# Tesseract OCR for Dyslexia and non-Dyslexia typefaces

Lorenzo Caselli
Matricola: 7061365
lorenzo.caselli1@stud.unifi.it

Lisa Cresti
Matricola: 7045456
lisa.cresti@stud.unifi.it

## Abstract

*Dyslexia is a learning disability in reading and writing. Dyslexic persons may see characters distorted, creating difficulties during their daily life. Reading a book for pleasure could become vexing and exhausting. With the aim of improving the reading experience, over the years different typefaces have been developed.*

*In this work we decided to train an OCR, in particular Tesseract, with various typeface categories, including those designed for dyslexia. We then applied image processing tecniques in order to simulate the visual processing disorders of dyslexic subjects. Finally, we assessed how much recognition performances deteriorate as the scale of distortions grew.*

*This way we want to evaluate whether typefaces designed for Dyslexics actually manage to be more resilient than other categories, such as Serif or Sans Serif ones.*

## 1. Introduction

Dyslexia is a specific learning disability that can cause problems with certain abilities, such as reading and writing. It is a lifelong problem that can present challenges on daily bases and it is important to present support. Some signs of dyslexia are: confusing letters in words, inconsistent spelling and difficulty with information that is written down. Technology and techniques can often support people with these difficulties and an example can be the use of certain types of typefaces. Specifically, some typefaces are designed exclusively to address these kind of symptoms.

In our work, we intend to simulate the visual processing disorders of dyslexic subjects to test if these kind of typefaces are effectively more resilient to distortions than others. To perform this study we trained models of several typefaces, dyslexia-designed and not, to obtain recognition of visually distorted images.

In the first part of this paper, in section 2 and 3, we present the typefaces we choose to work with and how we generated our dataset. In section 4, we present the OCR engine we used to create the models, Tesseract. In section 5, we introduce the image distortions we worked with. Finally in section 6 and 7, we discuss about the experiments done and the results we obtained.

## 2. Typefaces

A typeface is a particular alphabet, numerals and punctuation, that share a common style. It plays an important role in documents design especially regarding readability and legibility functions. Research demonstrates that standard typefaces are not as effective for readers with dyslexia as for other readers. Therefore, over time studies were carried on in order to ensure an easier and more comfortable time performing reading. Because of that, many new typefaces were exclusively designed for readers with dyslexia. Some examples are Dyslexie, Open Dyslexic, Lexie Readable, Sylexiad and so on [2]. Figure 1 shows the letterforms of dyslexia typefaces compared to a standard typeface, Arial.

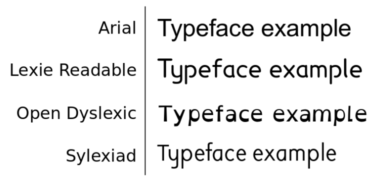


Figure 1: Comparison between Arial and Dyslexic typefaces.

Since different typefaces can determine whether people with visual disorders can read with ease or not, we decided to work on three different typeface categories: Serif, Sans Serif and Dyslexia-designed.

The main feature of a Serif typeface is the presence of an extend at the end of the letterform with a decorative stroke, which makes the text more formal and readable. Sans-serif fonts do not present these decorative strokes and are often considered easier to read using digital mediums, making the experience of reading the most comfortable possible. The assumptions we just made about the ease of reading these two typeface categories do not hold for readers with dyslexia.

The characteristics that make a font dyslexia-friendly have the aim of reducing the symmetry present between letters. This happens because dyslexic readers often present difficulties in distinguishing certain couples of characters, either because some of then mirrors others, or because they present the same height. For example if we take p and q, we observe that in most common typefaces one is the exact mirror of the other. The aim is therefore to be able to break these similarities by modifying their shapes and differentiating them especially between similar characters, widening the stroke in certain areas or breaking the continuity of the line. So we obtain a unique shape for each one in order to facilitate the recognition of the character [6].

We used 10 different typefaces as pictured in fig 2, grouped in each category.

| | | |
|---|---|---|
| **Dyslexic** | Lexie Readable | Sympathique |
| | Open Dyslexic | Sympathique |
| | Sylexiad | Sympathique |
| **Sans Serif** | Arial | Sympathique |
| | Comic Sans Ms | Sympathique |
| | Tahoma | Sympathique |
| | Verdana | Sympathique |
| **Serif** | Baskervville | Sympathique |
| | Georgia | Sympathique |
| | Times New Roman | Sympathique |

Figure 2: List of typefaces used.

## 3. Dataset Creation

The dataset we used was generated from an English literature book, from which we extracted the content. Then we created a TXT on which we removed the metadata and special symbols brought by the conversion from the PDF format. We did it because it could be an hindrance during the training.

The next step was to use LaTex to create PDF files with the typefaces introduced in the previous chapter. The *fontspec* library [5] has allowed us to generate, starting from the same information content, a series of PDF files by iterating on the typefaces we required. This was necessary in order to start training Tesseract OCR, for which we need to convert our PDFs into single line images.

### 3.1. Binarization and Segmentation

To get single line images, we separated our PDFs into single pages, which will be first binarized and then segmented. This process creates a JSON file which stores the bounding boxes informations used to segment the source page in single lines. Through the Python *Kraken* library [9], we were able to perform these steps.

This process was more challenging than we initially expected. In fact, after few tries we realized that using the same parameters for each typeface didn't create a correct segmentation. For binarization it was necessary to examine the typeface design to find the most correct threshold parameter. We empirically found that typefaces with very close characters, such as Times New Roman, require a lower threshold. With a correct binarization of the page, it followed a correct identification of the connected components that identifies the single lines. It all ends with the segmentation of the page into line boxes as desired. The images were saved with the TIFF extension, because this is one of the favorite ones for Tesseract training.

In figures 3, 4 and 5 we show this process.

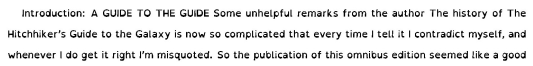
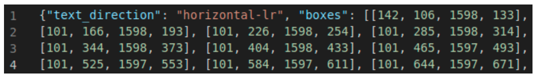

Introduction: A GUIDE TO THE GUIDE Some unhelpful remarks from the author The history of The
Hitchhiker's Guide to the Galaxy is now so complicated that every time I tell it I contradict myself, and
whenever I do get it right I'm misquoted. So the publication of this omnibus edition seemed like a good

Figure 3: Binarized image



```
{"text_direction": "horizontal-lr", "boxes": [[142, 106, 1598, 133],
[101, 166, 1598, 193], [101, 226, 1598, 254], [101, 285, 1598, 314],
[101, 344, 1598, 373], [101, 404, 1598, 433], [101, 465, 1597, 493],
[101, 525, 1597, 553], [101, 584, 1597, 611], [101, 644, 1597, 671],
```

Figure 4: Line boxes

Introduction: A GUIDE TO THE GUIDE Some unhelpful remarks from the author The history of The

Figure 5: From binarized to segmented image

### 3.2. Ground Truth Generation

In order to proceed with the training, Tesseract also needs ground truths of our data, meaning that he needs to couple each image to a transcription of its content. So we finally made TXT files containing the textual information in each single line image.

Starting from the previously generated PDFs, we extracted the content using *PDFMiner*, a Python library that provides tools to handle PDFs [11].

## 4. Training

In our project, we decided to produce a model for each typeface chosen. So once the dataset was prepared, with images and ground truths, we selected Tesseract to perform the training.

### 4.1. Tesseract OCR

Tesseract is an OCR engine that was initially developed at HP in the early '90s and then, in 2005, released as open-source [12]. Since 2006 the development has been sponsored by Google. The current official release is 4.1.1 [14], which is the one we used, even if a newer version is already under development.

Tesseract 4.0 created a new OCR engine based on LSTM neural networks. A LSTM, long short-term memory, is an artificial recurrent neural network that has feedback connections, so it can take into account all of the predecessor words [13], and can process entire sequence of data such as speech and video. This kind of networks yielded excellent results on handwriting recognition [3].

In our case, we performed the training using about 2500 single line image for each typeface. These were split between evaluation and trainig set in a ratio of 1:9 using the RATIO_TRAIN parameter of Tesseract. Table 6 shows the parameters chosen for training.

| Training Parameters | |
|---:|:---|
| PSM | 7 |
| MAX_ITERATIONS | 100000 |
| RATIO_TRAIN | 0.9 |
| LEARNING_RATE | 0.002 |
| TARGET_ERROR_RATE | 0.01 |

Figure 6: Assigned values during training.

## 5. Image Distortions

To simulate the visual distortions perceived by a dyslexic subject, we created a different set of images of English text, on which we applied some distortions. We took the textual information from an English literature book and followed the same generation scheme described for the creation of the training set, thus obtaining about 370 single line images for each typeface and the respective ground truths. Taking a cue from the article [15], we decided to work on three different distortions. Our decision was to apply three different scale parameters for each transformation to the image set just mentioned.

### 5.1. Blurring

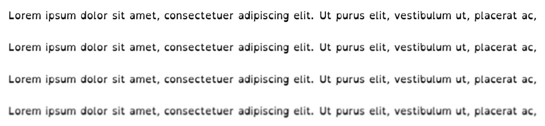

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,

Figure 7: Blurring with different rates

Blurring an image is equivalent to reducing the details. This can be done by smoothing the color transition between the pixels. To accomplish this, we need to apply a convolutional operation of a specialized matrix, called kernel, to the image's matrix. In this case, we used a gaussian kernel with different parameters to create different grades of blurring. To achieve this, we used the *OpenCV* Python library [7].

### 5.2. Slant

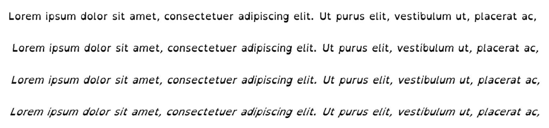

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,

Figure 8: Slant with different tilt

To create a slant effect on the images, we use an affine transformation of the same image, where we set different slant angles to produce different scale effects. To achieve this, we used the *Pillow* Python library [4].

### 5.3. Superimposition

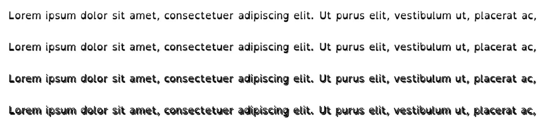

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac,

Figure 9: Superimposition with different distances

This kind of effect is when two images are placed one over the other. To create this effect we generate from the original a background and a foreground image. For the first one, we add a white padding on the top and on the left side. For the second one we first convert it from RGB to RGBA, and then we set the opacity of white pixels to zero. Finally we paste them together. To create different scale of superimposition, we change the padding size, producing a different offset between background and foreground. To achieve this, we used the *Pillow* Python library [4].

## 6. Results

We now present the results obtained from our experiments. We first present the distance metric used to compute the effective difference between two strings of text. In our case, the prediction produced by Tesseract OCR on the image and the respective ground truth.

### 6.1. Levenshtein Distance

$$\operatorname{lev}(a,b)=\begin{cases}|a| & \text{if } |b|=0,\\ |b| & \text{if } |a|=0,\\ \operatorname{lev}(\operatorname{tail}(a),\operatorname{tail}(b)) & \text{if } a[0]=b[0]\\ 1+\min\begin{cases}\operatorname{lev}(\operatorname{tail}(a),b)\\ \operatorname{lev}(a,\operatorname{tail}(b))\\ \operatorname{lev}(\operatorname{tail}(a),\operatorname{tail}(b))\end{cases} & \text{otherwise.}\end{cases}$$

Figure 10: Levenshtein distance.

The Levenshtein Distance is a measure of the similarity between two strings. The distance is the number of the deletions, insertions or subtitutions required to transform te source string into the target string. Obviusly the greater the Levenshtein distance the more different the strings are [10]. The implementation of the algorithm that we used to permorm the distance is provided by the *Jellyfish* Python Package [8].

### 6.2. Comparison

So we computed the distance between the OCR prediction and the ground truth for every single image. We do this for the unaltered images and also for every image generated from the distortion of the original one. Then we store the results in CSV files. The choice of computing the comparison of the unaltered images too, derives not only to estimate the capacity of predicting the correct results of our models, but also to comprehend how much the model is invariant to the distortions applied. To achieve this, we also compute the mean value of the distances obtained from our precedent step and we collect them in an other CSV file. We do this for every font and every different distortion parameter.

## 7. Data results

In Figure 11 we represent our collected data.

For the first row, we present the mean values of the distance between the prediction and the ground truth. Each one of the three figures shows one of the distortions for each category of the typefaces.

In the next rows we reproduce the same results but grouped by category, being the second for the Dyslexic, the third for the Sans Serif and the last for the Serif ones.

### 7.1. Blurring

Between all the distorsions, blurring is the one that produced the best results of prediction. We can observe that Dyslexic and Sans Serif typefaces performs well. On the other hand, Serif ones do not achieve comparable results, even with low distortion parameters.

This supremacy of the first two categories may be referred to the inability of Tesseract to recognize the features that represent the Serif typefaces, compromising the OCR task. In our opinion with the blurring of images we obtain a smoother letterform, that hides the main trait of the Serif typefaces, i.e. the decorative stroke at the end of the letterform.

We can also observe that Sylexiad is the worst of three Dyslexic typefaces and it is the main contributor to the worsening of the category performances. As a matter of fact, it is even worse than most of the Sans Serif typefaces despite Open Dyslexic and Lexie Readable are always better.

### 7.2. Slant

Unlike blurring, the worst performances for this distortion are obtained with the Sans Serif. Even if Dyslexic typefaces seems to perform well with a low distortion parameter, we have a drastic change with the increasing of the angle of the slant.

With Serif typefaces despite what we could expect observing the precedent case, we have comparable results with Dyslexic ones, even if slightly worse. This may be caused by the unaltered presence of the decorative strokes at the end of the letterform. Comparing this fact with the case of Dyslexic typefaces, we can imagine that a similar behaviour may be occurring. In fact, the presence of a widening of the stroke at the bottom of the letterform and the discontinuance of the line in certain characters are not influenced by the distortion.

### 7.3. Superimposition

Even in this case, Dislexic typefaces perform slightly better for every scale of distortion. The best typeface seems to be Open Dyslexic, with Verdana, even if a Sans Serif, immediately follows.

What we observe is that certain typefaces, like Tahoma, present an accentuated closeness between characters. In our opinion this may be a cause of the worsening of performances.

## 8. Conclusions

As we expected, observing the results we obtained we can affirm that Dyslexic typefaces perform generally better than the other categories. Specifically we can observe that the greater is the distortion, the greater is the gap in performances between the Dyslexic typefaces and the non-Dyslexic ones. The best performances of the Dyslexic ones are obtained computing the prediction of the blurred images with respect of the other categories. It seems that no Dyslexic typeface is overally better than others: only Sylexiad appear to be slightly worse than the other two.

We also observed that some Sans Serif obtain good results, like Verdana. This was in our expectation, as it is affirmed by the British Dyslexia Association [1]. It acknowledges that some Sans Serif typefaces can appear less crowded, leading to a better readability.

This work is just the tip of the iceberg. In future research, we suggest to further investigate with a wider range of distortions to simulate visual processing disorders. This could open several perspectives in dyslexia research. The final goal could be the refinement and the understanding of what makes a typeface well designed for dyslexic readers and it could improve our ways to detect this type of disorder.

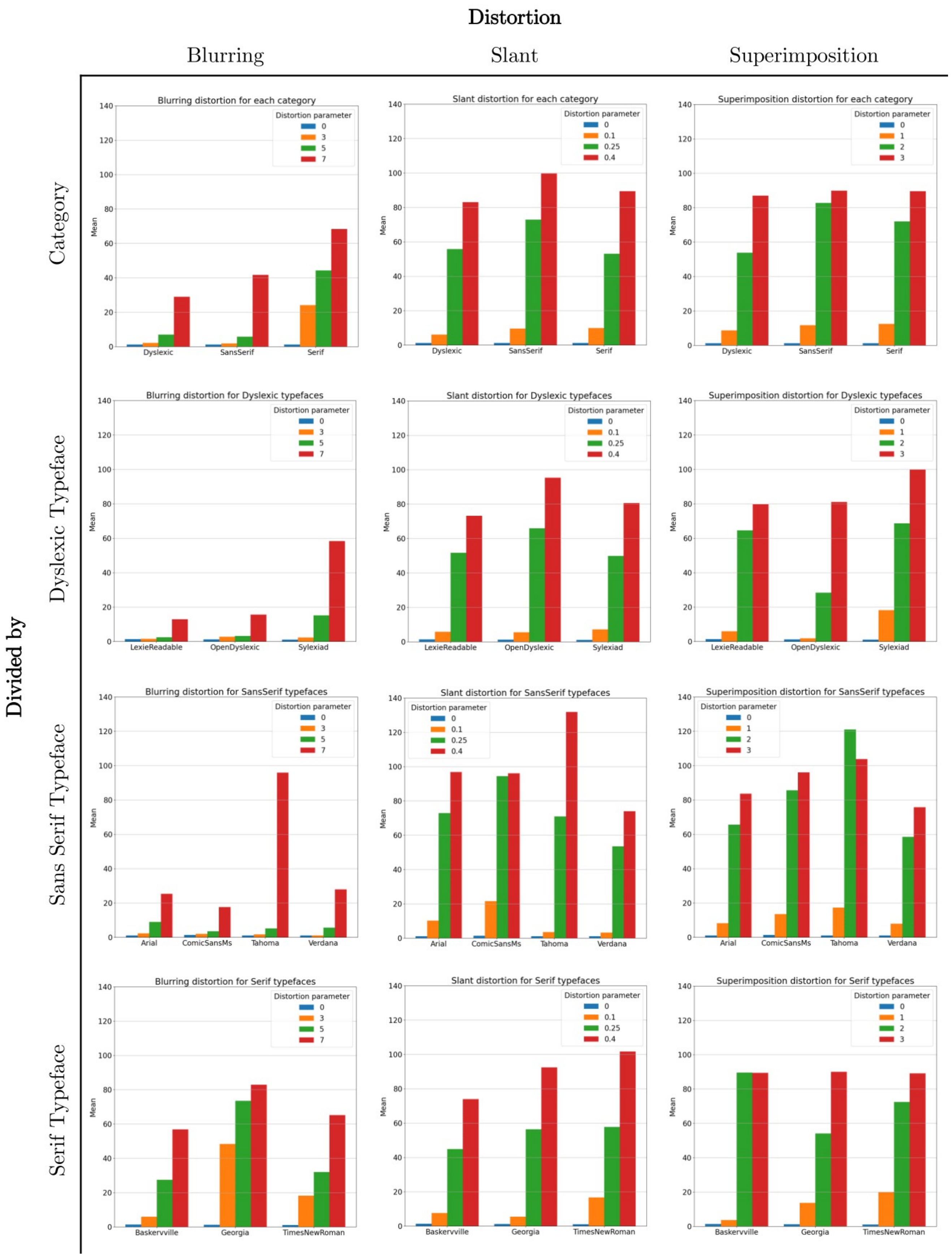


Figure 11: Bar graphs representing the mean values of the distances.

## References

[1] British Dyslexia Association. Dyslexia style guide. `https://www.bdadyslexia.org.uk/advice/employers/creating-a-dyslexia-friendly-workplace/dyslexia-friendly-style-guide`.

[2] Michael Bates. Dislexia font and style guide. `https://www.dyslexia-reading-well.com/dyslexia-font.html`.

[3] Thomas M Breuel, Adnan Ul-Hasan, Mayce Ali Al-Azawi, and Faisal Shafait. High-performance ocr for printed english and fraktur using lstm networks. In *2013 12th international conference on document analysis and recognition*, pages 683–687. IEEE, 2013.

[4] Alex Clark. Pillow library. `https://pypi.org/project/Pillow/`.

[5] CTAN. Fontspec library. `https://ctan.org/pkg/fontspec`.

[6] DesignMantic.com. Dyslexia-friendly fonts to facilitate learning and reading. `https://www.designmantic.com/community/dyslexia-friendly-fonts-for-better-learning.php`.

[7] Olli-Pekka Heinisuo. Opencv library. `https://pypi.org/project/opencv-python/`.

[8] jamesturk. Jellyfish python package. `https://pypi.org/project/jellyfish/`.

[9] Benjamin Kiessling. Kraken library. `https://pypi.org/project/kraken/`.

[10] Merriam Park Software Michael Gilleland. Levenshtein distance, in three flavors. `https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm`.

[11] Yusuke Shinyama. Pdfminer python package. `https://pypi.org/project/pdfminer/`.

[12] Ray Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[13] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[14] zdenop. Train tesseract lstm with make. `https://github.com/tesseract-ocr/tesstrain`.

[15] Xinru Zhu, Kyo Kageura, and Shin'ichi Satoh. Analysis of typefaces designed for readers with developmental dyslexia. In *International Workshop on Document Analysis Systems*, pages 529–543. Springer, 2020.