ReadBlind: Text-Image to Speech

Konstantina Timoleon, Rupal Saxena

October 15, 2023

Abstract

ReadBlind is a Text-Image to Speech pipeline which was developed to assist individuals with visual impairments. The pipeline involves the extraction of textual content from images, followed by the conversion of this text into speech. The pipeline encompasses two distinct text-detection setups: Scene-Text Detection and Handwritten Text Detection. Following the detection of text, a simple text-to-speech model is utilized to convert the detected text to speech.

The code developed for this project can be found on Github.

Contents

1 Introduction

In the realm of technological advancement, there exists substantial potential for enhancing the quality of life for individuals with disabilities. With this overarching objective in mind, we have devised a pipeline that accepts images as input and produces synthesized speech as output. This innovation aims to support individuals with varying degrees of visual impairment, enabling them to access textual information contained within images. Nevertheless, this endeavor is not without its inherent challenges. The text of interest may be situated within diverse environmental contexts, encompassing both outdoor and indoor settings, and may also manifest in the form of handwritten script. To address these dual challenges comprehensively, we have meticulously engineered a pipeline that adeptly addresses both scenarios. This comprehensive solution comprises two distinct components: a Scene-Text Detection pipeline and a Handwritten Text Detection pipeline. Both of these pipelines are designed to extract and process textual information, which is subsequently transformed into synthesized speech using Text-to-Speech algorithms.

2 Dataset

2.1 TextOCR Dataset: Scene-Text Detection

For Scene-Text Detection we used the TextOCR dataset [2] which contains almost 30000 JPG images, complete with annotations. The images are of ordinary scenes that one would encounter in their everyday life which also have some text on them, e.g. road signs, menus, billboards. A few examples of the contents of the dataset can be found in Figure 1.

Figure 1: TextOCR dataset sample images.

Such a dataset is suitable for the nature of the pipeline we're trying to implement since it corresponds exactly to the challenges that visually impaired people have to tackle daily. We will however revisit the suitability of the TextOCR dataset for our pipeline when running model evaluations. Regarding the annotations, they contain both the ground-truth text as well as the bounding boxes surrounding each word. As a final remark, we need to point out that the annotations are mainly in English and can also contain punctuation.

2.2 IAM Handwriting Database: Handwritten Text Detection

For the Handwritten Text Detection task, we chose the IAM Handwriting Database [1]. The IAM Handwriting Database contains forms of handwritten English text which can be used for text recognition tasks, making it a perfect choice for us. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. Figure 2 provides samples of a complete form, a text line and some extracted words. For this project, we choose images of text lines, as shown in Figure 3.

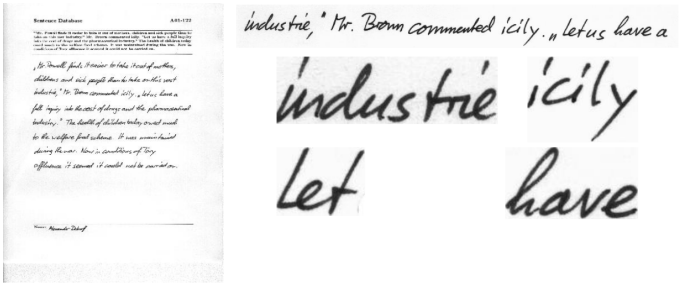


Figure 2: IAM dataset sample image.

Figure 3: IAM dataset text line sample image.

3 Methodology

This project comprises of three elements: Scene-Text Detection, Handwritten Text Detection, and Text-to-Speech Conversion. We chose to use different approaches for the two separate text detection tasks: for Scene-Text Detection we will run and compare existing Python libraries whereas for Handwritten Text Detection we will fine-tune an existing model on the IAM dataset. The comprehensive workflow illustrating the methodology employed is presented in Figure 4.

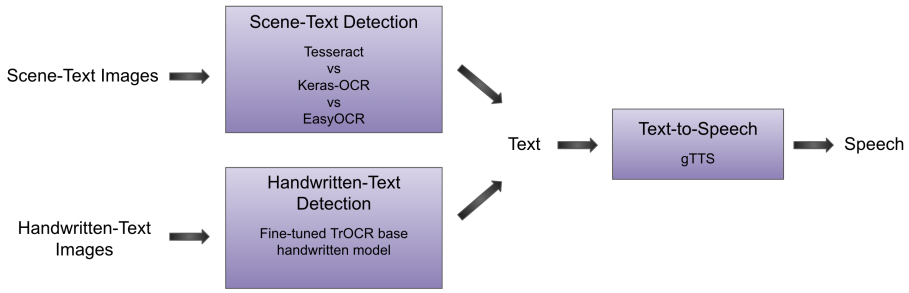


Figure 4: Pipeline workflow

3.1 Scene-Text Detection

We employed existing optical character recognition (OCR) methods that recognize and read the text embedded in images. In particular, we used and compared three OCR tools: Tesseract, EasyOCR and Keras-OCR. Our decision to work with the aforementioned methods was based on the fact that they all have readily available Python libraries that are easy to install and import. Specifically, we chose Tesseract because it is lightweight and EasyOCR and Keras-OCR because they are heavier but also more accurate. All in all, the goal was to test the capabilities of different, publicly available models, with varying degrees of complexity, that were developed for the same task: optical character recognition. Since for this part we are only evaluating existing methods and not training a model from scratch, we will use instances of the training set, that also include annotations (whereas the test set annotations are not publicly available).

3.1.1 Postprocessing

Each of the three methods we employed yields the detected bounding boxes with different formatting, e.g, varying ordering of the coordinates (clock- or counter-clockwise), different

data types, etc. In order to be able to visualize the results and compare the three methods with each other as well as against the ground truth, we had to transform the coordinates.

3.2 Handwritten-Text Detection

To perform Handwritten Text-Detection, we leveraged the IAM Handwriting Database and fine-tuned the HuggingFace TrOCR base handwritten model using this dataset.

3.2.1 Preprocessing

We conducted dataset preprocessing as an initial step before inputting it into the neural network. In total, 336 training images were employed, with 85% allocated for training and the remaining 15% for validation. To encode text labels, we utilized a pretrained tokenizer based on Huggingface's TrOCR model [4], which employs the WordPiece tokenization method.

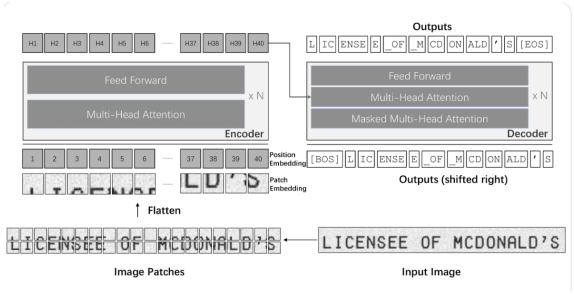


Figure 5: TrOCR model architecture

3.2.2 Model training

We fine-tuned HuggingFace TrOCR base handwritten model with IAM Handwriting Database. For training, we opted for the AdamW optimizer and conducted training over 20 epochs. Due to GPU limitations, we worked with a reduced dataset. We selected the HuggingFace TrOCR base handwritten model as our pretrained network, given its prior training on handwritten data. Given the constraints of our smaller dataset for fine-tuning, this choice proved to be a suitable one for our model. The architecture of TrOCR from the original paper is shown in Figure 5.

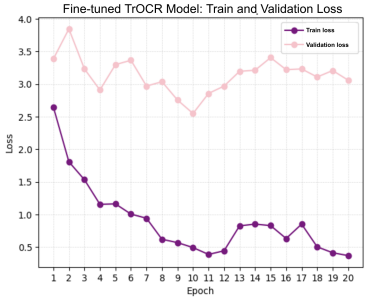


Figure 6: Loss values for the training and validation sets, computed over each epoch for the fine-tuning of the TrOCR Model.

Figure 6 the loss curve observed during our model's training. Notably, the validation loss consistently surpasses the training loss throughout the training process, indicating the presence of overfitting. The underlying cause is clear: our training dataset is exceedingly small. To mitigate this issue, it is imperative to employ a larger dataset.

3.3 Text-to-Speech Conversion

We employed the Google Text-to-Speech (gTTS) Python library to transform identified text into speech.

4 Evaluation

4.1 Scene-Text Detection

We measure the performance of each method according to whether it successfully identifies the bounding boxes of the text in each image. This is achieved by calculating the Intersection over Union (IoU) between a method's detected bounding box and the ground truth (see Appendix A for further information).

Average IoU scores	
Library	**Score**
Tesseract	0.011
EasyOCR	0.075
Keras-OCR	0.089

Table 1: Average IoU scores calculated over 500 annotated images, for each of the libraries used.

Due to limited compute power, we chose 500 images at random out of the 20000 images in the original train set to evaluate the three methods on, such that all methods are evaluated on the same data. For each image, we go through the annotation boxes and for each annotation box we find the closest box (distance less than 20 units in the Cartesian plane) that each method yields. If such a box exists, we calculate the IoU between the annotation bounding box and the detected bounding box of the method; otherwise we set the IoU to zero. Finally, we average the IoU scores of each method over all 500 images to get an overview of the methods' performance.

The results, which can be found in Table 1, hint to what we already suspected: more sophisticated methods (EasyOCR, Keras-OCR) lead to better performance compared to the more lightweight Tesseract. However, if we take into account that the IoU values run between 0 and 1, the overall performance does not seem to be satisfactory. Such behavior can be attributed to two main reasons: the first one relates to the dataset and the second one to the evaluation method. Pertaining to the dataset, we refer the reader to Appendix F which showcases that a good portion of the images contains text annotations that would not be visible or decipherable by the naked eye, thus creating adverse conditions for the OCR methods to work with. On the other hand, the evaluation technique is probably too strict since it penalizes cases where the detected bounding box deviates from the annotation in terms of size and orientation, even though such parameters don't necessarily interfere with the quality of text detection (also visible in Appendix F).

Average IoU scores		
Library	**Score**	**CPU Run time[s]**
Tesseract	0.089	3.3
EasyOCR	0.146	40.5
Keras-OCR	0.167	55.4

Table 2: Average IoU scores calculated on Figure 7, for each of the libraries used.

Taking all this into account, we carefully chose a test image to include in our pipeline. We concluded that Figure 7 poses a fair challenge to the available methods while still corresponding to a realistic setting where ReadBlind might come in handy. The performance of the three methods on this specific image can be found in Table 2. For the visualization

Figure 7: Test image for Scene-Text Detection

of the detected boxes and text we refer the reader to Appendices C, D and E where it is evident that the results for the EasyOCR method are way better than what the IoU score suggests. In the end, we decided to integrate EasyOCR into our pipeline because it has acceptable performance, and it's not as computationally intensive as Keras-OCR.

4.2 Handwritten-Text Detection

We evaluated our model's performance using the Character Error Rate (CER) as our primary evaluation metric. This choice was driven by the widespread utilization of CER in the evaluation of optical character recognition (OCR) systems and analogous text recognition tasks. It represents the ratio of characters that were incorrectly recognized or transcribed compared to the total number of characters in the reference or ground truth text. The CER of our Handwritten-Text Detection model is 0.425. In Figure 8 we show the output of our model, given an input.

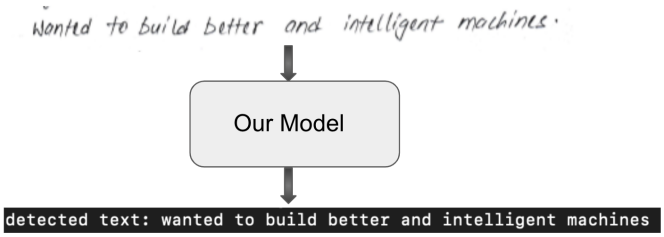


Figure 8: Detected text output given an image of handwritten text, using our the fine-tuned TrOCR Model.

5 Future work

As a general remark, enhanced GPU resources could be harnessed in the future for training on a larger and more varied dataset. ReadBlind could also evolve in a unified pipeline that works for both text detection modes in order to maximize its usefulness. Regarding Handwritten Text Detection, the dataset we utilized features a white background. To enhance the model's versatility and applicability across various scenarios, a more diverse dataset with different backgrounds could be considered. Finally, regarding Scene-Text Detection, adding a second level to the evaluation, where we also use the CER metric along with the IoU, would provide a holistic overview of the performance, going further than just evaluating object/text detection capabilities. Moreover, coming up with method-specific comparison algorithms that account for the different types of bounding boxes that each method produces, would be a fairer way to compare a method against the annotations.

References

[1] Urs-Viktor Marti and H. Bunke. "The IAM-database: An English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition* 5 (Nov. 2002), pp. 39–46. DOI: 10.1007/s100320200071.

[2] Amanpreet Singh et al. "TextOCR: Towards Large-Scale End-to-End Reasoning for Arbitrary-Shaped Scene Text". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 8802–8812.

[3] Medium Analytics Vidhya. *IOU (Intersection over Union)*. 2021. URL: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef.

[4] Minghao Li et al. "TrOCR: Transformer-Based Optical Character Recognition with Pre-trained Models". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.11 (June 2023), pp. 13094–13102. DOI: 10.1609/aaai.v37i11.26538. URL: https://ojs.aaai.org/index.php/AAAI/article/view/26538.

A Intersection over Union

The Intersection over Union (IoU) between two overlapping boxes is defined as follows:

$$IoU = \frac{\text{Area of intersection of two boxes}}{\text{Area of union of two boxes}}.$$

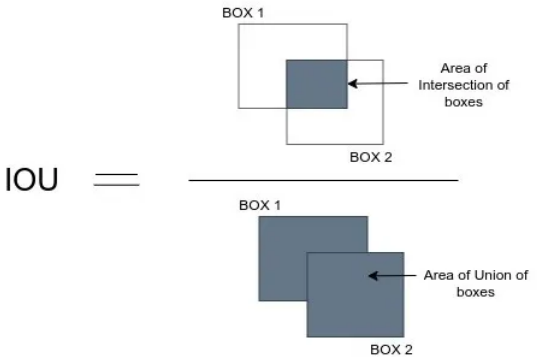


Figure 9: Diagrammatic representation of the formula to calculate IoU. Source [3]

The IoU of two boxes can have any values between 0 and 1. In case that the boxes are not in fact overlapping, the area of their intersection would be 0, and therefore the IoU would also be 0. This metric is mainly used in applications related to object detection, where we train a model to output a box that fits perfectly around an object.

B Python Libraries

B.1 Scene-Text Detection

We used the pytesseract, easyocr and keras-ocr libraries for Tesseract, EasyOCR and Keras-OCR respectively. We used shapely to handle boxes and polygons with ease and also matplotlib, random, numpy, tqdm and pandas.

B.2 Handwritten Text Detection

The following Python libraries were used for Handwritten-Text Detection pipeline: transformers, jiwer, evaluate, pandas, torch, sklearn, tqdm, PIL.

C Tesseract

Figure 10: Scene-Text Detection using Tesseract. Detected text (top) and detected bounding boxes (bottom).

D EasyOCR

Figure 11: Scene-Text Detection using EasyOCR. Detected text (top) and detected bounding boxes (bottom).

E Keras-OCR

Figure 12: Scene-Text Detection using Keras-OCR. Detected text (top) and detected bounding boxes (bottom).

F More results

Figure 13: Sample 1: Detected bounding boxes on train images from the TextOCR dataset using Tesseract (blue), EasyOCR (red) and Keras-OCR (yellow), along with ground-truth (green).

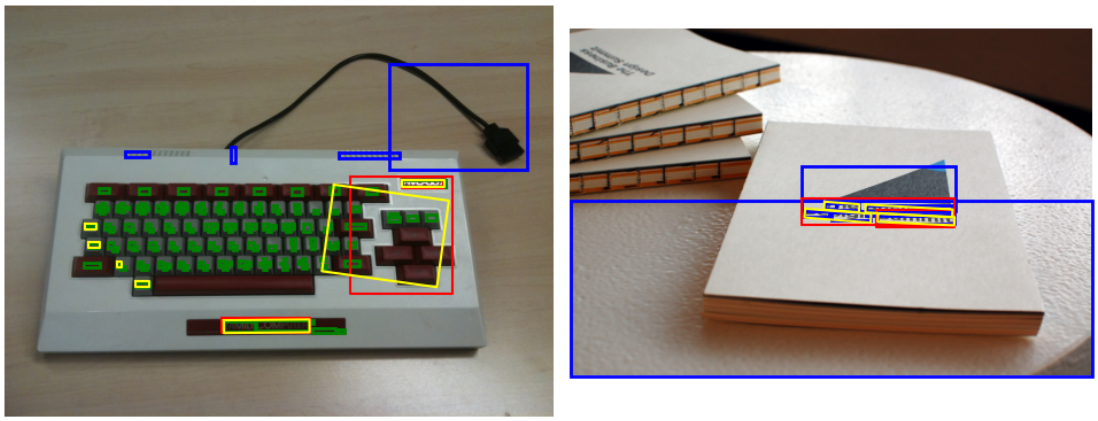


Figure 14: Sample 2: Detected bounding boxes on train images from the TextOCR dataset using Tesseract (blue), EasyOCR (red) and Keras-OCR (yellow), along with ground-truth bounding boxes (green).

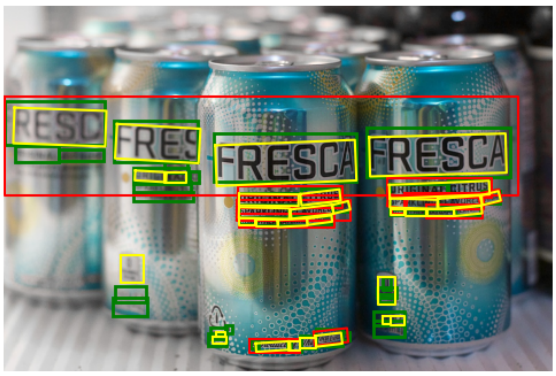
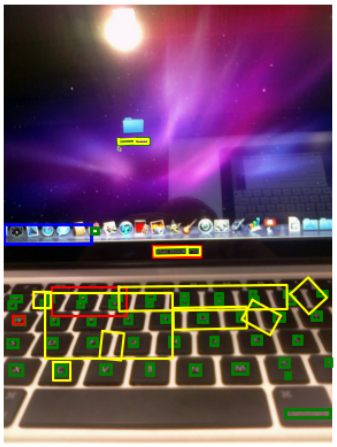


Figure 15: Sample 3: Detected bounding boxes on train images from the TextOCR dataset using Tesseract (blue), EasyOCR (red) and Keras-OCR (yellow), along with ground-truth bounding boxes (green).



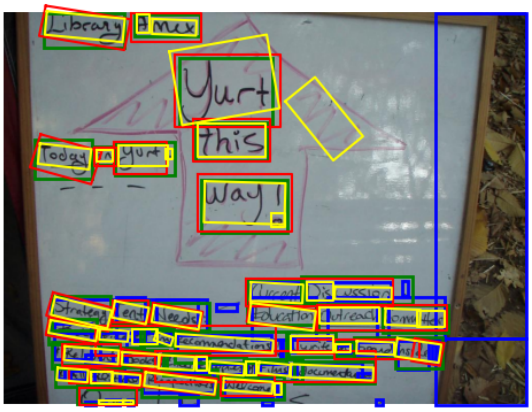


Figure 16: Sample 4: Detected bounding boxes on train images from the TextOCR dataset using Tesseract (blue), EasyOCR (red) and Keras-OCR (yellow), along with ground-truth bounding boxes (green).

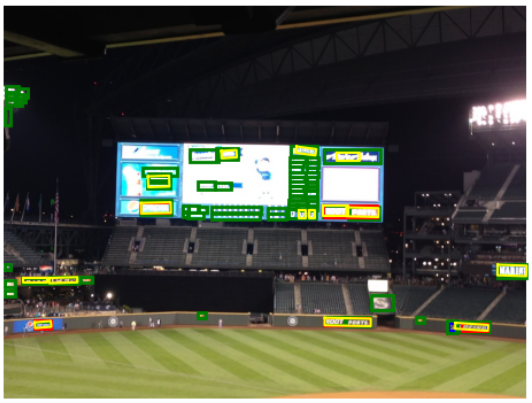


Figure 17: Sample 5: Detected bounding boxes on train images from the TextOCR dataset using Tesseract (blue), EasyOCR (red) and Keras-OCR (yellow), along with ground-truth bounding boxes (green).