 0, local_end: int = 0, global_start: int = 0, global_end: int = 0*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray], *, parse_float=None, parse_int=None, parse_constant=None, infer_missing=False, **kw*) → A

classmethod schema(**, infer_missing: bool = False, only=None, exclude=(), many: bool = False, context=None, load_only=(), dump_only=(), partial: bool = False, unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(**, skipkeys: bool = False, ensure_ascii: bool = True, check_circular: bool = True, allow_nan: bool = True, indent: Optional[Union[int, str]] = None, separators: Optional[Tuple[str, str]] = None, default: Optional[Callable] = None, sort_keys: bool = False, **kw*) → str

class calamari_ocr.ocr.predict.params.**Prediction**(*id: str = '', sentence: str = '', labels: List[int] = <factory>, positions: List[calamari_ocr.ocr.predict.params.PredictionPosition] = <factory>, logits: Union[<built-in function array>, NoneType] = None, total_probability: float = 0, avg_char_probability: float = 0, is_voted_result: bool = False, line_path: str = '', voter_predictions: Union[List[ForwardRef('Prediction')], NoneType] = None*)

Bases: object

id: str = ''

sentence: str = ''

labels: List[int]

positions: List[*PredictionPosition*]

logits: Optional[array] = None

total_probability: float = 0

avg_char_probability: float = 0

is_voted_result: bool = False

line_path: str = ''

voter_predictions: Optional[List[*Prediction*]] = None

__init__(*id: str = '', sentence: str = '', labels: ~typing.List[int] = <factory>, positions: ~typing.List[~calamari_ocr.ocr.predict.params.PredictionPosition] = <factory>, logits: ~typing.Optional[~numpy.array] = None, total_probability: float = 0, avg_char_probability: float = 0, is_voted_result: bool = False, line_path: str = '', voter_predictions: ~typing.Optional[~typing.List[~calamari_ocr.ocr.predict.params.Prediction]] = None*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]], *, infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

class calamari_ocr.ocr.predict.params.**Predictions**(*predictions: List[calamari_ocr.ocr.predict.params.Prediction] = <factory>*, *line_path: str = ''*)

Bases: object

predictions: List[*Prediction*]

line_path: str = ''

__init__(*predictions: ~typing.List[~calamari_ocr.ocr.predict.params.Prediction] = <factory>*, *line_path: str = ''*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

class calamari_ocr.ocr.predict.params.**PredictorParams**(*device: tfaip.device.device_config.DeviceConfigParams = <factory>*, *pipeline: tfaip.data.databaseparams.DataPipelineParams = <factory>*, *silent: bool = True*, *progress_bar: bool = True*, *run_eagerly: bool = False*, *include_targets: bool = False*, *include_meta: bool = False*)

Bases: PredictorParams

silent: bool = True

__init__(*device: ~tfaip.device.device_config.DeviceConfigParams = <factory>*, *pipeline: ~tfaip.data.databaseparams.DataPipelineParams = <factory>*, *silent: bool = True*, *progress_bar: bool = True*, *run_eagerly: bool = False*, *include_targets: bool = False*, *include_meta: bool = False*) → None

classmethod from_dict(*kvs: Optional[Union[dict, list, str, int, float, bool]]*, ***, *infer_missing=False*) → A

classmethod from_json(*s: Union[str, bytes, bytearray]*, ***, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *infer_missing=False*, ***kw*) → A

classmethod schema(***, *infer_missing: bool = False*, *only=None*, *exclude=()*, *many: bool = False*, *context=None*, *load_only=()*, *dump_only=()*, *partial: bool = False*, *unknown=None*) → SchemaF[A]

to_dict(*encode_json=False*) → Dict[str, Optional[Union[dict, list, str, int, float, bool]]]

to_json(***, *skipkeys: bool = False*, *ensure_ascii: bool = True*, *check_circular: bool = True*, *allow_nan: bool = True*, *indent: Optional[Union[int, str]] = None*, *separators: Optional[Tuple[str, str]] = None*, *default: Optional[Callable] = None*, *sort_keys: bool = False*, ***kw*) → str

class calamari_ocr.ocr.predict.params.**PredictionResult**(*prediction*, *codec*, *text_postproc*, *out_to_in_trans: Callable[[int], int]*, *ground_truth=None*)

Bases: object

__init__(*prediction*, *codec*, *text_postproc*, *out_to_in_trans: Callable[[int], int]*, *ground_truth=None*)

The output of a networks prediction (PredictionProto) with additional information

It stores all required information for decoding (*codec*) and interpreting the output.

Parameters

- **prediction** (*PredictionProto*) – prediction the DNN
- **codec** (*Codec*) – codec required to decode the *prediction*
- **text_postproc** (*TextPostprocessor*) – text processor to apply to the decodec *prediction* to receive the actual prediction sentence

class calamari_ocr.ocr.predict.predictor.**Predictor**(*params: PredictorParams*, *data: DataBase*, ***kwargs*)

Bases: Predictor

static from_checkpoint(*params:* PredictorParams, *checkpoint: str*, *auto_update_checkpoints=True*)

class calamari_ocr.ocr.predict.predictor.**MultiPredictor**(*voter_params*, **args*, ***kwargs*)

Bases: MultiModelPredictor

classmethod from_paths(*checkpoints: List[str]*, *auto_update_checkpoints=True*, *predictor_params: Optional[*PredictorParams*] = None*, *voter_params: Optional[VoterParams] = None*, ***kwargs*) → MultiModelPredictor

Create a MultiModePredictor. The data of the first model (in paths) will be used as the defining scenario and data (i.e. the post-processing). All data pre-procs must be identical.

Parameters

- **paths** – paths to the scenario_params (see ScenarioBase.params_from_path)
- **params** – PredictorParams
- **scenario** – Type of the ScenarioBase
- **use_first_params** – Only False is supported ATM.
- **model_paths** – Paths to the actual models saved dirs (optional), by default based on paths
- **models** – Already instantiated models (optional), by default created based on paths

- **predictor_args** – Additional args for instantiating the Predictor (that are not part of PredictorParams)

Returns

An instantiated and ready to use MultiModelPredictor

__init__(*voter_params*, **args*, ***kwargs*)

create_voter(*data_params: DataParams*) → MultiModelVoter

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

F

S

T

V

X