
tesstrain

Training workflow for Tesseract 5 as a Makefile for dependency tracking.

- Installation
 - Auxiliaries
 - Leptonica, Tesseract
 - ★ Windows
 - Python
 - Language data
- Usage
 - Choose the model name
 - Provide ground truth data
 - Train
 - Change directory assumptions
 - Make model files (traineddata)
 - Plotting CER
- License

Installation

Auxiliaries

You will need at least GNU [make](#) (minimal version 4.2), [wget](#), [find](#), [bash](#), and [unzip](#).

Leptonica, Tesseract

You will need a recent version (≥ 5.3) of tesseract built with the training tools and matching leptonica bindings. Build instructions and more can be found in the Tesseract User Manual.

Windows

1. Install the latest tesseract (e.g. from <https://digi.bib.uni-mannheim.de/tesseract/>), and make sure that tesseract is added to your PATH.
2. Install Python 3

-
3. Install Git SCM to Windows - it provides a lot of linux utilities on Windows (e.g. `find`, `unzip`, `rm`) and put `C:\Program Files\Git\usr\bin` to the beginning of your PATH variable (temporarily you can do it in `cmd` with `set PATH=C:\Program Files\Git\usr\bin;%PATH%` - unfortunately there are several Windows tools with the same name as on linux (`find`, `sort`) with different behavior/functionality and there is need to avoid them during training.
 4. Install winget/Windows Package Manager and then run `winget install ezwinports.make` and `winget install wget` to install missing tools.

Python

You need a recent version of Python 3.x. For image processing the Python library `Pillow` is used. If you don't have a global installation, please use the provided requirements file `pip install -r requirements.txt`.

Language data

Tesseract expects some configuration data (a file `radical-stroke.txt` and `*.unicarset` for all scripts) in `DATA_DIR`. To fetch them:

```
1 make tesseract-langdata
```

(While this step is only needed once and implicitly included in the `training` target, you might want to run it explicitly beforehand.)

Usage

Choose the model name

Choose a name for your model. By convention, Tesseract stack models including language-specific resources use (lowercase) three-letter codes defined in ISO 639 with additional information separated by underscore. E.g., `chi_tra_vert` for **t**raditional Chinese with **v**ertical typesetting. Language-independent (i.e. script-specific) models use the capitalized name of the script type as an identifier. E.g., `Hangul_vert` for Hangul script with vertical typesetting. In the following, the model name is referenced by `MODEL_NAME`.

Provide ground truth data

Place ground truth consisting of line images and transcriptions in the folder `data/MODEL_NAME-ground-truth`. This list of files will be split into training and evaluation data, the ratio is defined by the `RATIO_TRAIN` variable.

Images must be TIFF and have the extension `.tif` or PNG and have the extension `.png`, `.bin.png`, or `.nrm.png`.

Transcriptions must be single-line plain text and have the same name as the line image but with the image extension replaced by `.gt.txt`.

The repository contains a ZIP archive with sample ground truth, see `ocrd-testset.zip`. Extract it to `./data/foo-ground-truth` and run `make training`.

NOTE: If you want to generate line images for transcription from a full page, see tips in issue 7 and in particular @Shreeshrii's shell script.

Train

Run

```
1 make training MODEL_NAME=name-of-the-resulting-model
```

which is a shortcut for

```
1 make unicharset lists proto-model tesseract-langdata training
   MODEL_NAME=name-of-the-resulting-model
```

Run `make help` to see all the possible targets and variables:

```
1
2   Targets
3
4   unicharset      Create unicharset
5   charfreq       Show character histogram
6   lists          Create lists of lstmf filenames for training and
   eval
7   training       Start training (i.e. create .checkpoint files)
8   traineddata    Create best and fast .traineddata files from each
   .checkpoint file
9   proto-model    Build the proto model
10  tesseract-langdata Download stock unicharsets
11  evaluation      Evaluate .checkpoint models on eval dataset via
   lstmeval
12  plot           Generate train/eval error rate charts from
   training log
```

```

13     clean                Clean all generated files
14
15     Variables
16
17     MODEL_NAME            Name of the model to be built. Default: foo
18     START_MODEL           Name of the model to continue from (i.e. fine-
19         tune). Default: ''
20     PROTO_MODEL           Name of the prototype model. Default: OUTPUT_DIR
21         /MODEL_NAME.traineddata
22     WORDLIST_FILE         Optional file for dictionary DAWG. Default:
23         OUTPUT_DIR/MODEL_NAME.wordlist
24     NUMBERS_FILE          Optional file for number patterns DAWG. Default:
25         OUTPUT_DIR/MODEL_NAME.numbers
26     PUNC_FILE             Optional file for punctuation DAWG. Default:
27         OUTPUT_DIR/MODEL_NAME.punc
28     DATA_DIR             Data directory for output files, proto model,
29         start model, etc. Default: data
30     OUTPUT_DIR            Output directory for generated files. Default:
31         DATA_DIR/MODEL_NAME
32     GROUND_TRUTH_DIR      Ground truth directory. Default: OUTPUT_DIR-
33         ground-truth
34     TESSDATA_REPO         Tesseract model repo to use (_fast or _best).
35         Default: _best
36     TESSDATA              Path to the directory containing START_MODEL.
37         traineddata
38         (for example tesseract-ocr/tessdata_best).
39         Default: ./usr/share/tessdata
40     MAX_ITERATIONS        Max iterations. Default: 10000
41     EPOCHS                Set max iterations based on the number of lines
42         for training. Default: none
43     DEBUG_INTERVAL        Debug Interval. Default: 0
44     LEARNING_RATE         Learning rate. Default: 0.0001 with START_MODEL,
45         otherwise 0.002
46     NET_SPEC              Network specification (in VGSL) for new model
47         from scratch. Default: [1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96
48         Lrx96 Lfx256 0lc###]
49     FINETUNE_TYPE         Fine-tune Training Type - Impact, Plus, Layer or
50         blank. Default: ''
51     LANG_TYPE             Language Type - Indic, RTL or blank. Default: ''
52     PSM                   Page segmentation mode. Default: 13
53     RANDOM_SEED           Random seed for shuffling of the training data.
54         Default: 0
55     RATIO_TRAIN           Ratio of train / eval training data. Default:
56         0.90
57     TARGET_ERROR_RATE     Stop training if the character error rate (CER
58         in percent) gets below this value. Default: 0.01
59     LOG_FILE              File to copy training output to and read plot
60         figures from. Default: OUTPUT_DIR/training.log

```

Choose training regime

First, decide what kind of training you want.

- Fine-tuning: select (and install) a `START_MODEL`
- From scratch: specify a `NET_SPEC` (see documentation)

Change directory assumptions

To override the default path name requirements, just set the respective variables in the above list:

```
1 make training MODEL_NAME=name-of-the-resulting-model DATA_DIR=/data
   GROUND_TRUTH_DIR=/data/GT
```

If you want to use shell variables to override the make variables (for example because you are running `tesstrain` from a script or other makefile), then you can use the `-e` flag:

```
1 MODEL_NAME=name-of-the-resulting-model DATA_DIR=/data GROUND_TRUTH_DIR=
  /data/GT make -e training
```

Make model files (traineddata)

When the training is finished, it will write a `traineddata` file which can be used for text recognition with Tesseract. Note that this file does not include a dictionary. The `tesseract` executable therefore prints a warning.

It is also possible to create additional `traineddata` files from intermediate training results (the so-called checkpoints). This can even be done while the training is still running. Example:

```
1 # Add MODEL_NAME and OUTPUT_DIR like for the training.
2 make traineddata
```

This will create two directories `tessdata_best` and `tessdata_fast` in `OUTPUT_DIR` with a best (double based) and fast (int based) model for each checkpoint.

It is also possible to create models for selected checkpoints only. Examples:

```
1 # Make traineddata for the checkpoint files of the last three weeks.
2 make traineddata CHECKPOINT_FILES="$(find data/foo -name '*.checkpoint'
   -mtime -21)"
3
4 # Make traineddata for the last two checkpoint files.
5 make traineddata CHECKPOINT_FILES="$(ls -t data/foo/checkpoints/*.
   checkpoint | head -2)"
6
```

```
7 # Make traineddata for all checkpoint files with CER better than 1 %.
8 make traineddata CHECKPOINT_FILES="$(ls data/foo/checkpoints
/*[^1-9]0.*.checkpoint)"
```

Add `MODEL_NAME` and `OUTPUT_DIR` and replace `data/foo` with the output directory if needed.

Plotting CER

Training and Evaluation Character Error Rate (CER) can be plotted using Matplotlib:

```
1 # Make OUTPUT_DIR/MODEL_FILE.plot_*.png
2 make plot
```

All the variables defined above apply, but there is no explicit dependency on `training`.

Still, the target depends on the `LOG_FILE` captured during training (just will not trigger training itself). Besides analysing the log file, this also directly evaluates the trained models (for each checkpoint) on the eval dataset. The latter is also available as an independent target `evaluation`:

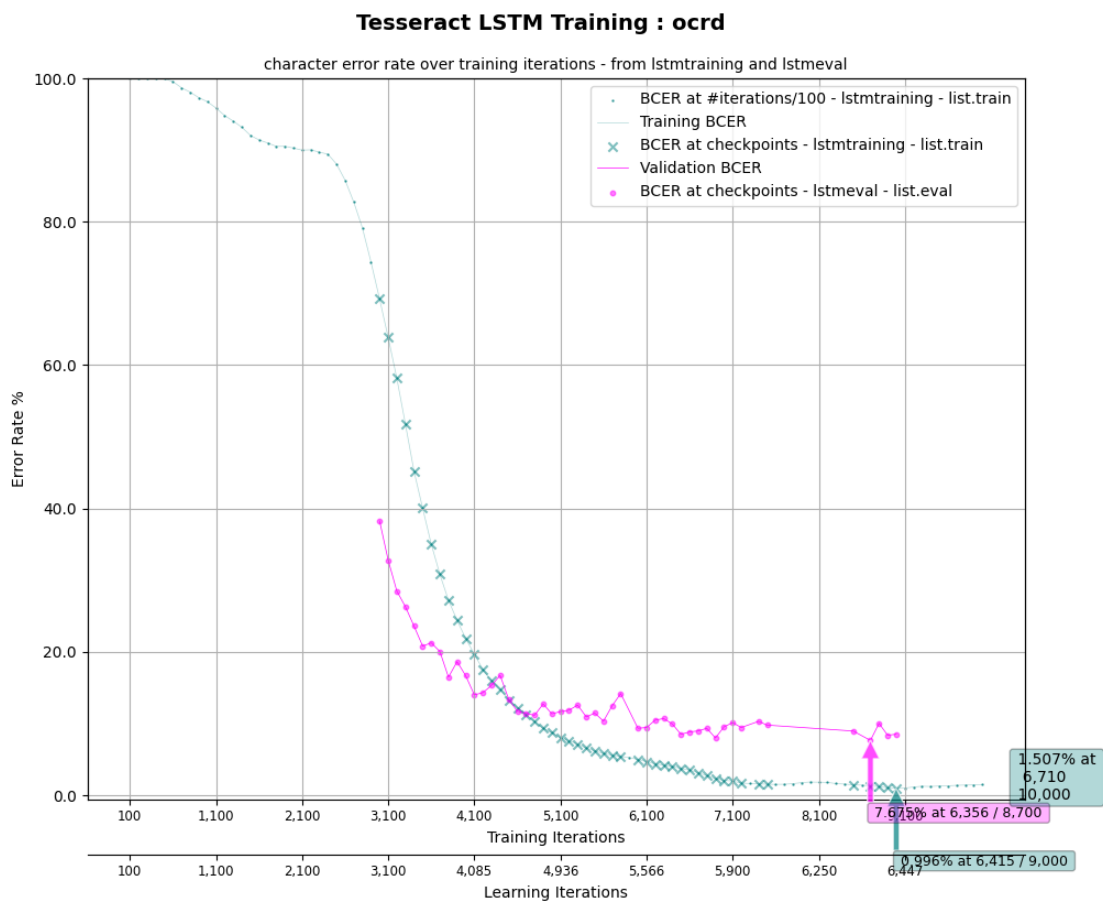
```
1 # Make OUTPUT_DIR/eval/MODEL_FILE*.log
2 make evaluation
```

Plotting can even be done while training is still running, and will depict the training status up to that point. (It can be rerun any time the `LOG_FILE` has changed or new checkpoints written.)

As an example, use the training data provided in `ocrd-testset.zip` to do some training and generate the plots:

```
1 unzip ocrd-testset.zip -d data/ocrd-ground-truth
2 make training MODEL_NAME=ocrd START_MODEL=frk TESSDATA=~/.tessdata_best
  MAX_ITERATIONS=10000 &
3 # Make data/ocrd/ocrd.plot_cer.png and plot_log.png (repeat during/
  after training)
4 make plot MODEL_NAME=ocrd
```

Which should then look like this:



License

Software is provided under the terms of the [Apache 2.0](#) license.

Sample training data provided by Deutsches Textarchiv is in the public domain.