Text Extraction from Image and Basic Text Analysis

Petropoulos Panagiotis

panos.petr1@gmail.com

Student, MSc in Artificial Intelligence,

University of Piraeus & NCSR Demokritos,

Athens, Greece, June 2022

1. Abstract

Nowadays there are a variety of image text detection applications. There is an urgent need for such applications for many reasons such as data annotation, segmentation and for various other functions that facilitate modern life. In the present paper I created such a tool that has the ability to extract text from an image to make a segmentation in it related to region of words and characters, where the text was found, but also to perform text analysis with basic Natural Language Processing techniques, with the aim of Feature extraction for solving other machine learning tasks.

2. Introduction

There is a lot of research on text extraction from images. Sometimes we need to get information about the extracted texts in order to get feedback about the linguistic characteristics, the grammar or the syntax of them. In this particular paper, a text extraction is performed using the easyOCR[1] library and then some text analysis methods to extract elements and features from the extracted text. The purpose is to create a tool that will extract text from images and analyze the content in terms of basic text analysis methods:

1. Tokenization
2. Stemming
3. Lemmatization
4. POS tagging

EasyOCR is a python package that was created by Jaided AI[2], a company that specializes in Optical Character Recognition services. This package uses the CRAFT algorithm [1] for feature extraction and the estimation about the region and affinity among the characters. The main recognition model of easyOCR is a CRNN [2] which consists of 3 main components:

1. ResNet for feature extraction [3].
2. LSTM for Sequence labeling.
3. CTC (Connectionist Temporal Classification) for decoding [4].

[1] https://github.com/JaidedAI/EasyOCR
[2] https://jaided.ai/

In order to train the easyOCR model there was used TextRecognitionDataGenerator[3] to generate Data.

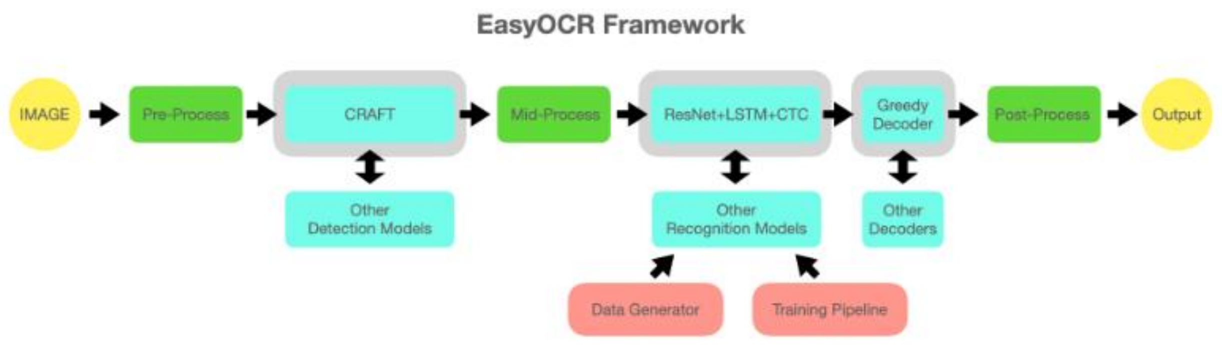


Figure-1: easyOCR Implementation RoadMap and architecture. [5]

3. Data

The data was collected from the internet. There are 4 images with digital content, namely written text from a computer and 4 images with handwritten text. The content of the texts is written in English. To define ground truth, an online tool was used to extract text from an image with great accuracy. Then, the parts that were not extracted properly were manually corrected.

4. Method

For the text extraction, the easyOCR library was used, which generates a confidence posterior probability for the extracted text. Then based on a confidence threshold the user can confirm how many words were retrieved with confidence over this threshold. Some of the images, especially the handwritten, need preprocessing steps in order to be able, for a model, to extract the text more accurately.

The following basic methods were used for image processing:

1. Median Blur Filter to clear the “salt and pepper” noise
2. Gaussian Blur filter to clear noise.

The following methods were used for text analysis:

1. Tokenization
2. Stemming
3. Lemmatization
4. POS tagging

The NLTK and Spacy libraries were used for all of these methods. In the case of tokenization and stemming, except for these libraries, a custom tokenizer was created using Regular Expressions. We will compare it with other methods. Two types of experiments were performed for tokenization and stemming. Having as ground truth the following cases:

[3] https://github.com/Belval/TextRecognitionDataGenerator

1. Use ground truth text from txt files with:
    a. NLTK[4]
    b. SpaCy[5]

2. Use extracted text as ground truth with::
    a. NLTK
    b. SpaCy

Based on the above ground truths I compared the Custom Tokenizer and the Custom Stemmer. The same experiments were followed and in the case of Stemming simply ground truth came up from:

1. SnowBall Stemmer
2. Porter Stemmer

Therefore, this paper focuses on creating a tool where the user will enter an image, features will be generated and some text analysis elements such as:

1. Tokenization
2. Lemmatization
3. Stemming
4. POS Tagging
5. POS Tagging Tree
6. Top 30 most common tokens
7. Percentile of tokens that were observed 1 time in the whole document.

For number 7, I compared the percentage with Zipf's law, which says that 50% of tokens in the Document appear once.

5. Results

Sometimes, many images have noise like salt & pepper. Below we can see an example of an image with the "before" and "after" preprocessing to clean the noise. Two methods are applied for this purpose, a median[6] and a Gaussian filter[7].

[4] https://www.nltk.org/howto/corpus.html
[5] https://spacy.io/api/tokenizer

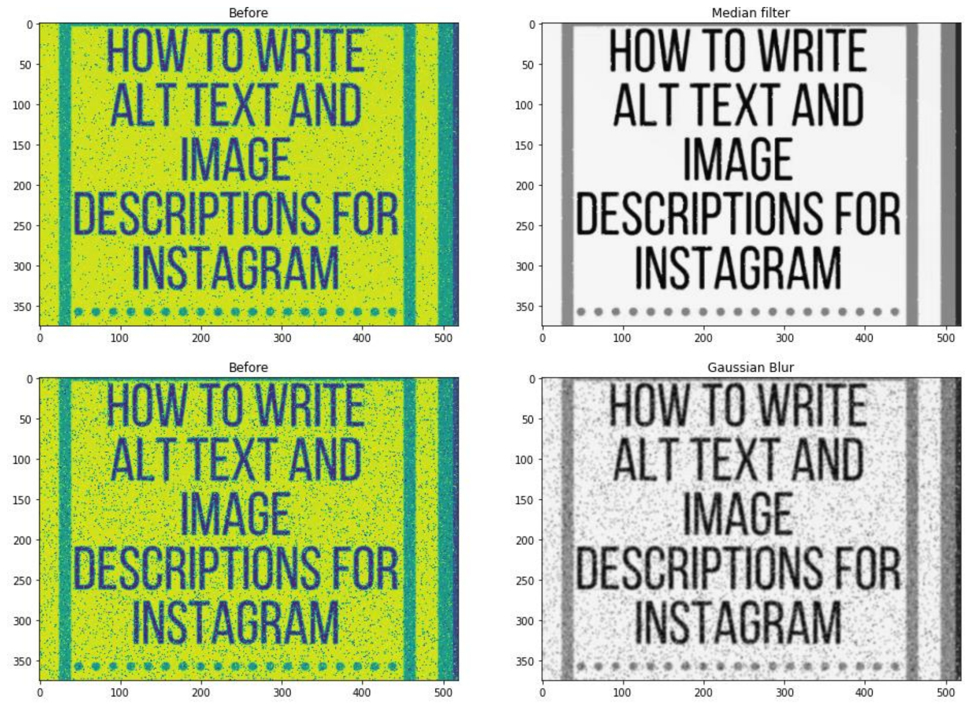


Figure-2: Before and after applying Median (top) and Gaussian (bottom) filters.

The above step of pre-process the image is not mandatory but is an option in the final python package I created. As a second step in the below images (digital and handwritten) we can see an example of an extraction of text. The text is represented on the image as annotation. Also, I took the pixel values from easyOCR output, in order to create the rectangles over the words that this package extracted.

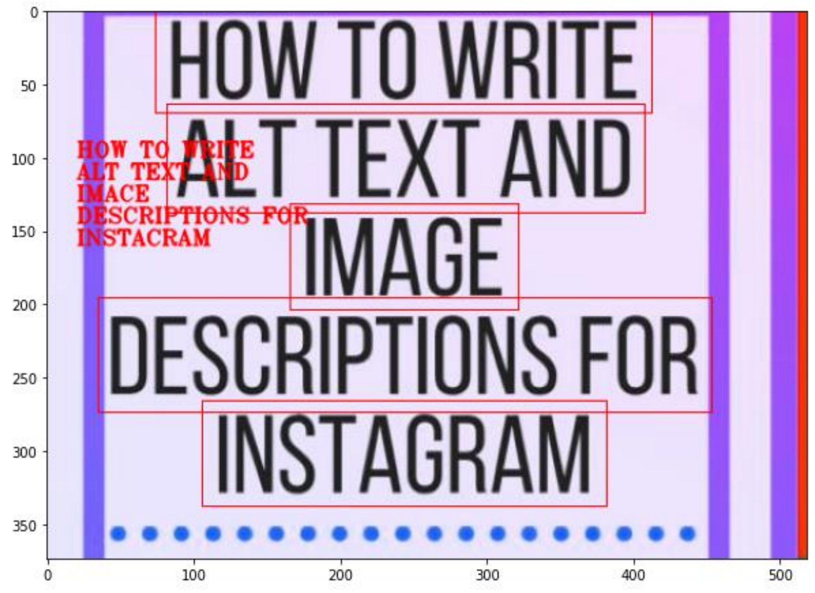


Figure-3: Result from easyOCR in digital image.

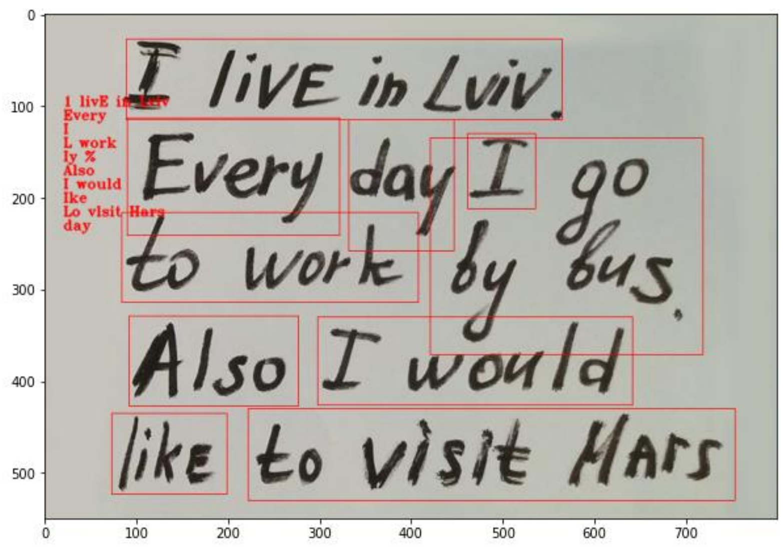


Figure-4: Result from easyOCR in handwritten image.

From Figure-4 we can observe that the easyOCR model is not so accurate in handwritten characters.

After the text extraction, I performed text analysis using Natural Language Processing techniques. For testing my custom Tokenizer and stemmer I took 4 images with essays as content. In this paper, I am reporting only the results from the first two essays. Further information and results can be found in Colab Notebook[6].

Table-1 shows us the percentile of tokens that appeared once in the whole document (essay). Compared to Ziph's Law[7] this percentile is almost 50%. In this project I got almost 65%.

	percentage of tokens with Frequency = 1 per Tokenizer (Essay 1)	Frequency = 1 (%)
0	NLTK word_tokenize	0.669456
1	SpaCy	0.649789
2	Custom	0.672199

	percentage of tokens with Frequency = 1 per Tokenizer (Essay 2)	Frequency = 1 (%)
0	NLTK word_tokenize	0.669456
1	SpaCy	0.649789
2	Custom	0.672199

Table-1: percentile of tokens that appeared once in the whole document.

[6] https://nbviewer.org/github/icsd13152/TextExtraction_from_image/blob/main/Multimodal.ipynb

[7] https://en.wikipedia.org/wiki/Zipf%27s_law

The below images show us the top 40 tokens and their frequencies in each of the essays. The tokens like “the”, “to”, “you” “a '' are the most frequent tokens in the whole document. That is something obvious, due to the fact that those words are determiners, as we can see in the Colab notebook from the POS tagging procedure. Below is an example.

```
('The', 'DT'),
```

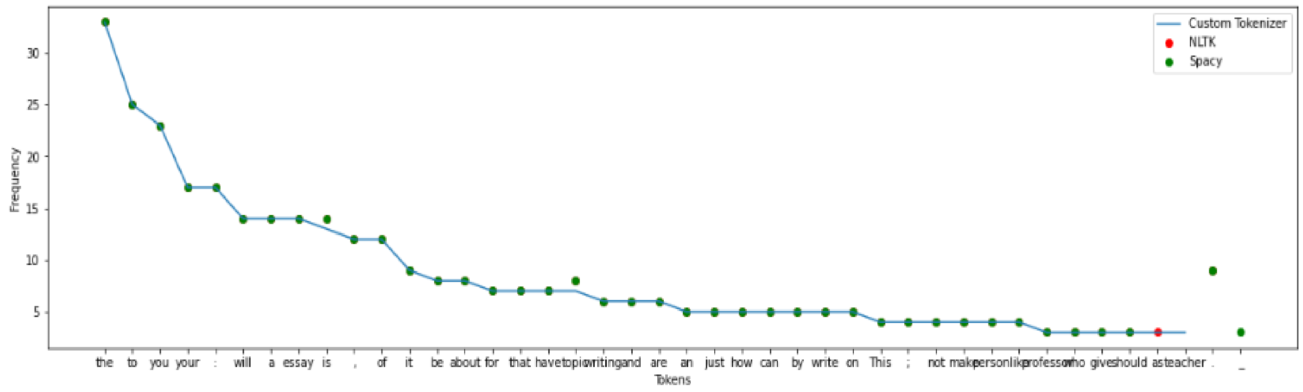


Figure-5: top 40 tokens and their frequencies in essay 1.

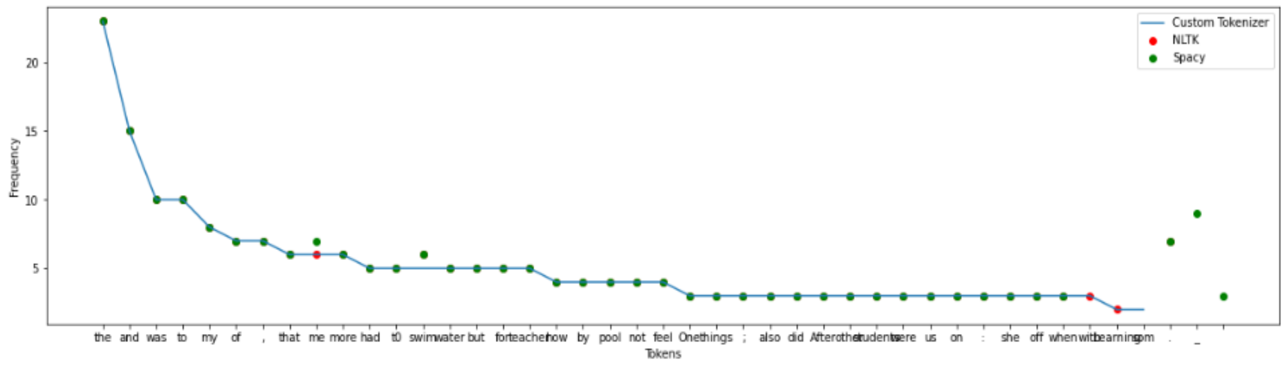


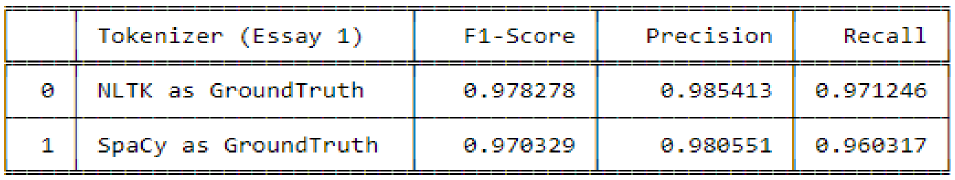
Figure-6: top 40 tokens and their frequencies in essay 2.

Blue line from the above images shows us the tokens that were extracted from Custom tokenizer. As we can see, in both essays my custom Tokenizer has almost the same results as NLTK and SpaCy.

I used precision, recall and F1-score to evaluate my custom Tokenizer and Stemmer as described in the below mathematical types and based on information retrieval:

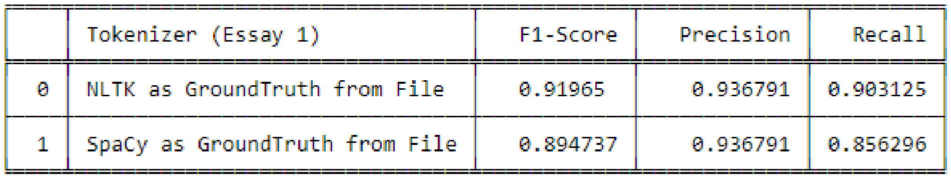
$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$



	Tokenizer (Essay 1)	F1-Score	Precision	Recall
0	NLTK as GroundTruth	0.978278	0.985413	0.971246
1	SpaCy as GroundTruth	0.970329	0.980551	0.960317

Table-2: precision, recall, f1-score for essay 1 having as ground truth results from NLTK and SpaCy.

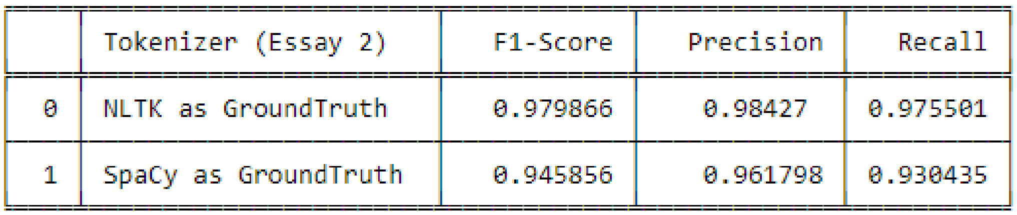


	Tokenizer (Essay 1)	F1-Score	Precision	Recall
0	NLTK as GroundTruth from File	0.91965	0.936791	0.903125
1	SpaCy as GroundTruth from File	0.894737	0.936791	0.856296

Table-3: precision, recall, f1-score for essay 1 having as ground truth results from NLTK and SpaCy from text files.

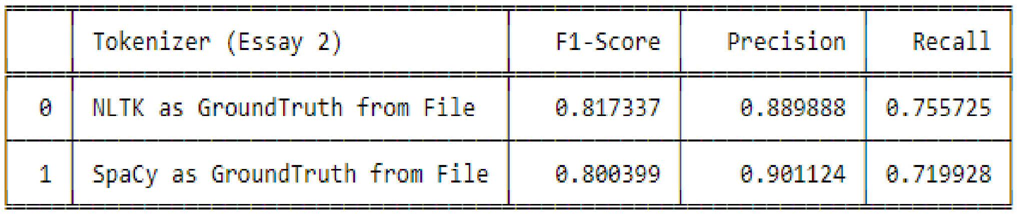
The above tables show us the results of custom tokenizer using as ground truth the tokens from NLTK and SpaCy. In Table-2, the tokens were extracted directly from images and in Table 3 tokens were extracted from text files. Custom Tokenizer for Essay 1 achieves over 91% of F1-score and over 90% of Recall.

The same results are presented below for Essay 2.



	Tokenizer (Essay 2)	F1-Score	Precision	Recall
0	NLTK as GroundTruth	0.979866	0.98427	0.975501
1	SpaCy as GroundTruth	0.945856	0.961798	0.930435

Table-4: precision, recall,f1-score for essay 2 having as ground truth results from NLTK and SpaCy.



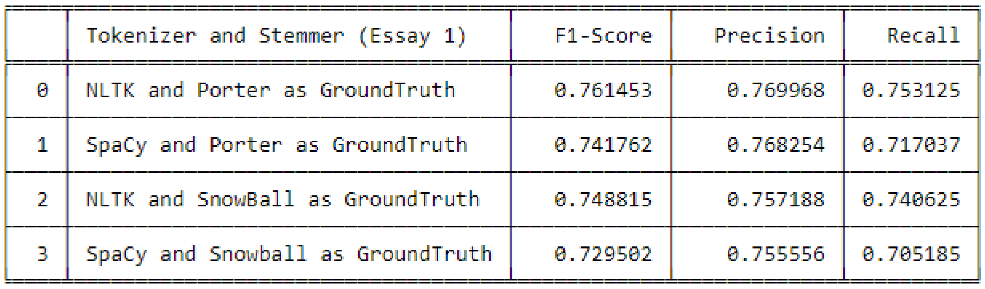
	Tokenizer (Essay 2)	F1-Score	Precision	Recall
0	NLTK as GroundTruth from File	0.817337	0.889888	0.755725
1	SpaCy as GroundTruth from File	0.800399	0.901124	0.719928

Table-5: precision, recall,f1-score for essay 2 having as ground truth results from NLTK and SpaCy from text files.

Finally, the results from custom Stemmer are presented below, in Table-6 and Table-7. For this experiment the evaluation is performed based on the below approaches of ground truth:

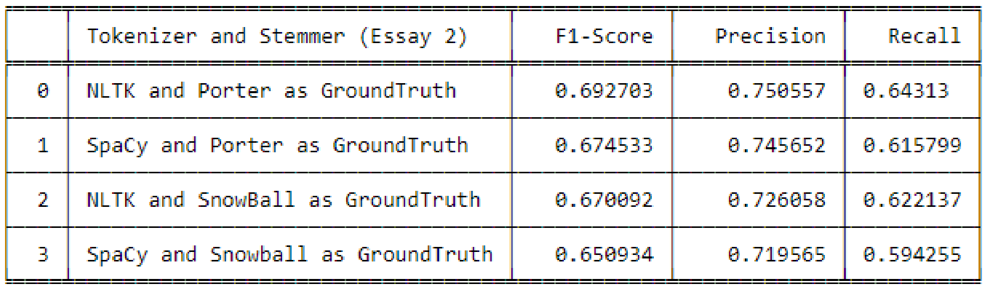
1. NLTK tokens and Porter stemmer directly from images.
2. NLTK tokens and Snowball stemmer directly from images.
3. SpaCy tokens and Porter stemmer directly from images.
4. SpaCy tokens and Snowball stemmer directly from images.

The above ground truths are considered for both essays.



	Tokenizer and Stemmer (Essay 1)	F1-Score	Precision	Recall
0	NLTK and Porter as GroundTruth	0.761453	0.769968	0.753125
1	SpaCy and Porter as GroundTruth	0.741762	0.768254	0.717037
2	NLTK and SnowBall as GroundTruth	0.748815	0.757188	0.740625
3	SpaCy and Snowball as GroundTruth	0.729502	0.755556	0.705185

Table-6: precision, recall, f1-score for essay 1.



	Tokenizer and Stemmer (Essay 2)	F1-Score	Precision	Recall
0	NLTK and Porter as GroundTruth	0.692703	0.750557	0.64313
1	SpaCy and Porter as GroundTruth	0.674533	0.745652	0.615799
2	NLTK and SnowBall as GroundTruth	0.670092	0.726058	0.622137
3	SpaCy and Snowball as GroundTruth	0.650934	0.719565	0.594255

Table-7: precision, recall, f1-score for essay 1.

My custom Stemmer achieves over 70% F1-score in essay 1 and almost 70% in essay 2.

6. Conclusion & Future work

In the future, it would be worthwhile to create custom POS taggers and Lemmatizers. Especially POS tagging needs linguistic analysis because each language has its own peculiarities. Also, as an extra feature extraction for the cases that we need Embeddings Vectors, it would be worthwhile to do experiments with:

1. Embeddings from BERT pretrained model.
2. Embeddings from Word2Vec

Finally, models for other languages could be defined as additional work.

References

[1]. Baek, Youngmin, et al. "Character region awareness for text detection." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[2]. Shi, Baoguang, Xiang Bai, and Cong Yao. "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition." IEEE transactions on pattern analysis and machine intelligence 39.11 (2016): 2298-2304.

[3]. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[4]. Graves, Alex, et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." Proceedings of the 23rd international conference on Machine learning. 2006.

[5]. https://github.com/JaidedAI/EasyOCR

[6]. Chang, Chin-Chen, Ju-Yuan Hsiao, and Chih-Ping Hsieh. "An adaptive median filter for image denoising." 2008 Second International Symposium on Intelligent Information Technology Application. Vol. 2. IEEE, 2008.

[7]. Deng, Guang, and L. W. Cahill. "An adaptive Gaussian filter for noise reduction and edge detection." 1993 IEEE conference record nuclear science symposium and medical imaging conference. IEEE, 1993.

[8] https://en.wikipedia.org/wiki/Zipf%27s_law