Vom Bild zum Text – praktische OCR für die DH

—

12.05.2021, 15–17 Uhr: Evaluation, Transkription, Training

Evaluation

"Wie gut ist die Texterkennung?"

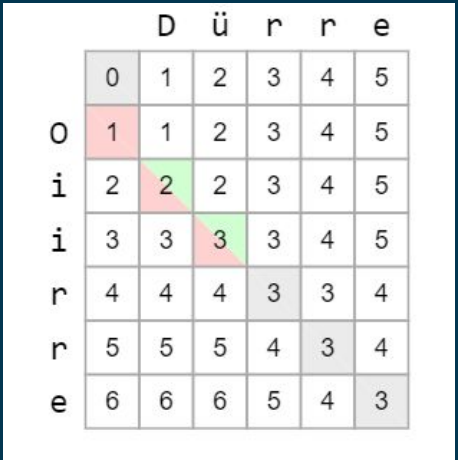
- Was heißt "gut"?
  - Im Vergleich zu Ground Truth?
  - Im Vergleich zur Konfidenz der OCR-Engine?
  - Im Vergleich zu einem Wörterbuch?
- Was heißt "Texterkennung"
  - Wie viele Zeichen sind falsch erkannt?
  - Wie viele Worte sind korrekt?
  - Wie viele Phrasen sind korrekt?
- Was ist mit Sonderfällen?
  - Können historische Schreibweisen als OCR-Fehler gewertet werden?
  - Zeichenerkennung korrekt, aber Segmentierung fehlerhaft?
  - Gewichtung von Fehlern nach Relevanz für Anwendung?

Konventionen

- Error rate = 1 - Accuracy
- Prozentuale Fehlerrate
  - 0 ⇒ Fehlerfrei
  - 1 ⇒ Komplett falsch
  - > 1, < 0: Verarbeitungsfehler, ~~sollte wie 1 gezählt werden~~ **Aussortieren!**
- Fehlerratenberechnungen erfordern meist Ground Truth

Zeichenfehlerrate - Character Error Rate (CER)

- Wie viele Fehler: Variationen der Levenshtein-Distanz
- Wie viele Editieroperationen werden benötigt um von einer Zeichenkette zur anderen zu gelangen, bzw. von der OCR zur GT
- CER = # Editieroperationen / # Zeichen GT (?)



		D	ü	r	r	e
	0	1	2	3	4	5
O	1	1	2	3	4	5
i	2	2	2	3	4	5
i	3	3	3	3	4	5
r	4	4	4	3	3	4
r	5	5	5	4	3	4
e	6	6	6	5	4	3

Verwechslungstabelle

- Liste der häufigsten Verwechslungen von Zeichen(sequenzen) zwischen OCR und GT
- Hilfreich, um systematische Fehler zu finden, wie nicht trainierte Zeichen und Ligaturen oder suboptimale Modellkombinationen

(Beispiele aus Reports von ocreval)

Typische OCR-Fehler

```
      1        0   {A}-{H}
      1        0   {h}-{b}
      1        0   {l}-{i}
      1        0   {n}-{u}
      1        0   {n}-{v}
      1        0   {n}-{y}
      1        0   {t}-{k}
      1        0   {u}-{ü}
      1        0   {v}-{u}
      1        0   {≠}-{-}
      1        0   {□}-{ö}
      1        0   {□}-{u}
```

PUA Ligaturen im GT aber nicht im Codec der OCR

```
Errors   Marked   Correct-Generated
    16        0   {□}-{ch}
    10        0   {ff}-{ff}
     8        0   {□□}-{ſich}
     6        0   {□}-{ſſ}
     4        0   {□}-{ſi}
     4        0   {□}-{ſt}
     4        0   {□}-{ck}
```

Wortfehlerrate = Word Error Rate (WER)

		W	i	r		e	s	s	e	n	,		m	e	i	n		K	i	n	d
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
W	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
i	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
r	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
e	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
e	8	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
n	9	8	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11
	10	9	8	7	6	5	4	3	2	1	1	1	2	3	4	5	6	7	8	9	10
m	11	10	9	8	7	6	5	4	3	2	2	2	1	2	3	4	5	6	7	8	9
e	12	11	10	9	8	7	6	5	4	3	3	3	2	1	2	3	4	5	6	7	8
i	13	12	11	10	9	8	7	6	5	4	4	4	3	2	1	2	3	4	5	6	7
j	14	13	12	11	10	9	8	7	6	5	5	5	4	3	2	2	3	4	5	6	7
n	15	14	13	12	11	10	9	8	7	6	6	6	5	4	3	2	3	4	5	5	6
	16	15	14	13	12	11	10	9	8	7	7	6	6	5	4	3	2	3	4	5	6
K	17	16	15	14	13	12	11	10	9	8	8	7	7	6	5	4	3	2	3	4	5
i	18	17	16	15	14	13	12	11	10	9	9	8	8	7	6	5	4	3	2	3	4
n	19	18	17	16	15	14	13	12	11	10	10	9	9	8	7	6	5	4	3	2	3
d	20	19	18	17	16	15	14	13	12	11	11	10	10	9	8	7	6	5	4	3	2

Anzahl Worte mit CER > 0 / Worte insgesamt

Zeichen: 20

Worte: 4 (Satzzeichen gehören zum Wort)

Zeichenfehler: 2

CER = 2 / 20 = **0.1**

WER = 2 / 4 = **0.5**

WER ist im Allgemeinen größer als CER

Bag of Words (BoW) - WER ohne Reihenfolge

Ich bin klug. Du bist	der doofe Dödel hier

Mit Spaltentrennung:

"Ich bin klug. Du bist der doofe Dödel hier"

Ohne Spaltentrennung:

"Ich bin der doofe klug. Dödel Du bist hier"

Zählen wie viele Worte richtig erkannt wurden, ohne deren Reihenfolge zu beachten

Besonders bei komplexen Layouts aber einfachen Anwendungsszenarien relevant

CER ohne Spaltentrennung: 27/42 = **0.65**

WER ohne Spaltentrennung: 6/9 = **0.66**

BoW ohne Spaltentrennung: **0**

Varianten von CER/WER/BoW

- Andere Behandlung von Ligaturen, Unicode-Normalisierungen u.ä.
- Andere Gewichtung von Operationen (bspw: "vertauschen ist schlimmer als löschen weil schwerer zu bemerken")
- Andere Gewichtung mit Sprachmodell (bspw. "Stoppworte wie 'und', 'oder', 'dass' weniger wichtig als Eigennamen")
- Flexibleres Alignment von GT und OCR für mehr Toleranz gegenüber Fehlern in der Segmentierungs- und Lesereihenfolge

Sampling - Auswählen von Beispielen

- Alle genannten Verfahren erfordern GT als Referenz
- Logistisch unmöglich, GT für alle OCR-Ergebnisse zu produzieren
- Auswahl des Evaluationssets hat großen Einfluss auf die Ergebnisse
- "Repräsentative" Beispiele finden erfordert Domänenwissen
- Bernoulli-Experiment der DFG (zufällige Seite, zufälliger Offset in Buchstaben) nicht zu empfehlen
- Wichtig, keine Trainings- bzw. Validierungsdaten zu verwenden

Evaluation von vorgelagerten Schritten

- Textbasierte Evaluation geht erst nach der OCR
- Segmentierung
  - Intersection over Union: Wie groß ist Überlappung von Segmentierung in GT und OCR im Vergleich zur Fläche insgesamt
  - Darauf basierend: Precision, Recall, F1
- Reading Order
  - Abgleichen der Sequenzen in GT und OCR
- Binarisierung
  - Abgleich von Pixeln oder Histogrammen

GT-freie OCR-Evaluation

- Gegenstand der Forschung :-)
- Konfidenzen und Alternativen der OCR-Engines
  - Wie verlässlich sind die Konfidenzen?
  - Wie findet man plausible Zeichensequenzen?
  - Wie aligniert man Strings inklusive Alternativen?
- Statistische Heuristiken basierend auf Sprachmodellen
  - Wie viele Worte finden sich in einem Lexikon
  - Konfidenzen von Spracherkennungssoftware
- Heuristiken über die Segmentierung
  - Wie viele Regionen enthalten Text, wie viele Rauschen
- Trainierbare Software, die mehr als den Text als Feature nutzt
  - Bildmetadaten, Schrift, Entstehungszeitraum, Bildqualität etc.

Tools

Software	Metriken	Bemerkungen
dinglehopper	CER, WER, Diff	OCR-D Interface
ocrd-cor-asv-ann	CER, Varianz, Konfusionen	OCR-D Interface, Support für OCR-D Transkriptionslevel
ocrevalUAtion	CER, WER, BoW, Diff	Unterstützt viele Formate
ocreval	CER, WER, Konfusionen, Stoppworte	Seit mehr als 25 Jahren in Entwicklung
PRImA TextEval	CER, WER, BoW, Flexible CER	Flexible CER für Robustheit gegen Segmentierungsfehler; Windows-only; keine Freie Software
PRImA LayoutEval	Merge, Split, Miss etc., Lesereihenfolge, Visualisierung	Feingranulare Steuerung (Profile); Windows-only; keine Freie Software

Demo

https://github.com/kba/vdhd-2021-05-12

Training

Einführung

Christian Reul

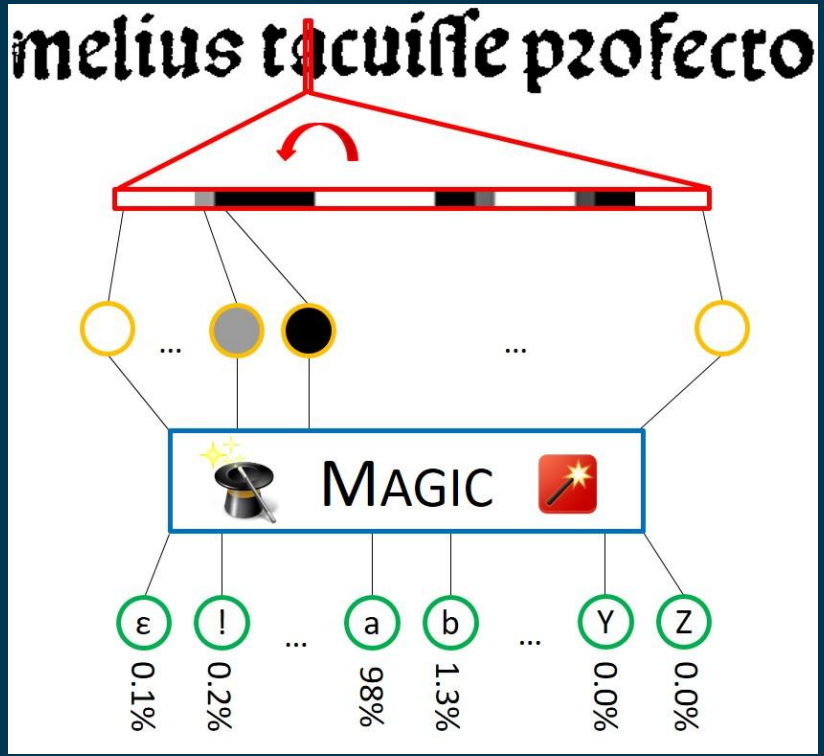
Training? Modell? Lernen?

- Training
  - Duden: “planmäßige Durchführung eines Programms von vielfältigen Übungen zur Ausbildung von Können, Stärkung der Kondition und Steigerung der Leistungsfähigkeit”
  - Allgemeiner: etwas (wiederholt) durchführen, um sich zu verbessern
  - Aber… wie sieht das bei der OCR aus? Was wird wiederholt, was verbessert?
- Und… was ist eigentlich ein “Modell”? Was “lernt” es (und wie)?
- Beispiel OCR (Rückblick Einführungsveranstaltung):
  - Modelle extrahieren Text aus Textzeilenbildern
  - Modell: Input/Output-Relation:
    - Input: Bild (?)
    - Output: Text (?)

Er wird eifrig geſammelt.

Grundprinzip der zeilenweisen OCR Erkennung

- “Streifchenweise” Verarbeitung
- Input: Pixelwerte eines 1px breiten Streifens
  - Schwarz: 0, weiß: 1
  - Schwarz: 0, etwas weniger schwarz: 0.01, ..., mittleres Grau 0.5, ..., weiß: 1
  - ...
- Output: Wahrscheinlichkeitsverteilung über Codec (Zeichensatz) des Modells
- Zunächst Streifen für Streifen, anschließend Kombination der Ergebnisse
- “Modell” besteht aus
  - Spezifikation von Input und Output
  - “Anweisungen” für die Umwandlung von Input nach Output



Modellparameter

- Anzahl der Eingabeknoten (Werte einzelner Pixel):
  - Nach Initialisierung unveränderlich
  - Abhängig von Höhe der Bildzeile → Skalierung
  - Calamari Default: 48
- Anzahl Ausgabeknoten = Größe des Codecs (bekannte Zeichen)
  - Kann durch weitere Trainingsprozesse reduziert/erweitert werden
  - Erkennung unbekannter Zeichen ist ausgeschlossen
- “Gewichte”
  - Beschreiben den Weg von Input (Pixelwerte) nach Output (Wahrscheinlichkeits-verteilung für Codec)
  - Können/müssen trainiert bzw. gelernt werden
  - Anzahl abhängig von “Netzstruktur”
  - Calamari Default: ca. 1,5mio

Zusammenfassung

- Neben den (während des Trainings) fixen Ein- und Ausgabeknoten enthält ein OCR Modell zahlreiche “Gewichte”
- Diese Gewichte bestimmen die Input/Output Relation:
  Bild (Pixel(streifen)werte) → Text (Wahrscheinlichkeitsvert. über Codec)
- Das Training erfolgt iterativ, also Schritt für Schritt
- Jeder Schritt besteht aus zwei Teilschritten:
  - Bestimmung der aktuellen Qualität (Loss): Erkennung einer Zeile durch das aktuelle Modell (= mit der aktuellen Gewichtskonfiguration) und Abgleich mit der GT
  - Anpassung der Gewichte: Untersuchung der “anliegenden” Gewichtskonfigurationen und Auswählen der besten

Transkription

Andreas Büttner

Transkriptionsumgebungen/GT-Produktion

- LAREX (@OCR4all)
- eScriptorium
- Origami
- nashi
- neat (@qurator-spk)
- ocropus-gtedit
- kraken ketos
- nw-page-editor
- Aletheia
- Transkribus
- ...

Iteratives Training

- falls bereits passende Modelle existieren, ist es schneller, OCR-Text zu korrigieren, als neu zu beginnen
- Zyklus: Prediction, Teilkorrektur, Training, Prediction, Teilkorrektur, …
- Aufwand und Vorgehen abhängig von Trainingszeit und -ressourcen, Komplexität der Texteingabe (Sonderzeichen etc.), Ziel der Transkription (Edition?, Textmining?)
- nach Möglichkeit Verteilung der GT über das ganze Buch, Fokus auf spezifische Schwachstellen des Ausgangsmodells, z. B. Ziffern, Überschriften, Kursive

Transkriptionsrichtlinien, Normalisierung, Codec

- Ziel: möglichst allgemeines oder möglichst spezifisches Modell?
- Codec: so klein wie möglich, so groß wie nötig
- Normalisierungsmöglichkeiten:
  - Leerzeichen
  - Anführungszeichen: “ ” »«
  - Halb-/Geviertstriche
  - Umlaute
  - s-ſ, I-J, &-et
  - konsonantische/vokalische Ligaturen
- diverse typographische Details können später regelbasiert rekonstruiert werden
- Zeichen für historische Drucke: Medieval Unicode Font Initiative
- one-to-many vermeiden (s-s/ſ, I-I/J etc.)
- Abkürzungsauflösung möglich (Kompromisse bei der Genauigkeit der OCR)
- Beispiele für Transkriptionsrichtlinien: OCR-D: Level 1-3

Calamari

Andreas Büttner

Calamari

- Open Source OCR Engine basierend auf dem Tensorflow-Framework für GPU-beschleunigte neuronale Netze
- keine eigenen Werkzeuge für Segmentierung, Transkription, Formatkonvertierung etc., Fokus auf Training und Erkennung
- https://github.com/Calamari-OCR/calamari
- Wick, C., Reul, C., Puppe, F.: Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition. Digital Humanities Quarterly 14(1) (2020). Preprint on arXiv.
- detaillierte Analyse und Optimierung: Liebl, B. and Burghardt, M., “On the Accuracy of CRNNs for Line-Based OCR: A Multi-Parameter Evaluation”, preprint, 2020.

Dataset

- default: **Plaintext**: 0001.png, 0001.gt.txt (Zeilenbilder, vgl. Ocropus)
- **PageXML**: 0001.png, 0001.xml (Seitenbilder, korrespondierendes PAGE)
- **Abbyy XML**: 0001.png, 0001.abbyy.xml
- **HDF5**: alle Daten in einer Datei
- bei sehr großen Datensätzen/wenig RAM: train.preload False!

Pretraining, Modelle

- Modelle können als Ausgangspunkt für eigenes Training genutzt werden (Warm-Starting)
- besonders bei geringer Menge an eigener Ground Truth hilfreich, selbst wenn die Schriftarten sich nicht entsprechen (Reul, Wick, Springmann, Puppe: Transfer Learning for OCRopus Model Training on Early Printed Books)
- whitelisting mit “codec.include” oder “codec.keep_loaded” verhindert, dass Zeichen aus dem Codec gelöscht werden, die nicht in den Trainingsdaten vorhanden sind

Modelle zum Download:

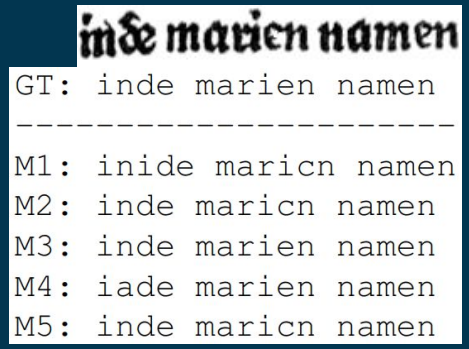
- https://github.com/Calamari-OCR/calamari_models
- https://github.com/Calamari-OCR/calamari_models_experimental
- https://qurator-data.de/calamari-models/
- https://github.com/poke1024/origami_models

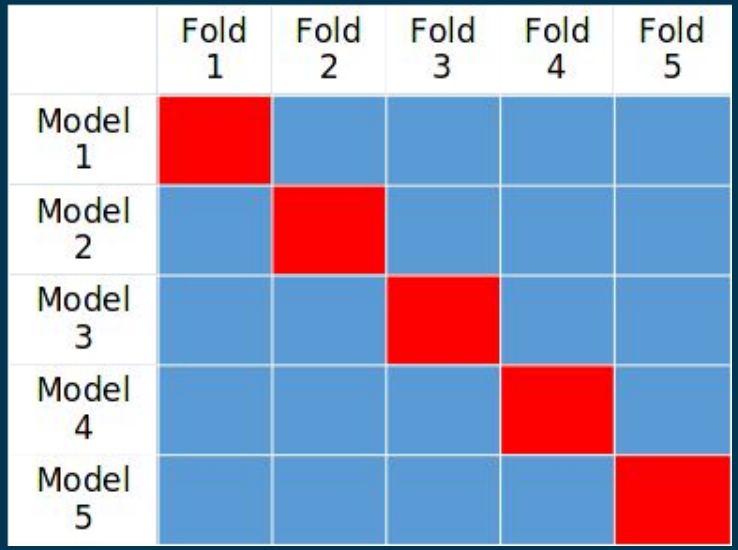
Early Stopping, Validierung

- Wann ist das Training fertig? Wenn das Modell in einer definierten Anzahl an Schritten nicht mehr besser wird.
- Daten zur Validierung:
  - Trainingsdaten (Overfitting!)
  - zufällig selektierte und separierte Teilmenge der Trainingsdaten
  - eigenes Validierungsset (für konsistente Evaluation zu bevorzugen)
- Resultat der Evaluation auf Validierungsdaten gibt ersten Hinweis auf zu erwartende Genauigkeit der Erkennung auf unbekannten Daten

Voting, Cross Fold Training

- Voting: Kombination der Ergebnisse mehrerer Modelle
- Konfidenzwerte der Modelle bestimmen Einfluss auf das Ergebnis der Abstimmung
- Training mehrerer Modelle mit jeweils unterschiedlicher Teilmenge als Validierungsdaten (rot) und Trainingsdaten (blau)





Data Augmentation

- Vergrößerung der Menge an Trainingsdaten durch (zufällige) Manipulation der Bilddaten
- Implementierung ursprünglich in OCRopus3: OCRodeg
- Vorgehen: Training auf orig.+augm., danach nur auf Originaldaten

Live-Demo

Das Jupyter-Notebook mit den Beispiel ist auf Github im Repo andbue/calamari_demo verfügbar. Falls keine eigene CUDA-fähige GPU vorhanden ist, kann Google Colaboratory verwendet werden.

Tesseract

Jan Kamlah

Tesstrain

Save some stress and take the train

Tesseract https://github.com/tesseract-ocr/tesseract

- seit der Version 4.0 kann Tesseract Texte mittels neuronaler Netze (LSTM-Modelle) erkennen
- mehr als 100 Sprachen- und 35 Skript/Schriftsystemmodelle
- zwei Modellvarianten: Fast (Production) und Best (Training)
- Größe der Modelle: ca. ½ MB (Fast) - ca. 3 MB (Best) (ohne Wörterbuch)
- Erkennung und Training (derzeit) nur auf der CPU
- Training per komplexen CLI-Befehlen (hoher Einarbeitungsaufwand!)
  https://tesseract-ocr.github.io/tessdoc/tess4/TrainingTesseract-4.00.html

Modelle zum Download

- https://github.com/tesseract-ocr/tessdata_best
- https://github.com/tesseract-ocr/tessdata_fast
- https://ub-backup.bib.uni-mannheim.de/~stweil/tesstrain/frak2021/
  (historische Schriftarten aus dem deutschsprachigen Raum)

Tesstrain – choo choo! https://github.com/tesseract-ocr/tesstrain

Makefile-basierte CLI-Programm um Trainings mit Tesseract anwenderfreundlich zu starten

- Installation von Softwareabhängigkeiten
- Abnahme von vorbereitenden Arbeitsschritten
  - Unicodenormalisierung (NFC, NFKC..)
  - Shuffle (Veränderung der Trainingsreihenfolge)
  - ...
- Vereinfachung des Aufrufs des Trainings
- sinnvoll voreingestellte Trainingsparameter mit Tuning-Möglichkeit für fortgeschrittene Anwender

Finetuning or from scratch

- Finetuning (Warm Starting/Nachtraining/Werksspezifisches Training)
  - Geeignetes Basismodell vorhanden
    - Erweiterung des Basismodells (bsp. astronomische Zeichen)
    - Verbesserung der Texterkenunngsqualität des ganzen Basismodells
    - Korrektur/Modifikation des Basismodells (bsp. OCR-D Level 2 konform)
    - Schaffung eines werksspezifischen Modells
  - Geringe Anzahl an GT-Daten (ein „paar hundert“ Zeilen)
  - Trainingsdauer: 3–4 Stunden (Büro PC)

- Wann lohnt sich „from scratch“?
  - Kein geeignetes Basismodell vorhanden
  - Umfangreiche GT-Daten
  - Trainingsdauer: 2–3 Wochen (Büro PC)

Ground Truth Tips

- OCR-D Level 2 (nahe am Originaltext, ohne zu viel Transkriptionsaufwand zu erzeugen)
- Normalisierung der Ground Truth
  - I/J
  - s/ſ
  - Leerzeichen
  - …
- *Optional:* Vervielfachung der Ground Truth durch Data Augmentation
- *Nachtraining:* Stabilisierung des Trainingsverhalten durch Vermischung der Ground Truth, die für das Basismodell verwendet wurde, mit der neuen Ground Truth

make training

- Starten des Trainings aus dem „tesstrain“-Hauptordner
- Daten werden per default in den „data“-Unterordner gespeichert
- Tesseract erzeugt Checkpoints mit den besten Zwischenergebnissen

Relevante Parameter

- MODEL_NAME (Ausgabename des Modells)
- GROUND_TRUTH_DIR (Pfad zur Ground Truth)
- MAX_ITERATIONS / EPOCHS (Anzahl der Lerndurchläufe)
- Empfehlungen: 30 < Epochen < 120
- START_MODEL (Nachtraining) (Basismodell)
- TESSDATA (Nachtraining) (Pfad zum Basismodell)

make training

```
# make training
make training MODEL_NAME=vDHd START_MODEL=frak2021_1.109_651416_3009600
TESSDATA=/home/stweil/src/github/tesseract-ocr/tesstrain/data/frak2021/tessdata_best
GROUND_TRUTH_DIR=/home/jkamlah/vDHd/ EPOCHS=30

# Erzeugung von lstmf und box Dateien
+ tesseract /home/jkamlah/vDHd/Number/010033.png /home/jkamlah/vDHd/Number/010033 --psm 13 lstm.train
Tesseract Open Source OCR Engine v5.0.0-alpha-20210401-70-gce05b with Leptonica
PYTHONIOENCODING=utf-8 python3 generate_line_box.py -i "/home/jkamlah/vDHd/Number/010030.png" -t
"/home/jkamlah/vDHd/Number/010030.gt.txt" > "/home/jkamlah/vDHd/Number/010030.box"

# Aufruf des Tesseract-Trainings
lstmtraining \
  --debug_interval 0 \
  --traineddata data/vDHd/vDHd.traineddata \
  --old_traineddata /home/stweil/src/github/tesseract-ocr/tesstrain/data/frak2021/tessdata_best/
…

# Traingsfortschritte
2 Percent improvement time=110, best error was 100 @ 0
At iteration 110/300/300, Mean rms=0.284000%, delta=0.509000%, char train=1.365000%,
word train=4.391000%, skip ratio=0.000000%, New best char error = 1.365000 wrote best model:
data/vDHd/checkpoints/vDHd_1.365000_110_300.checkpoint wrote checkpoint.
```

How to read the output?

At iteration 14615/695400/698614, Mean rms=0.158%, delta=0.295%,
char train=1.882%, word train=2.285%, skip ratio=0.4%, wrote checkpoint.

- 14615 : learning_iteration – Anzahl von Iterationen, die einen nicht-null Delta-Fehler haben und so signifikant zum Training beitragen.
- 695400 : training_iteration – Anzahl von Iterationen, die in den Trainingsprozess mit eingeflossen sind.
  Typische Fehlerquellen: Leeres Bild, mehrzeilige Ground Truth, Zeilenbilder mit vertikaler Orientierung..
- 698614 : sample_iteration – Anzahl von Iterationen, wie viele Trainingsdaten in den Trainingsprozess hätten einfließen sollen

Checkpointname

- <model_base>_<char_error>_<learning_iteration>_<training_iteration>.checkpoint
- Beendigung des Trainings nach Erreichen der vorgegebenen Iterationen/Epochen oder der gewünschten Genauigkeit

make traineddata

- Umwandlung der Checkpoints in best und fast Modelle
- Umwandlung aller Checkpoints

```
# make traineddata
make traineddata MODEL_NAME=vDHd CHECKPOINT_FILES="$(find data/vDHd -name '*.checkpoint')"

# Checkpoints werden zu Modellen umgewandelt
lstmtraining \
          --stop_training \
          --continue_from data/vDHd/checkpoints/vDHd_1.529000_78_200.checkpoint \
          --traineddata data/vDHd/vDHd.traineddata \
          --model_output data/vDHd/tessdata_best/vDHd_1.529000_78_200.traineddata
Loaded file data/vDHd/checkpoints/vDHd_1.529000_78_200.checkpoint, unpacking...
```

- Optional: Anreicherung der Modelle mit Wörterbuch, Bi-Gramme, Muster für Zahlen und Satzzeichen

Is the last checkpoint really the best?

- **Nein**. Evaluation der Modelle ist nötig!
  Ein paar Seiten nicht in die Ground Truth mit einbringen, sondern als Evaluationsset zurücklegen.

  Beispiel (Werksspezfisches Training)

- AP_0.945000_32263_145400: 99.15
- AP_0.903000_32271_145500: 99.15
- AP_1.630000_14032_50500: 99.13
- AP_1.025000_27971_120900: 99.13
- AP_1.096000_27958_120800: 99.13
- AP_0.834000_33156_151000: 99.11
- AP_0.803000_33164_151100: 99.11
- AP_1.188000_20381_80600: 99.08
- AP_0.711000_43778_218800: 99.08
- AP_1.143000_24254_100800: 99.05
- AP_0.772000_38090_181200: 99.01
- AP_0.687000_45550_231500: 99.01
- frak2021: 97.56

- Muss “blind” ein Modell ausgewählt werden, ist jedoch das zuletzt erzeugte zu empfehlen.

What‘s coming next from UB Mannheim?

Ausblick auf OCR-D Phase 3

- Unterstützung von werksspezifischem Training für Endanwender
  - Verbesserung der Erkennungsqualität
  - Erkennung zusätzlicher Zeichen
- Trainingsprozesse noch anwenderfreundlicher machen
- Performanceverbesserung