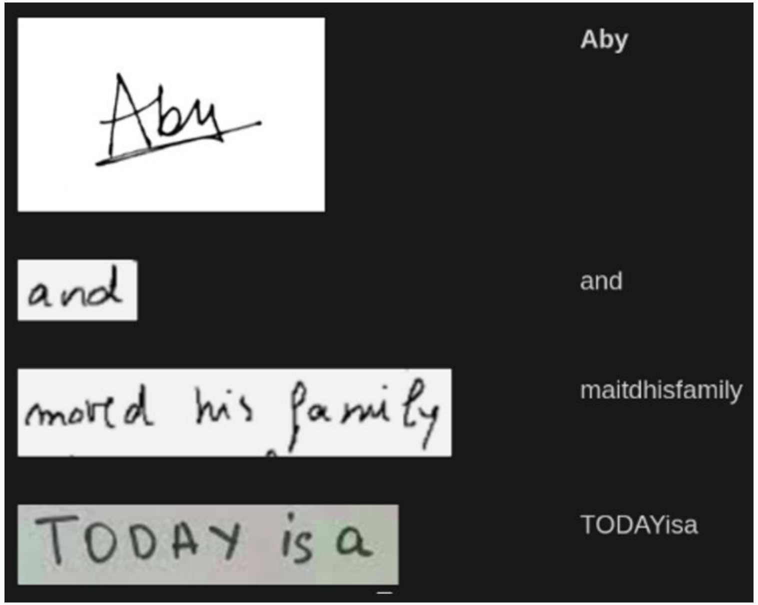
Improving OCR Pipeline @Decimal Point Analytics

Issue at hand.

- Current pipeline not end-to-end
  - Detection + Recognition
- Issue with recognition pipeline??
  - **Unable to reproduce white-spaces**
  - Unable to recognize special characters
  - Unable to parse numbers in financial amounts and hyperlinks



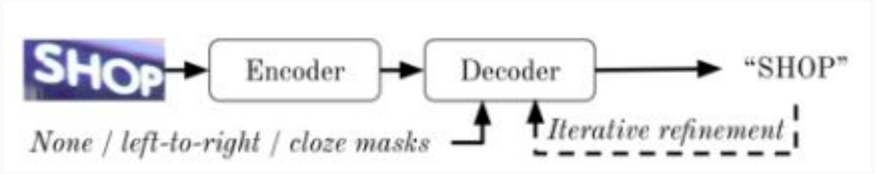
Approach #1: Improve detection to one word

- Baselined various word detector models:
  - WordNN, DBNet, CRAFT, Textfusenet, EasyOCR, Donut, LMV3, TrOCR
  - Markers: inference time and accuracy
  - Recurrent issues?
    - Missing out of small words
    - Merging of 2 or more words
- Dataset reasoning
  - Impact of training data
  - Custom training
    - PS: annotation would have taken my entire summer

Apart for this, I reviewed literature on text detection and followed current works exhaustively.

Approach #2: Improve recognition phase

- What is PARSEQ?
    - Literature, set-up, training.
- Reviewed current state of text recognizers
- Impact of dataset?



- Issue at hand? How to create such a large custom dataset

"They are trained by enforcing an autoregressive (AR) constraint on the language context where future tokens are conditioned on past tokens but not the other way around"

Issues with dataset?

- Current checkpoint trained on **single word images**
- Lack of special characters
- Need to create a dataset which is rich in white-spaces and maybe other special characters; especially those from financial domain

Creating custom dataset

1. PyMuPDF;
   a. Input : PDF
   b. Output : Images of individual words; corresponding ground-truth text
2. Tiger-Synth;
   a. Input : Corpus of text
   b. Output : Images with text as in input corpus

- Created **char_set;** bundled in **lmdb** format; set to **train**

Issues with dataset v0

- PyMuPDF: Random concat of words -> Model doesn't generalize
- Tiger-Synth: Invalid words -> Poisons language model
  - Can output ***par&eq, hell0, `good mourning`, `good dye`***

41.65%

Accuracy of model on test-set that was trained on v0 (136k training samples)

PS: Trained 3 more different models with minor tweaks but negligible trade-off gain

Dataset v1: `Semantic dataset`

- Words occur in order of natural language
- Individual words are valid english words

Outcome?

- **Labels coming from spoken english gave good results**
- **White-spaces recognized**
- **Rest; unsatisfactory**

Ignoring punctuation; accuracy:

88%

Hypothesis #1: Special characters under-representation

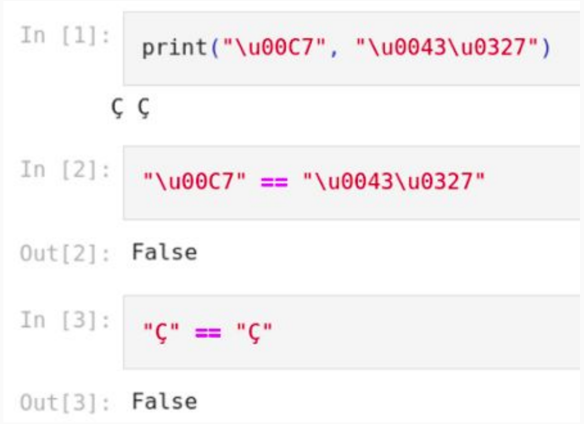
(PS: Hypothesis #2: overfitting was ruled out after experimentation)
(PS: Hypothesis #3: shallow model architecture was ruled out)

- 101 special characters
- Frequency of SC: **10.34%** (~ 100-88)
- 62.58% of labels had one or more SC
- 94/101 SCs are starved; their occurrence < 500; (out of total 10M characters)

Failure Categories:

- Confusing chars:
  `* vs *, ± vs +, ‵´″ vs "`
- Non-english Latin chars:
  alpha, beta, gamma, …

Next step?
Unicode normalization



```
In [1]:  print("\u00C7", "\u0043\u0327")

         Ç Ç

In [2]:  "\u00C7" == "\u0043\u0327"

Out[2]:  False

In [3]:  "Ç" == "Ç"

Out[3]:  False
```

99.23%

Accuracy of model on test-set that was trained on v1.2 + ***partial unicode normalization***

PS: Tested on different population of dataset and save similar results

Refining edge cases

- Email IDs and hyperlinks
- Dates
- Function names
- HTML code and other programming languages
- Tags and mentions
- Mathematical equations
- IPs and amounts

Parallely, created a dataset with 1.4M labels (10x the original) using;

- **Gov docs**
- **NCERT books**
- **Research papers**
    - **Trade-off was however found unsatisfactory**