Improving OCR Pipeline @Decimal Point Analytics

Issue at hand.

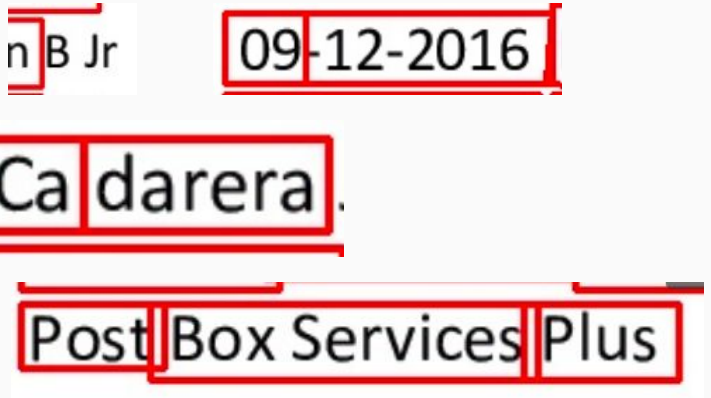
- Current pipeline not end-to-end
  - Detection + Recognition
- Issue with recognition pipeline??
  - **Unable to reproduce white-spaces**
  - Unable to recognize special characters
  - Unable to parse numbers in financial amounts and hyperlinks
- Another model text-fuse-net is used to detect white spaces**

Why?

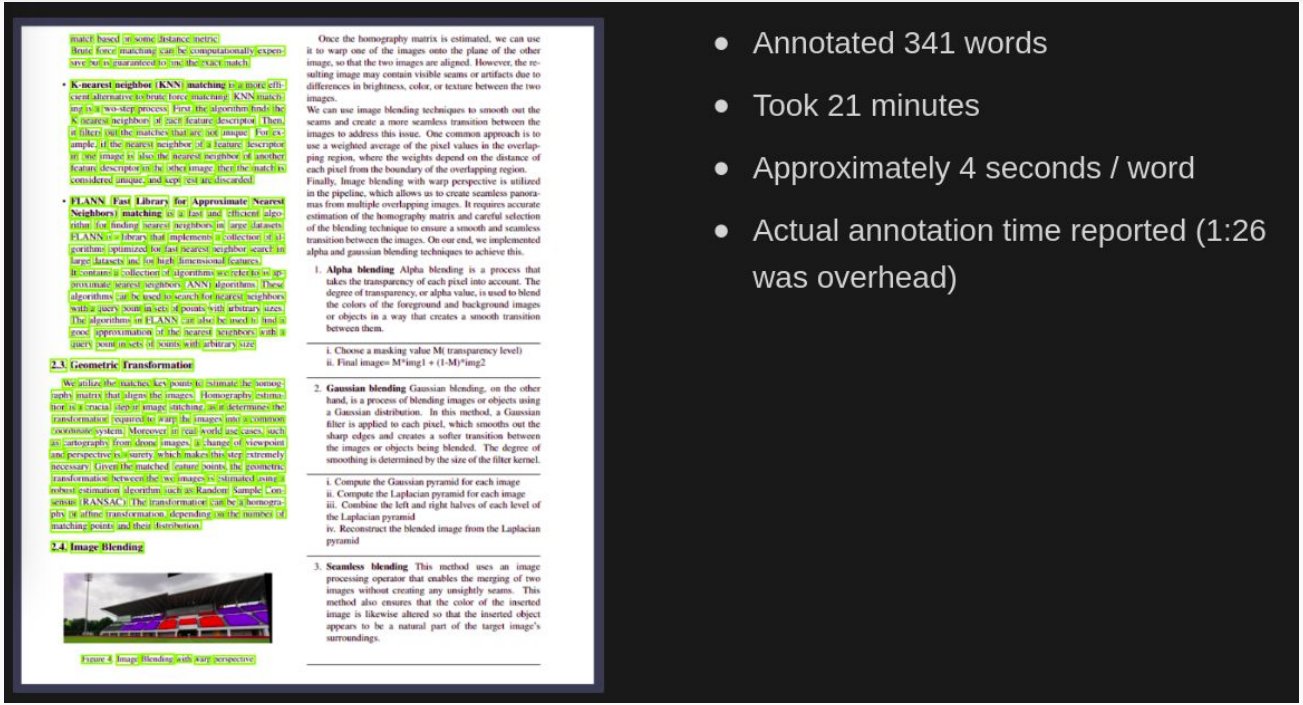
Approach #1: Improve detection to one word

- 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 ??

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



Approach #1: Improve detection to one word



match based on some distance metric.
Brute force matching can be computationally expensive, but is guaranteed to find the exact match.

- **K-nearest neighbor (KNN) matching** is a more efficient alternative to brute force matching. KNN matching is a two-step process. First, the algorithm finds the K nearest neighbors of each feature descriptor. Then, it filters out the matches that are not unique. For example, if the nearest neighbor of a feature descriptor in one image is also the nearest neighbor of another feature descriptor in the other image, then the match is considered unique, and kept. Rest are discarded.
- **FLANN (Fast Library for Approximate Nearest Neighbors) matching** is a fast and efficient algorithm for finding nearest neighbors in large datasets. FLANN is a library that implements a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. It contains a collection of algorithms we refer to as approximate nearest neighbors (ANN) algorithms. These algorithms can be used to search for nearest neighbors with a query point in sets of points with arbitrary sizes. The algorithms in FLANN can also be used to find a good approximation of the nearest neighbors with a query point in sets of points with arbitrary size.

2.3. Geometric Transformation

We utilize the matched key points to estimate the homography matrix that aligns the images. Homography estimation is a crucial step in image stitching as it determines the transformation required to warp the images into a common coordinate system. Moreover, in real world use cases, such as cartography from drone images, a change of viewpoint and perspective is surely which makes this step extremely necessary. Given the matched feature points, the geometric transformation between the two images is estimated using a robust estimation algorithm such as Random Sample Consensus (RANSAC). The transformation can be a homography or affine transformation, depending on the number of matching points and their distribution.

2.4. Image Blending

Figure 4. Image Blending with warp perspective

Once the homography matrix is estimated, we can use it to warp one of the images onto the plane of the other image, so that the two images are aligned. However, the resulting image may contain visible seams or artifacts due to differences in brightness, color, or texture between the two images.
We can use image blending techniques to smooth out the seams and create a more seamless transition between the images to address this issue. One common approach is to use a weighted average of the pixel values in the overlapping region, where the weights depend on the distance of each pixel from the boundary of the overlapping region.
Finally, Image blending with warp perspective is utilized in the pipeline, which allows us to create seamless panoramas from multiple overlapping images. It requires accurate estimation of the homography matrix and careful selection of the blending technique to ensure a smooth and seamless transition between the images. On our end, we implemented alpha and gaussian blending techniques to achieve this.

1. **Alpha blending** Alpha blending is a process that takes the transparency of each pixel into account. The degree of transparency, or alpha value, is used to blend the colors of the foreground and background images or objects in a way that creates a smooth transition between them.

   i. Choose a masking value M(transparency level)
   ii. Final image= M*img1 + (1-M)*img2

2. **Gaussian blending** Gaussian blending, on the other hand, is a process of blending images or objects using a Gaussian distribution. In this method, a Gaussian filter is applied to each pixel, which smooths out the sharp edges and creates a softer transition between the images or objects being blended. The degree of smoothing is determined by the size of the filter kernel.

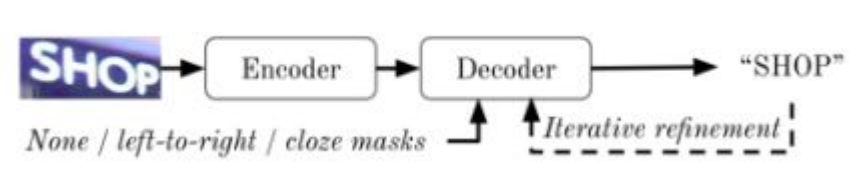
   i. Compute the Gaussian pyramid for each image
   ii. Compute the Laplacian pyramid for each image
   iii. Combine the left and right halves of each level of the Laplacian pyramid
   iv. Reconstruct the blended image from the Laplacian pyramid

3. **Seamless blending** This method uses an image processing operator that enables the merging of two images without creating any unsightly seams. This method also ensures that the color of the inserted image is likewise altered so that the inserted object appears to be a natural part of the target image's surroundings.

- Annotated 341 words
- Took 21 minutes
- Approximately 4 seconds / word
- Actual annotation time reported (1:26 was overhead)

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**

>Also explored the issue of superscripts and subscripts and how it cannot be tackled with the given setup.

> Inference, visualization, testing, demo, early-stopping, logging scripts

Deploying and documentation

- Installation
- Inferencing
  - With ground truth available
  - Without ground truth available
    - With image
    - Without image
- Training
- Creating custom dataset
- Model description and logs
- Dataset description

PARSEQ-robust

Getting started

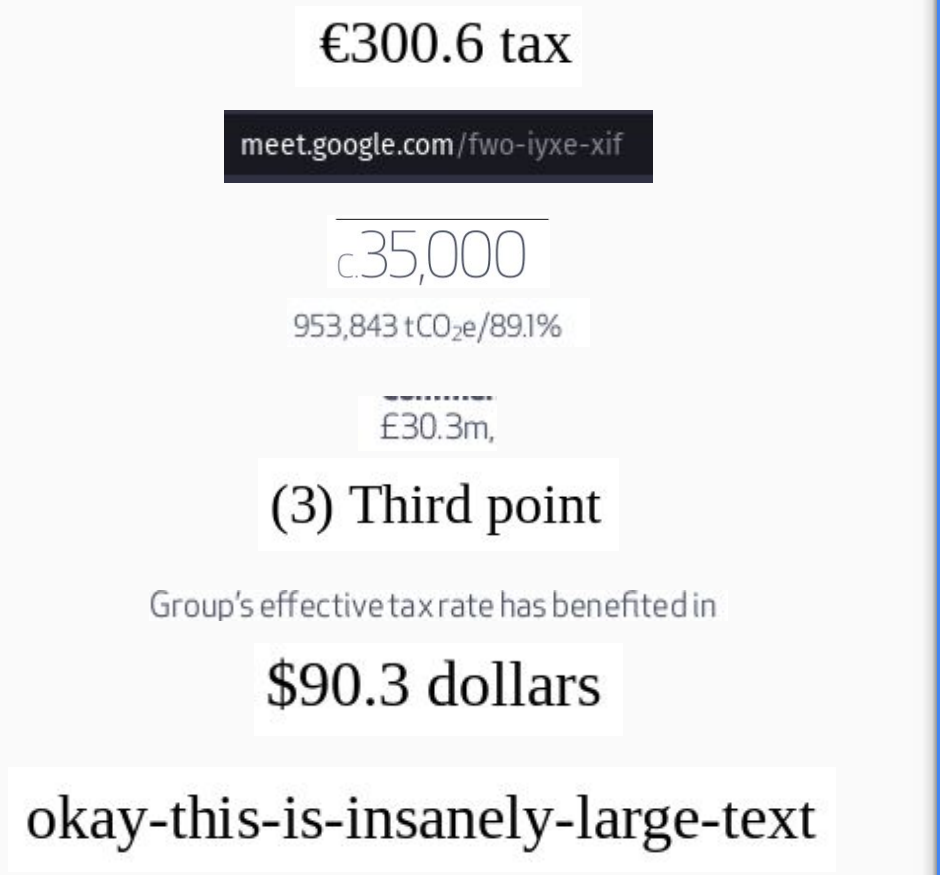
This repository contains code for fine-tuning the OCR pipeline and its inference and other supporting scripts.

- To view unedited HTML-Logs (*recommended*); download the repo locally and open the html file here
- Atlernatively, logs in PDF can be found here
- Please note; that logs run from bottom to top; that is, initial stages of development are down below and more recent developments were logged above.

Repo description

- assets : Supporting documents and scripts
- ds-maker : Script for constructing dataset
- sub-model : Model checkpoints and logs
- suite-infer : Demo script
- v2 : Codebase for training/ testing

Image



PARSEQ-old	PARSEQ-new
300.6tax	€300.6 tax
meet.google.com/two-iyxe-	meet.google.com/fwo-iyxe-xif
35,000	c35,000
953,843tCO,e/891%	953,843tCO2e/89.1%
30.3m,	£30.3m,
(3)Thirdpoint	(3) Third point
Groupseffectivetatehacdin	Group's effective taxrate has benefited in
$90.3dollars	$90.3 dollars
okay-this-insamely-lageext	okay-this-is-insanely-large-text

Wrapping things up….

- Old model unable to detect white-spaces and certain characters
  - Hallucinates with long labels and symbol-heavy labels
- Time saved by using 1 model instead of two
- Additional cases, symbols, contexts, data-populations can easily be covered with the current codebase set-up.

Thank you :)