Handwriting recognition & Auto Completion

Abhilekkh Krishna[1] and Abhigyan Krishna[2]

[1] krishnabhilekkh@gmail.com
[2] abhigyan01krishna@gmail.com

Abstract. In the digital age, handwriting remains crucial for expressing illustrations and formulas despite the prevalence of keyboard input. However, the diverse nature of handwriting poses challenges for computer processing, limiting its portability and input assistance. To address these issues, Handwriting Recognition technology has been developed, gaining traction with electronic pens and tablets. This report explores the application of Handwriting Recognition on iOS devices, emphasizing the growing need for improved recognition capabilities. The report recognizes the evolving landscape and proposes auto-completion to enhance handwriting input convenience. The study demonstrates that auto-completion can reduce the number of characters required for a word entry, mitigating input challenges. Additionally, a technique is proposed to minimize computational demands on devices. Two metrics are introduced to evaluate auto-completion performance, comparing outcomes between utilizing iOS's text recognition API and a custom Handwriting Recognition model designed for computational efficiency. This research aims to unlock the full potential of electronic tablets as a medium for handwriting, addressing current limitations and fostering advancements in Handwriting Recognition applications.

1 Introduction

Handwriting has played a significant role as a means of human knowledge preservation or communication from the invention of letters to modern times. Its role has been reduced with the spread of printing and computers, but in many parts of our lives, I still have the opportunity to write and read handwriting. For Example, Students in a classroom still prefer to use paper and pen or electronic tablet to store knowledge, mathematical formulas, graphs, etc., rather than taking notes using a keyboard and computer. On the other hand, in computer work such as searching or document writing, input is almost always performed with a keyboard, and information written by hand is rare. This is, of course, because the computer cannot handle handwritten input as text as it is, but also because the keyboard is more efficient for inputting characters. With a keyboard, I can enter characters with less movement than handwriting. Besides, there are also assisting functions for keyboard input, such as copying and pasting, searching, or formatting. Since keyboard input has these advantages, people usually try to use a keyboard even for easier tasks and a tablet device for tasks that are not achievable with the keyboard. However, this situation shows signs of change with the

spread of tablet devices in recent years. On these devices, input work that keyboards are not good at, such as drawing illustrations and writing mathematical formulas, can be more accessible. In addition, applications [3][4] have appeared to handle handwriting as text format by applying Handwriting Recognition technology that has been studied for many years. Thanks to these technological improvements, writing is becoming an available option for inputting text, but it is not yet as efficient as a keyboard. One of the reasons for this is the lack of input assistance. Therefore, in this work, I propose an application that combines Handwriting Recognition and auto-completion to assist handwriting and to enable it to be treated as text format on devices.

Our application consists of two elements: Handwriting Recognition and auto-completion. Handwriting Recognition (HWR) is a technology that interprets handwritten inputs from sources such as papers, photographs, electronic tablets, and other devices into a format computers can easily handle that. In most of the modern work of HWR, the process pipeline is divided into Handwritten Text Detection and Handwritten Text Recognition [11]. The processing of text detection tends to be complicated[11], and it takes considerable time to perform processing on the entire input image. It is known [5] that longer processing time in applications that require a quick response, such as auto-completion, can seriously degrade the user experience. Therefore, I propose a method to avoid performing Text Detection using a simple heuristic: Region of Interest (ROI) detection.

Auto-completion is a program that predicts the rest of the word a user is typing based on the letters entered or the recent word pairs [2]. It is widely used in web browsers, email programs, source code editors, and word processors. If the correct word is included in the predicted words list, the user can save time to enter the rest of the word. In this work, to make the process simple, I used a forward forward-matching algorithm for this purpose.

The following section introduces typical approaches of HWR and related works. Section 3 provides technical details of the process the problem addressed in this report. Section 4 describes the result result process and discusses that. Section 5

2 Related Work

2.1 Handwriting Recognition Taxonomy

HWR can be roughly divided into two approaches: online approach and offline approach [14]. While the online approach uses information on the trajectory of the pen tip obtained from a special pen for classification, the offline method uses optically scanned images as input and performs recognition using computer vision techniques. In this work, for simplicity of the pipeline, I focus only on the offline approach.

[3] https://www.nebo.app/
[4] http://mazec.jp/
[5] http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

The offline approach of HWR can be positioned as one variant of Scene Text Detection / Recognition, which is a technology to extract and recognize text information written in natural images. Due to the recent development of Neural Networks technology, much research has been done in this field to this day. Except for few methods[10][12], most approaches of Scene Text Detection / Recognition separate text detection and text recognition and perform stepwise inference.

2.2 Detection

Scene Text Detection can be subsumed under general object detection; those methods usually follow the same object detection procedure is dichotomized as one-stage methods and two-stage ones [8]. After the emergence of FasterR-CNN[17], most of the modern text detection algorithms are based on FasterR-CNN, YOLO[16], SSD[9].

In addition to the general object detection model, text detection models are devised to detect tilted bounding boxes[19][7] and character regions of arbitrary shapes[18], or to simplify the pipeline[6], since pipeline of text detection tends to be complicated[8].

2.3 Recognition

Some text recognition algorithms divide the task into character segmentation and character recognition [1][13]. Character segmentation is considered the most challenging part of scene text recognition and may affect overall accuracy. It is tough to segment connected characters such as cursive. Therefore, some techniques that do not rely on character segmentation have been developed. This report introduces a method called Connectionist Temporal Classification (CTC) [5].

CTC was first proposed to handle sequence labeling of arbitrary length, requiring no pre-segmented training data. A CTC network outputs probabilities for each label at each time step. The time step length can be any length longer than the label length. The output at each time step is the probability of the classes to be recognized plus the extra class representing "blank." Let this output probability be $\mathbf{y} = (y_1, y_2, \cdots, y_w)$ and denote by y_k^t the activation of label k at time step t. Given this probability distribution, the conditional probability of the sequence is calculated as follows.

$$p(\pi|\mathbf{y}) = \prod_{t=1}^{w} y_{\pi_t}^t \tag{1}$$

Then a many-to-one mapping $\mathcal{B}$ is defined to transform the sequence π to a shorter sequence. This mapping obtains the final predicted label. This mapping removes all blanks and repeated continuous labels from the sequence. For Example, $\mathcal{B}$ maps the predicted sequence "aa-p-pl—-ee" to "apple," where "-" represents the "blank." Since this mapping is a many-to-one, different sequences may be mapped to the same sequence. Therefore, the probability of the final

output sequence is the sum of all possible conditional probabilities of all π corresponding to that final sequence.

$$p(l|\mathbf{y}) = \sum_{\pi} p(\pi|\mathbf{y}) \tag{2}$$

where π represents all π which produces $l = \mathcal{B}(\pi)$.

The classifier's output should be the most probable labeling for the input sequence.

$$h(\mathbf{y}) = \arg\max p(l|\mathbf{y}) \tag{3}$$

In general, there are many mapping paths for a given sequence. Thus calculation of arg max requires heavy computation. In practice ng two approximate methods are known to give us a good result.

The first method is based on the assumption that the most probable path can be approximated by the sequence of most probable labeling

$$h(\mathbf{y}) \approx \mathcal{B}(\pi^*) \tag{4}$$

Wherefore π^* is a set of labethatich get the highest probabilities at each time step. Although it works well, it is not guaranteed to get the most probable labeling.

The second method uses a forward-backward algorithm to efficiently search for the most probable sequence. With enough time, this approach can always find the most probable labeling from the input sequence, but the amount of computation increases exponentially concerning the sequence length, so it is not practical to find the exact solution.

The maximum likelihood approach is utilized to train the network with the dataset $\mathcal{D} = \{I_i, l_i\}$, where I_i represents the input image and l_i represents the corresponding label. The objective function of this can be negative log-likelihood

$$\mathcal{O} = -\sum_{(I_i, l_i) \in \mathcal{D}} \log p(l_i|\mathbf{y}_i) \tag{5}$$

where $\mathbf{y}_i = f(I_i)$ and $f(\cdot)$ represents the classifier. Stochastic Gradient Descent (SGD) can be used to minimize negative log-likelihood. To summarize, a text recognition model with CTC can be obtained by defining a network that outputs a sequence longer than the required label length and training the network to minimize the loss function (**??**). After the fitting, the model outputs a sequence of probabilities of labels for a given input. Each time step of the sequence corresponds to, if the input is an image, image patches arranged in the direction of writing. I then get the final prediction by putting the output sequence into a many-to-one mapping. There are several options for this many-to-one mapping. Still, the one to get the highest probabilities at each time step and the one that uses a forward-backward algorithm is considered to achieve good performance in practice.

3 Methods

As in the case of Scene Text Detection / Recognition, it is adequate to separate Text Detection and Text Recognition and treat them as different problems. Furthermore, it is possible to record the written area due to the characteristics of the electronic tablet, so using a simple heuristic on the trajectory data eliminates the need to perform text detection. The role of this step, namely the "Region of Interest Detection" step, is to reduce the size of data passed to the subsequent processing and suppressed the increase in the amount of calculation. Detected regions are then preprocessed and passed to the text recognition module. In the current implementation, illustrations and handwritten characters cannot be separated, so the processing is passed to the handwriting recognition module even if the user writes pictures. Still, improvements in this part are discussed in Section **??**. In the text recognition module, two patterns of recognition using CTC and credit using the API provided by Apple were verified.

3.1 Region of Interest Detection

The purpose of Region of Interest (ROI) detection is to cut out sub-sequence representing words and illustrations from the dot sequence of the pen tip trajectory and cut back the data to be passed to the subsequent processing to reduce the amount of calculation and improve the recognition accuracy. A region of interest includes a dot sequence of the word or the illustration. To specify the region, two assumptions are made: region of interest.

1. Objects drawn in the ROI are close in time when drawn
2. Objects drawn in the ROI are spatially close

Based on these assumptions s, the sequence contained in ROI is detected by the following algorithm 1. T, hen the smallest rectangle containing the obtained point sequence is the ROI. It should be noted that I prioritize the reduction of the amount of computation and terminate the search for subsequences on the way. As a result, points that are not continuous in time but close in space may not be included in the ROI, depending on the threshold setting. However, when priority is given to spatial proximity, it is necessary to parse the entire trajectory every time I perform ROI detection. If the size of the point sequence becomes large, it may be a bottleneck in the calculation. Therefore, I decided to use this methwhicht it should be improve in future work.

Figure 1 shows an example of the region detected with this heuristic.

There are some cases where such heuristics do not work. For Example, when the scale of the depicted object is large, the gap between the sequence of points constituting the things may be too large and fail.It is also a significant problem that temporal and spatial proximity depends on parameters and sometimes does not match intuition. However, this method can certainlyn narrow down the target area for text recognition with a minimal amount of computation, so this report adopted this method.

Algorithm 1: ROI detection

Input: $S = \{s_1, s_2, \cdots, s_n\}$ *where* $s_i = (c_i, t_i)$
Output: $\sigma = \{s_k, s_k + 1, \cdots, s_n\}\ 1 \leq k \leq n$

$\sigma := \{s_n\}$
$c_{prev} := c_n$
$t_{prev} := t_n$
for s_i *in reversed*$(S) \setminus s_n$ **do**
    $c_i, t_i \leftarrow s_i$
    if $t_{prev} - t_i < t_{threshold}$ **then**
        $\sigma \leftarrow \sigma \cup \{s_i\}$
    else
        if *distance*$(c_{prev}, c_i) < c_{threshold}$ **then**
            $\sigma \leftarrow \sigma \cup \{s_i\}$
        else
            break
    $c_{prev} \leftarrow c_i$
    $t_{prev} \leftarrow t_i$

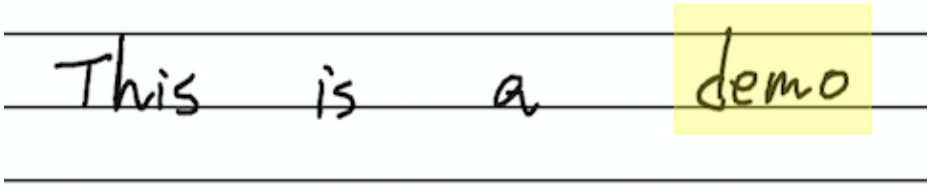

Fig. 1. The region with yellow overlay is detected region of interest

3.2 Recognition

In text recognition, two models were tried: a model that directly reads the content from the result of region of interest detection using CT, and the model which is provided by Apple[6]. Note that Apple's VNRecognizeTextRequest API does not disclose its implementation or the model used insidt is unknown exactly what processing is being done.

CTC CTC models are usually constructed usthe ing CNN model with Recurrent Neural Network (RNN) as the output layer. However, RNN performs computations slowly compared to CNN, and in recent y,ears its effectiveness in CTC is sometimes questioned [15]. Given that all calculations are done on iOS, and based on research that the output layer of the CTC model can also be configured using CNN [4], in this re,port a mothat is fully constructed only by CNN was used the for CTC model. = To get an approximation the for most probable labeling, greedy policy, the policy to take the most probable label at each time tep, was used.

VNRecognizeText API VNRecognizeText API is an API provided by Apple officially, and its function is to detect text from the whole area where the drawing is and read it. Therefore,e there's no need to detect the region of interest to read the content inside. However, I used ROI detection to preprocess the inference fast before putting an image into the API to make the inference fast. This API is entirely a black box accepts an image of any shape as input, reads the text it and outputs it with position information.

VNRecognizeText API has two modes for recognition, namely '.fast' and '.accurate'[7]. According to the official introduction video[8] if VNRecognizeText API, '.fast' ua ses traditional feature-based method inside while '.accurate' ua ses more sophisticated Computer Vision model inside. As the names of those imply '.fast' is faster but not so accurate, rate while '.accurate' is slower but more accurate. Both '.fast' and '.accurate' were tested. However, since '.fast' failed to recognize handwritten letters, measurement on '.fast' was not conducted.

3.3 Auto-Complete

In this report, I deal with auto-completion as an application of the inference result. This auto-completion emphasizes simplicity and presents words that start withinferredddevelopmentt in order of frequency. To get the word candidates who start withinferredddevelopmentt, I created an inverse index that maps the first few letters of words to a group of words with that prefix. For Example, index "elbo" corresponds to word group {"elbo,", "elbow',", "elbowe,," "elbowin,", "elbowroo,", "elbowroom,'", "elbows"}. Word suggestions for auto-completion

[6] https://developer.apple.com/documentation/vision/vnrecognizetextrequest
[7] https://developer.apple.com/documentation/vision/vnrequesttextrecognitionlevel
[8] https://developer.apple.com/videos/play/wwdc2019/234/

are fthe rom ‘wamerican‘ package[9] of Debian GNU/Linux. Word frequencies were counted using wikipedia-word-frequency[10]. I implemented auto-completed each time the pen is released from the tablet surface.

3.4 Dataset

Handwritten character recognition and handwritten sentence recognition are fields that have been studied for a long time, so there are many data sets[11][12][13][3], but these are often provided in different formats, and there is some difficulty in eliminating differences between formats and using them for the training dataset. Therefore, I took an approach to create a composite dataset by embedding a combination of existing handwritten-like fonts with randomly picked font sizes and selected English words in the image. The words are selected from the ‘American‘ package and 10,000 images were generated for training. I split the dataset to 80% / 20% each for training/validation. Figure 2 shows an example of training data generated with this method.

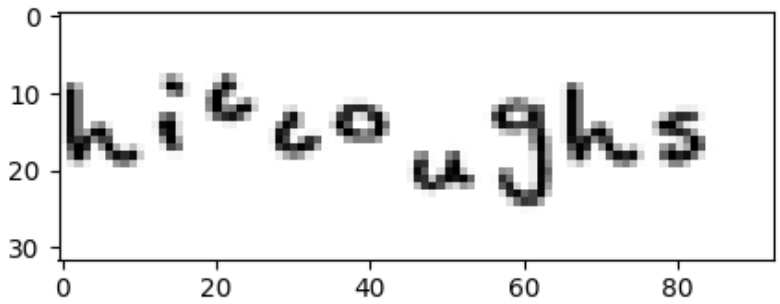

Fig. 2. Generated image using handwritten-like fonts

I only used vertical and horizontal random offset of characters in a word for image preprocessing. Since the image taken from the prototype application has white background with no blurriness, I didn't apply further image augmentation techniques.

[9] https://packages.debian.org/search?keywords=wamerican
[10] https://github.com/IlyaSemenov/wikipedia-word-frequency
[11] https://www.kaggle.com/vaibhao/handwritten-characters
[12] https://www.kaggle.com/ashishguptajiit/handwritten-az
[13] https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format

3.5 Implementation

I used Python3.6[14] and TensorFlow2.1[15] for building machine learning models and training, and used coremltools3.2[16] to convert the models into a format that can work on iOS.For training, Adam optimizer with learning rate 5e-5. 'beta_1' and 'beta_2' parameters are set to 0.9 and 0.999 each. The training was performed in 50 epochs with batch size 32. Due to the limitations of core tools, I fixed the image size to 32x200 and padded the image.

To get handwritten document images from the iOS electronic tablet, a function to print the input from the Apple Pencil[17] on the white canvas according to the path of the pen has been implemented.

For training the model, Google Colaboratory environment [18] has been used.

4 Results & Discussion

4.1 Region of Interest Detection

ROI detection should cut out the region the user focuses on. For Example, when a user writes a sentence, ROI should be a word written, and when a user is drawing an illustration, ROI should be the whole illustration drawn.

Since it is difficult to measure the goodness of the ROI detection quantitatively, only qualitative evaluation was performed this time. When cutting out only the words that the user is writing in the text, ROI detection worked almost without failure (Fig 3).

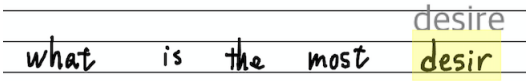

Fig. 3. ROI detection works well when cutting out a word that is being written

On the other hand, when cutting out a handwritten illustration, only a part of the illustration may be cut out if the lines constituting the illustrations are not spatially close to each other (Fig 4). Note that the failure example may seem spatially close inside, but it is not the case when I split it up into a set of strokes and compare the distance between starting point and ending point of strokes.

[14] https://www.python.org/downloads/release/python-360/
[15] https://www.tensorflow.org/
[16] https://apple.github.io/coremltools/
[17] https://www.apple.com/apple-pencil/
[18] https://colab.research.google.com/notebooks/welcome.ipynb

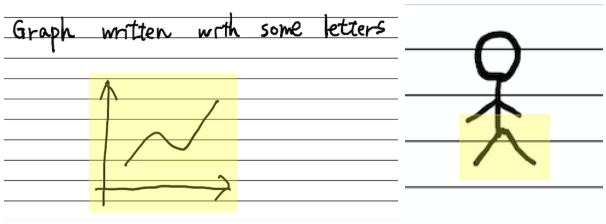

Fig. 4. Left: Example of the case ROI detection worked well on illustration. Right: Example of failure case of ROI detection

4.2 Evaluation of Text recognition

In the previous section, two patterns of text recognition were tried: recognition using CTC and recognition using VNRecognizeText API. Since this report worked on the recognition of handwritten text and made use of the result for auto-complete, the following metrics were designed to measure the goodness of auto-completion.

- Omitted Characters Count (OCC) - The gap between how many characters did the writer actually wrote before he found the word he wanted to write within top 10 of the auto-completion candidates and the number of characters in the word. If it is not found after writing to the end, the score is 0. A higher value of this metric indicates better results.
- Cumulative Time for Inference (CTI) - Cumulative time spent on auto-completion until the word that the writer actually wants to write is included in the top 10 auto-completion candidates. It should be noted that CTI is not proportional to the actual inference time at each time step because the earlier the word is included in the list, the shorter CTI is. If it is not found after writing to the end, the score is cumulative time spent on the recognition at each step. Note that the recognition process runs each time the writer releases the pen tip from the tablet. The lower value of this metric indicates a better result.

100 words were randomly selected from the word list of 'wamerican' package of Debian GNU/Linux for evaluation. Table 1 shows the performance of both methods on auto-completion. While OCC measures of VNRecognizeText API shows a better result than CTC; the cumulative time elapsed for inference is almost double that of CTC.

Figure 5 shows an example of how auto-completion works.

4.3 Discussion

The VNRecognizeText API is superior to CTC in character recognition accuracy, but in terms of speed, it takes about three times longer to infer for each time step. Since auto-completion is an application that requires real-time performance, a

Table 1.

method	OCC (mean)	CTI (mean)
CTC	1.0102	1.1060
VNRecognizeText (.accurate)	3.3405	1.9751

Perfromance of CTC and VNRecognizeText API on auto-completion

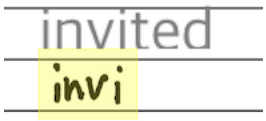

Fig. 5. Example of auto-completion showing the top 1 candidate

small lag does not provide a good user experience. Therefore, poor performance in speed can be a problem.

On the other hand, the CTC model shows inference speed that does not make the user feel uncomfortable in actual use, but the inference result is unstable, and a slight difference in notation greatly affects the inference result. Figure 6 shows an example of unstable inference. Considering the fact that the average length of the experimented words is 7.96, the performance of VNRecognizeText is promising since more than 40% of the word length can be potentially omitted by auto-completion.

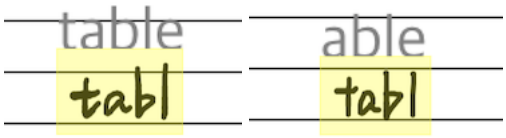

Fig. 6. Example of unstable inference

This is probably because training of the CTC model seems to be overfitting and strongly depends on the dataset used for training. Therefore, future tasks include reducing the reliance on datasets and using larger datasets or using data augmentation techniques to improve generalization performance.

5 Future Work & Conclusion

The goal of this report is to create an application to recognize sentences on iOS and assist handwriting so that handwriting can be on par with keyboard input. Although it is imperfect, the heuristic used for ROI detection successfully detect

a word out of a sentence, and can cut handwritten illustration out from the other part of document without requiring large amount of computation. Since current ROI detection algorithm does not parse all the trajectory and instead searches for the subsequence included in ROI from the end of the sequence iteratively for the sake of speed, it is sometimes the case that strokes that should be included in ROI intuitively are not included. Future tasks include improving this point. It should also be noted that current implementation cannot separate handwriting from handwritten illustrations, therefore text recognition process is always performed even if the user is drawing an illustration. To prevent this, I need to implement another process to check whether the output of ROI detection is an illustration or text. Putting that process into the pipeline is also a future work.

On the other hand, text recognition had a trade-off between speed and accuracy. Future tasks include improving the accuracy of text recognition without slowing it down. Since this work does not elaborate on collecting datasets or hyperparameters search, the starting point for improving the accuracy can be either.

Another example of future work is to perform more accurate text recognition and ROI detection using an online method.In auto-completion, prediction using information on surrounding words, and refinement of the technique of presenting complement candidates can also be one of future works.

References

1. Bissacco, A., Cummins, M., Netzer, Y., Neven, H.: Photoocr: Reading text in uncontrolled conditions. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 785–792 (2013)
2. Darragh, J.J., Witten, I.H., James, M.L.: The reactive keyboard: A predictive typing aid. Computer **23**(11), 41–49 (1990)
3. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
4. Gao, Y., Chen, Y., Wang, J., Lu, H.: Reading scene text with attention convolutional sequence modeling. arXiv preprint arXiv:1709.04303 (2017)
5. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd international conference on Machine learning. pp. 369–376. ACM (2006)
6. He, P., Huang, W., He, T., Zhu, Q., Qiao, Y., Li, X.: Single shot text detector with regional attention. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3047–3055 (2017)
7. Jiang, Y., Zhu, X., Wang, X., Yang, S., Li, W., Wang, H., Fu, P., Luo, Z.: R2cnn: rotational region cnn for orientation robust scene text detection. arXiv preprint arXiv:1706.09579 (2017)
8. Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M.: Deep learning for generic object detection: A survey. arXiv preprint arXiv:1809.02165 (2018)
9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016)

10. Liu, X., Liang, D., Yan, S., Chen, D., Qiao, Y., Yan, J.: Fots: Fast oriented text spotting with a unified network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5676–5685 (2018)
11. Long, S., He, X., Yao, C.: Scene text detection and recognition: The deep learning era (2018)
12. Lyu, P., Liao, M., Yao, C., Wu, W., Bai, X.: Mask textspotter: An end-to-end trainable neural network for spotting text with arbitrary shapes. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 67–83 (2018)
13. Phan, T.Q., Shivakumara, P., Su, B., Tan, C.L.: A gradient vector flow-based method for video character segmentation. In: 2011 International Conference on Document Analysis and Recognition. pp. 1024–1028. IEEE (2011)
14. Plamondon, R., Srihari, S.N.: Online and off-line handwriting recognition: a comprehensive survey. IEEE Transactions on pattern analysis and machine intelligence **22**(1), 63–84 (2000)
15. Puigcerver, J.: Are multidimensional recurrent layers really necessary for handwritten text recognition? In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). vol. 1, pp. 67–72. IEEE (2017)
16. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
17. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks (2015)
18. Zhang, C., Liang, B., Huang, Z., En, M., Han, J., Ding, E., Ding, X.: Look more than once: An accurate detector for text of arbitrary shapes (2019)
19. Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., Liang, J.: East: an efficient and accurate scene text detector. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 5551–5560 (2017)