SCHOOL FOR ADVANCED STUDIES LUCCA

Data Science & Statistical Learning | II Level Master

Text Mining and Natural Language Processing

UNIVERSITÀ DEGLI STUDI FIRENZE

LayoutReader:

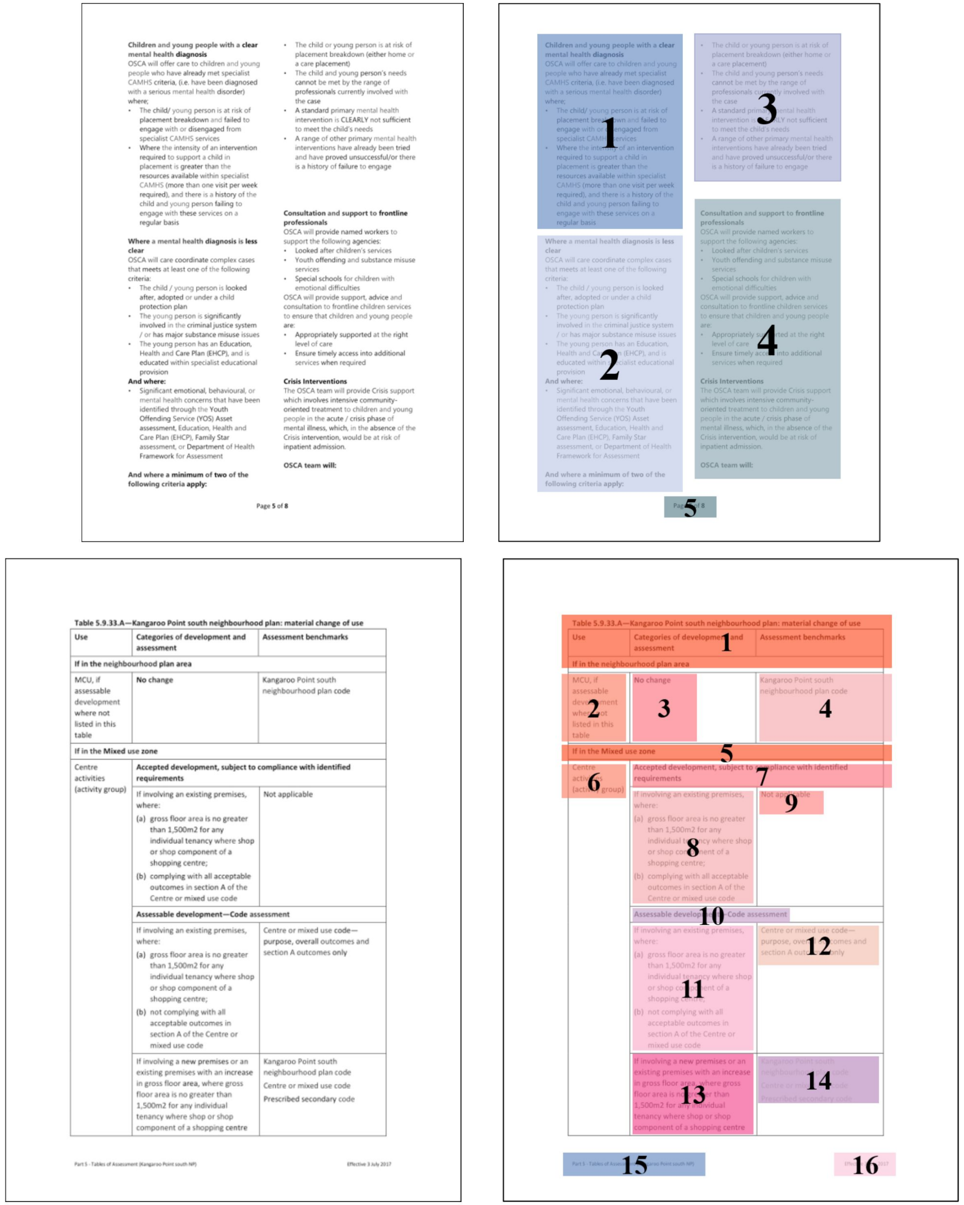
Pre-training of Text and Layout for Reading Order Detection

Prof. *Simone Marinai*

Gianmarco Santoro

29/02/2024

- **Reading order detection**:
  - **Understand visually-rich docs**, e.g. receipts or forms
  - Capture **word sequence**, which can be naturally **comprehended by human** readers
- Current methods directly use results from **Optical Character Recognition** (OCR) engines, which **arrange recognized tokens or text lines in a top-to-bottom and left-to-right way**
  - This **heuristic is not optimal for** certain doc types, such as **multi-column templates, forms** and **invoices**
- Incorrect reading order will lead to unacceptable results for doc understanding tasks as info extraction from invoices
- **Example** in images: coloured **areas show paragraph-level reading order**



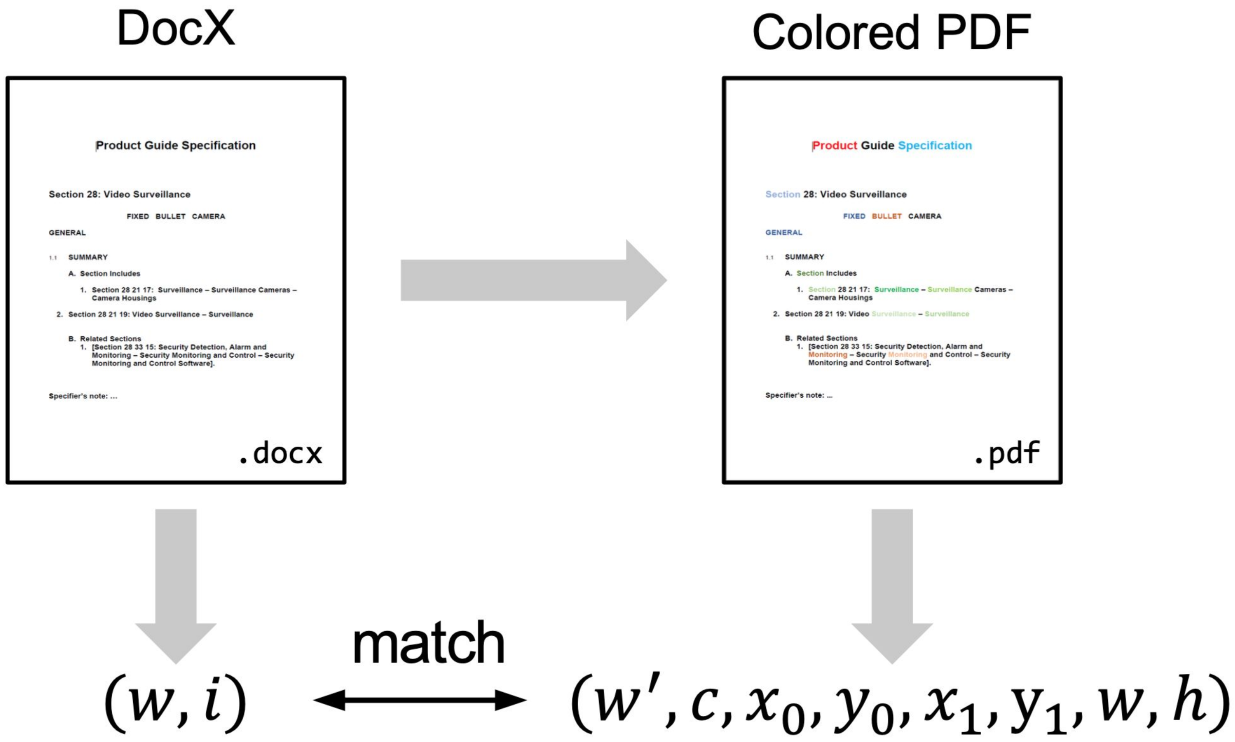
Images source: Paper*

- No existing work took advantage of advanced **DL models**, since it is **too laborious to annotate a large enough dataset**
- Automatically construct **ReadingBank,** first large-scale benchmark for reading order detection tasks:
  - **Benchmark dataset** that contains reading order, text and layout info for **500,000** images covering a **wide doc types**
  - Proposed method obtains high-quality **reading order annotations with automated metadata extraction**
- Microsoft **WORD documents** used, since:
  - Wide variety of templates **available** on internet
  - **DocX** files **format** are used as **reading order information is embedded in XML metadata**
  - **Converted** into **PDF** so that **2D bounding box of each word can be easily extracted** using any PDF parser
- Carefully designed coloring scheme is applied to align text in XML metadata with bounding boxes in PDFs

- Proposed **LayoutReader**:
  - Novel **reading order detection model** in which **seq2seq** is used by **encoding text and layout info** and **generating index sequence in reading order**
  - Studies on input modalities show that both **text and layout info impact performance**
  - **Performs almost perfectly**, improving open-source and commercial OCRs in ordering text lines

- **Reading order** detection **task**
    - Refers to **extract the natural reading sequence from document images (well-organized readable word sequence)**:
        - **Most OCRs fail to provide proper reading order** due to various formats, e.g. tables or multiple columns
    - Specifically, given a visually-rich **document image** $\mathcal{D}$, acquire discrete **token set** $\{t_1, t_2, t_3, ...\}$ where each **token** t_i consists of a **word** w_i and its bounding **box coordinates** $(x_0^i, y_0^i, x_1^i, y_1^i)$, left-top corner and right-bottom corner
    - Equipped with textual and layout info of tokens in document image, **intention is to sort tokens into reading order**

- **ReadingBank** includes two parts:
  - **Word sequence,** denoted as **Reading Sequence** that **is extracted from DocX files**
  - **Corresponding bounding boxes**, **extracted from PDFs** which are generated from DocX files
- Proposed **coloring scheme to solve word duplication** when **matching each word and its bounding box**



- Building pipeline of ReadingBank: where (w, i) is **pair of word** and its **appearance index,** $(w', c, x_0, y_0, x_1, y_1, w, h)$ is **word color** and **layout info**

Image source: Paper*

- DocX format:
  - Have been **crawled from internet** considering public domain license
  - Using language detection API with high confidence to **filter non-English** or bilingual docs, focusing on English ones
  - Only kept pages with **more than 50 words** to guarantee enough info each page. From a total of **210,000 WORD docs**, **500,000 pages have been randomly selected** to build dataset
- **Reading order** in ReadingBank **refers to order of words in DocX files**:
  - DocX file has **XML code with word sequence**, **extracted from open source *python-docx***
    - This tool also enables to **change words' color** for layout alignment

Steps:

- **Extract paragraphs and tables** sequentially from parsing result
- Traverse **paragraphs line by line and tables cell by cell** and obtain word sequence in DocX file
- Denoted sequence as $[w_1, w_2, ..., w_n]$, where n is the number of words in this doc. **Obtained sequence is reading order without layout info and is denoted as Reading Sequence**
- Align bounding box to each word in this sequence [see next slide]

- Same **word may appear multiple times**: it's necessary to **solve duplication** when **assigning coordinates** to **each word**:
  - **Each word has given an extra label** indicating its appearance index:
    - For example, given a sequence ***[the, car, hits, the, bus]***, extra labels should be **[0, 0, 0, 1, 0]** since there are two "the"s. In this way, each **pair of word and its appearance index is unique** and can be used as **key** when **assigning location coordinates**
- The coloring scheme to show keys in DocX file without changing original layout pattern is:
  - **Map appearance index to RGB colors** through **C**: **N** → **RGB** and **color words accordingly**. To eliminate interference from original word color, first color all words into black
- Where:
  - i: **appearance index** of given word
  - &: **bitwise-and** operation
  - C: **mapping function**

$$r = i\&0\text{x}110000$$
$$g = i\&0\text{x}001100$$
$$b = i\&0\text{x}000011$$
$$\mathcal{C}(i) = (\mathbf{R}: r, \mathbf{G}: g, \mathbf{B}: b)$$

- Location of each word in DocX files is not fixed
- Used PDF files produced by colored DocX files to **extract layout info**:
  - Adopted ***PDF Metamorphosis.Net*** to **convert DocX files to PDF**
  - Used open source **tool *MuPDF5*** as **PDF parser**
- **Extracted words, bounding box coordinates, word color from PDF**
- Since **mapping function C is a one-to-one correspondence**, easily get appearance index by using coloring scheme

- It possible to build a **one-to-one match between Reading Sequence and PDF layout info**, where:
  - w, w': are **word** in DocX and PDF, respectively
  - i: is appearance **index** of w
  - c: is **word color recognized by PDF** parser
  - x_0, y_0, x_1, y_1: are left-top and right-bottom **coordinates**
  - W, H: are width and height of page for future studies

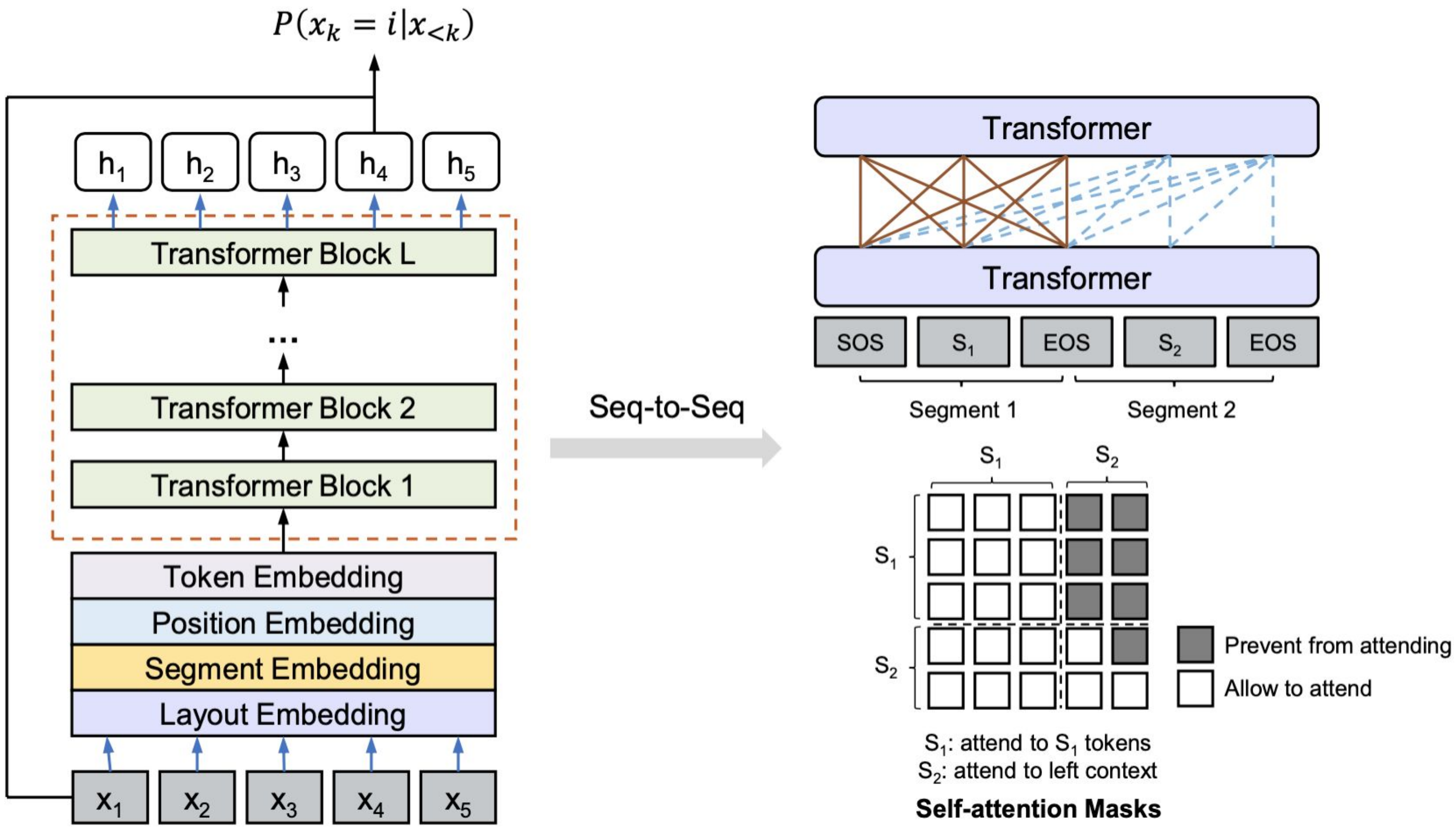
$$(w, i) \leftrightarrow (w', c, x_0, y_0, x_1, y_1, W, H)$$
$$\text{subject to } w = w'; c = \mathcal{C}(i)$$

- **ReadingBank** consists of 500,000 doc-pages including **images, sequence of words and coordinates in reading order**
- Dataset divided by ratio **8:1:1** for training, validation and testing
- **Average word number** and **average sentence-level BLEU score** are reported:
  - BLEU score is calculated for left-to-right and top-to-bottom order using ground truth reading order as reference, so as **to measure difficulty of training samples**
- So, it's assumed that ReadingBank will **not suffer from data unbalance** during pre-training or fine-tuning

Split	**#Word Avg.**	**Avg. BLEU**
Train	196.38	0.6974
Validation	196.02	0.6974
Test	196.55	0.6972
All	196.36	0.6974

Image source: Paper*

- **LayoutReader** is a sequence-to-sequence **model using both textual and layout info**
- **Leverage of layout-aware language model *LayoutLM*** as **encoder** and modify generation step in encoder-decoder structure to predict indices in source segment
- Encoding stage: **packs pair** of **source-target segments** into a contiguous **input sequence** of *LayoutLM* and carefully **designs self-attention mask** to control visibility between tokens



- LayoutReader **allows tokens** in source segment **to attend to each other** while **preventing tokens in target segment** from attending **to rightward context**
- If **1** means **allowing** and **0** means **preventing**, the detail of the mask M is as follows: $M_{i,j} = \begin{cases} 1, & \text{if } i < j \text{ or } i,j \in \text{src} \\ 0, & \text{otherwise} \end{cases}$
- Where: i, j are **indices** in packed input sequence, so they may be from source or target segments
- $i, j \in src$ means both **tokens are from source segment**

Image source: Paper*

- **Decoding stage**: **prediction candidates probability** is calculated as follows:

$$\mathcal{P}(x_k = i|x_{<k}) = \frac{\exp\left(e_i^T h_k + b_k\right)}{\sum_j \exp\left(e_j^T h_k + b_k\right)}$$

- Where:
  - i: **index** in **source** segment
  - e_i, e_j: i-*th*, j-*th* **input embeddings** of source segment
  - h_k: **hidden states** at k-*th* **time step**
  - b_k: **bias** at k-*th* time step

- **Implementation Details**: built upon *Hugging-Face Transformers*, LayoutReader is implemented with *s2s-ft toolkit* from repository of *Dong et al.* (2019). Pre-trained models used are in their base version
  - Used 4 *Tesla* V100 GPUs with batch size of 4 per GPU during training. The number of training epochs is 3 and training process takes approximately 6 hours. Optimized models with *AdamW* optimizer. Initial learning rate is 7×10^{-5} with 500 warm-up steps

- Designed **experiments** for LayoutReader on ReadingBank:
    1. **Reading order** detection
    2. **Input order** study
    3. **OCR** engines adoption
    4. **Real-world example**

- **Comparative** methods:
    - LayoutReader **considers both text and layout** info with **multi-modal encoder** *LayoutLM*
    - To study role of each modality, designed two comparative models:
        - LayoutReader, **text only**: **replace *LayoutLM* with textual language model**, **with *BERT* and *UniLM***, predicting **reading order only through textual info**
        - LayoutReader, **layout only**: **removed token embeddings in *LayoutLM***. Token embeddings are vital for Transformer to extract textual information. After removing these embeddings, LayoutReader **only considers the 1D and 2D positional layout info**
    - **Heuristic Method**, as baseline: this method refers to **sorting words from left to right and from top to bottom**

- **Evaluation Metrics**:
  - **Average Page-level BLEU score** is widely used in sequence generation, since it measure **overlaps between hypothesis and reference**, referring to micro-average precision within a page
  - **Average Relative Distance** (ARD): ARD score is proposed to evaluate difference between re-ordered sequences. It measures **relative distance between common elements in different sequence**. Since re-ordered sequence is generated, ARD allows element omission but adds a punishment for it. Given a sequence $A = [e_1, e_2, ..., e_n]$ and its generated re-ordered sequence $B = [e_{i1}, e_{i2}, ..., e_{im}]$, where $\{i_1, i_2, ..., i_m\} \subseteq \{1, 2, ..., n\}$, ARD score is:

$$s(e_k, B) = \begin{cases} |k - I(e_k, B)|, & \text{if } e_k \in B \\ n, & \text{otherwise} \end{cases}$$

$$\text{ARD}(A, B) = \frac{1}{n} \sum_{e_k \in A} s(e_k, B)$$

- Where: e_k is *k-th* **element in sequence** A; $I(e_k, B)$ is **index** of e_k in sequence B; n is **length of sequence** A

- Train models with left-to-right and top-to-bottom ordered inputs and report evaluation **results on test set, in table**
- Results show that **LayoutReader is** superior and achieves **SOTA** results compared with other baselines:
    - It improves average page-level BLEU by 0.2847 and decreases ARD by 6.71
    - **Even removing some of input modalities**, there is still 0.16 and 0.27 improvements of BLEU in LayoutReader-text only and LayoutReader-layout only and a steady 6.15 reduction of ARD in LayoutReader-layout only
    - A growth of ARD in LayoutReader-text only is visible, mainly because of severe punishment in ARD for token omission. LayoutReader-text only can guarantee right order of tokens but suffers from generation incompleteness
    - **Layout info plays a more important role than textual info in reading order detection**. LayoutReader-layout only surpasses LayoutReader-text only by ~0.1 in BLEU and ~9.0 in ARD

Method	**Encoder**	**Avg. Page-level BLEU ↑**	**ARD ↓**
Heuristic Method	-	0.6972	8.46
LayoutReader (text only)	BERT	0.8510	12.08
	UniLM	0.8765	10.65
LayoutReader (layout only)	LayoutLM (layout only)	0.9732	2.31
LayoutReader	LayoutLM	**0.9819**	**1.75**

Image source: Paper*

- **Shuffling input tokens of seq-to-seq model** in a certain proportion of training samples to **study accuracy of LayoutReader for different input orders**
- Proportion of token-shuffled training samples is denoted as r, building three versions of comparative models with r equaling **100%, 50% and 0%**
- Left-to-right and top-to-bottom order provide remarkable hints for reading order detection. However, in this input order study, these hints are incomplete during training
- Table shows results when **evaluating comparative models with left-to-right and top-to-bottom inputs**:
  - **LayoutReader-layout only** and **LayoutReader** are more **robust to shuffled tokens during training** and **all three comparative models perform well with left-to-right and top-to-bottom inputs in evaluation**
  - It can be attributed to consideration of **layout info**, which **is consistent under shuffling**

Method	Avg. Page-level BLEU ↑			ARD ↓		
	r=100%	r=50%	r=0%	r=100%	r=50%	r=0%
LayoutReader (text only, BERT)	0.3355	0.8397	0.8510	77.97	15.62	12.08
LayoutReader (text only, UniLM)	0.3440	0.8588	0.8765	78.67	13.65	10.65
LayoutReader (layout only)	0.9701	0.9729	0.9732	2.85	2.61	2.31
LayoutReader	**0.9765**	**0.9788**	**0.9819**	**2.50**	**2.24**	**1.75**

Image source: Paper*

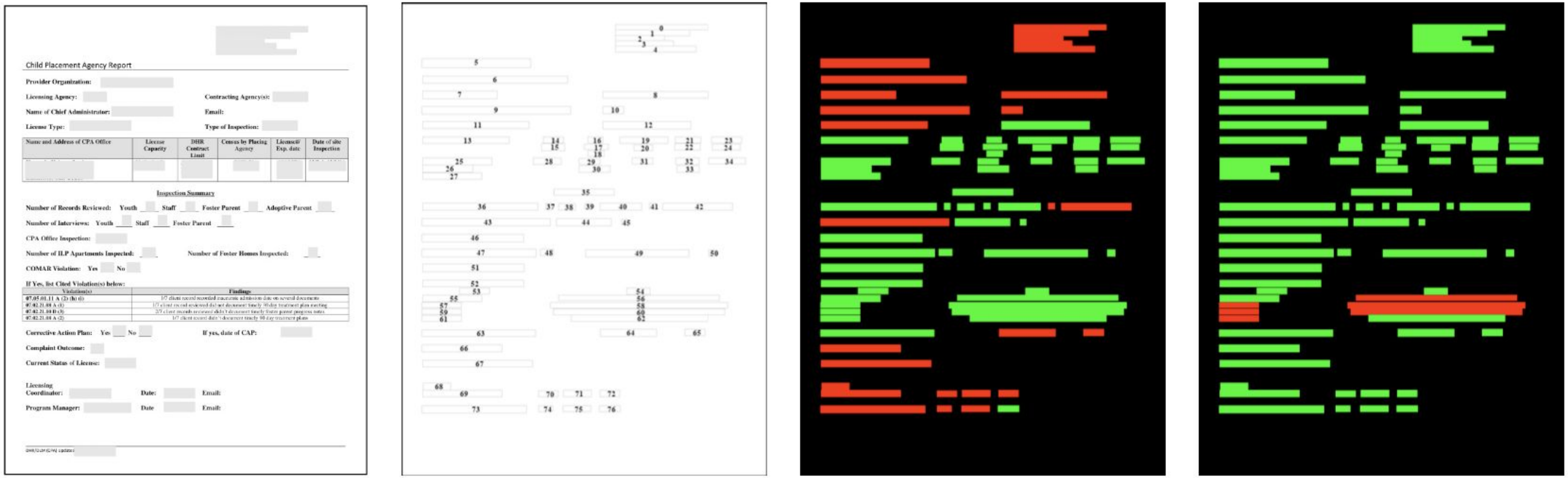
- Most OCR engines provide reading order info for text lines, where some of them may be problematic. **To improve text line ordering, extended token-level reading order to text lines and adapted it to OCR engines**
- First **assigned each token** in our token-level order **to text lines according to percentage of spatial overlapping**
- Given a token bounding box b and **a text line bounding box B, token is assigned to text line which overlaps most with token, i.e. $\hat{B} = argmax_B\ (B \cap b)$, where $\cap$ means spacial overlapping**. Then calculated minimum of token indices in each text line as its ranking value and produce an improved text line order from token-level order
- Note that token-level order can be the order given by ReadingBank or result generated by LayoutReader. So, built a text line ordering ground truth by adapting ReadingBank to text lines and evaluate performance of LayoutReader in text line ordering accordingly, reporting also performance of Heuristic Method and OCR engines
- **Experiments with** an open source **OCR engine Tesseract and a** cloud-based **commercial OCR API**, respectively left and right results are shown below: a great improvement with LayoutReader adaption in both cases
- This experiment **further demonstrates effectiveness** and extends application **of LayoutReader**

Method	Avg. Page-level BLEU ↑	ARD ↓
Heuristic Method	0.3391	13.61
Tesseract OCR	0.7532	1.42
LayoutReader	**0.9360**	**0.27**

Method	Avg. Page-level BLEU ↑	ARD ↓
Heuristic Method	0.3752	10.17
The commercial OCR	0.8530	2.40
LayoutReader	**0.9430**	**0.59**

Images source: Paper*

- Selected a **representative example from test set** and show text line orders in the figure
- **Comparing** text line order of a **OCR** engine and **LayoutReader Adaption** with groundtruth from ReadingBank Adaption
- Results with colors where **green and red denotes correct and incorrect results**
- **LayoutReader Adaption improves text line ordering of OCR engine**, confirming results



(a) Original image (b) Groundtruth (c) The commercial OCR (d) LayoutReader

Image source: Paper*

- Introduced:
  - **ReadingBank**, a **benchmark dataset** for reading order detection that contains 500,000 document images
  - **LayoutReader**, a novel **reading order detection approach built upon pre-trained *LayoutLM* model**
- Experiments show that **LayoutReader has significantly outperformed** left-to-right and top-to-bottom **heuristics as well as several strong baselines**
- Furthermore, it can be **easily adapted to any OCR engines** so that reading order can be improved for downstream tasks

- **Future research**:
  - Investigate how to **generate a larger synthesized dataset from ReadingBank**, where noisy information and rotation can be applied to clean images to make model more robust
  - **Label reading order information on a real-world dataset** from scanned documents
  - Considering LayoutReader model as a pre-trained reading order detection model, explore whether a **few human labeled samples would be sufficient for reading order detection in a specific domain**

https://www.vecteezy.com/photo/22454669-3d-rendering-of-a-little-robot-reading-a-book-in-a-dark-room-generative-ai

Thanks for Attention

Images and text have been gathered from the paper*: "*LayoutReader: Pre-training of Text and Layout for Reading Order Detection*" | *Zilong Wang, Yiheng Xu, Lei Cui, Jingbo Shang, Furu Wei* | *University of California, San Diego, Microsoft Research Asia* | 27 Aug 2021"